Conception et réalisation d'un dictionnaire pour un analyseur interactif de langues naturelles

Ernest Grandjean

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL ASSOCIE

DE

G R E N O B L E

xxxxxxxxxxxx

MEMOIRE

présenté en vue d'obtenir

LE DIPLOME D'INGENIEUR C.N.A.M.

en

INFORMATIQUE

par

Ernest GRANDJEAN

xxxxxxxxx

CONCEPTION ET REALISATION D'UN DICTIONNAIRE POUR UN ANALYSEUR INTERACTIF DE LANGUES NATURELLES

xxxxxxxxx

Les travaux relatifs au présent mémoire ont été effectués à l'UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE, au LABORATOIRE D'INFORMATIQUE - LABORATOIRE ASSOCIE AU C.N.R.S. N°7 sous la Direction de Monsieur KUNTZMANN, Directeur du Laboratoire et de Monsieur BOLLIET, Professeur Directeur du mémoire.

J'exprime ma reconnaissance,

à Monsieur L. BOLLIET, Professeur à l'Institut Universitaire de Technologie de Grenoble, qui a bien voulu me faire l'honneur de présider le jury de ce mémoire et qui a toujours montré la plus grande bienveillance pour mon travail,

à Monsieur P. NAMIAN, Professeur au Conservatoire National des Arts et Métiers de Paris, qui a accepté de faire partie du jury,

à Monsieur G. VEILLON, Professeur à l'Institut National Polytechnique de Grenoble, qui fut à l'origine de ce mémoire et dont les critiques bienveillantes m'ont été précieuses,

à Monsieur J. COURTIN, Maître-Assistant à l'Institut Universitaire de Technologie de Grenoble, dont la collaboration active et efficace a été déterminante pour mener à bien les travaux rapportés dans ce mémoire,

à Monsieur D. CLAUZEL, Ingénieur à La Télémécanique Electrique à Grenoble, qui a accepté de faire partie du jury,

Ce mémoire reflète ma participation aux travaux de l'équipe "Communication Homme-Machine en Langues Naturelles" placée sous la direction de Messieurs G. VEILLON et J. COURTIN au Laboratoire d'Informatique de l'Université Scientifique et Médicale de Grenoble,

Je tiens à remercier Mesdames D. DUJARDIN et F. VEILLON, qui, par leurs suggestions et leurs conseils, m'ont été d'une aide précieuse.

Je ne saurais oublier Madame M. TREVISAN pour sa gentillesse, la patience dont elle a fait preuve, et le soin qu'elle a apporté dans la réalisation pratique de ce document.

Je remercie le service tirage du C.I.C.G. qui a assuré avec le soin habituel la réalisation matérielle de cet ouvrage.

E.G.

TABLE DES MATIERES

I N T R O D U C T I O N

L'automatisation du traitement des langues naturelles soulève de nombreux problèmes de natures linguistique et informatique. Un problème commun à la plupart de ces traitements automatiques est celui du dictionnaire car il conditionne dans une large mesure la stratégie et le rendement du système.

Le dictionnaire doit contenir le "vocabulaire" de la langue naturelle traitée afin de permettre l'identification de chaque élément constituant un texte et il doit fournir le moyen d'évaluer la fonction et la valeur grammaticale de chaque forme identifiée.

Le "vocabulaire" constitue un ensemble de données extrêmement vaste et plutôt que de construire un dictionnaire a priori, qui sera toujours inadapté, on constituera progressivement ce dictionnaire à partir d'un "corpus", ce qui implique une procédure de mise à jour interactive.

D'autre part, l'algorithme d'identification étant activé pour chaque élément d'un texte, il est naturel d'en rechercher l'efficacité maximale.

Pour une technologie donnée, pour une certaine configuration machine et pour une application souhaitée : un compromis doit souvent être trouvé entre stocker des informations dans un espace mémoire minimal mais au prix d'une efficacité de traitement faible et implanter des informations dans un volume mémoire important mais permettant un accès très rapide.

L'évaluation de chaque stratégie permet de préciser ce choix.

Nous nous sommes donc intéressé plus particulièrement à ce problème de dictionnaire en proposant une structure hybride de données et en mesurant son comportement réel.

Le travail présenté dans cette mémoire peut se résumer en 4 points :

- conception d'une organisation de données satisfaisant aux contraintes particulières liées au traitement automatique des langues naturelles.
- mesures d'efficacité des algorithmes en fonction de divers paramètres.
- élaboration d'un algorithme général de construction automatique d'un arbre d'accès à une liste.
- réalisation globale d'un ensemble de programmes s'intégrant au système interactif d'analyse des langues.

C H A P I T R E I

I.1. PRESENTATION GENERALE

Un objectif essentiel du projet de l'équipe consiste à concevoir des outils informatiques en vue de l'analyse automatique des langues naturelles.

Les applications sur ce sujet peuvent être développées dans deux directions :

- d'une part, la résolution de problèmes de nature linguistique tels que des études statistiques de la langue, l'enseignement de la langue, la validation de données linguistiques, etc...
- d'autre part, l'application à la communication homme-machine :
  - un système d'interrogation des banques de données,
  - un contrôle et une édition de textes à partir d'une reconnaissance automatique de caractères ou de la parole, etc...

L'analyse d'une langue naturelle passe par les différents niveaux rencontrés lors de la compilation (partie analyse) des langages de programmation.

Toutefois, les données linguistiques n'étant pas formalisées, nous concevons des modèles formels ainsi que les algorithmes associés pour atteindre les niveaux lexicographique, syntaxique et ultérieurement sémantique.

L'analyse morphologique consiste à reconnaître les formes (mots ou locutions) constituant les phrases du langage et à associer à ces formes des informations grammaticales (catégories et variables) qui seront les données du modèle syntaxique.

L'analyse syntaxique consiste à construire une structure de dépendances en fonction des catégories et à vérifier les accords de variables grammaticales sur chaque arc de l'arbre (voir fig. 1).

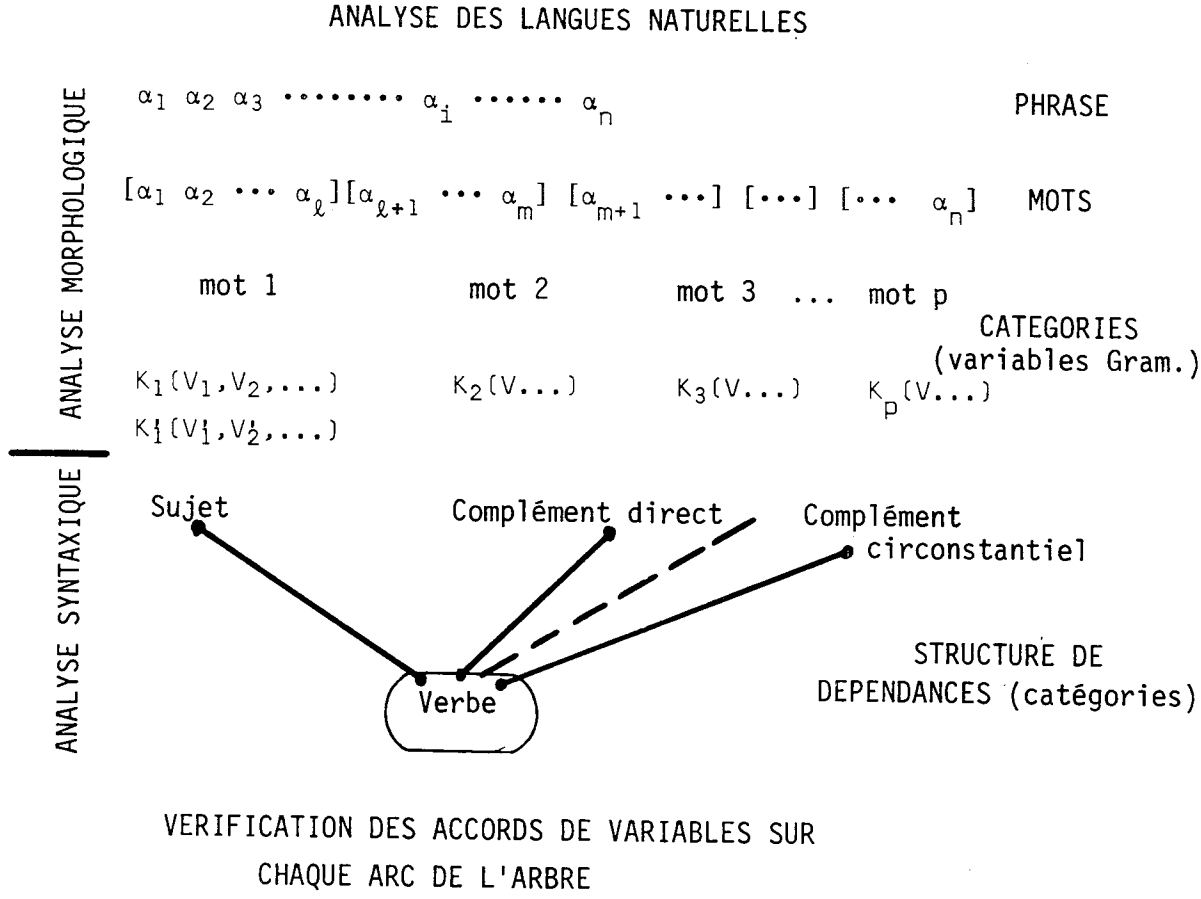


Figure 1

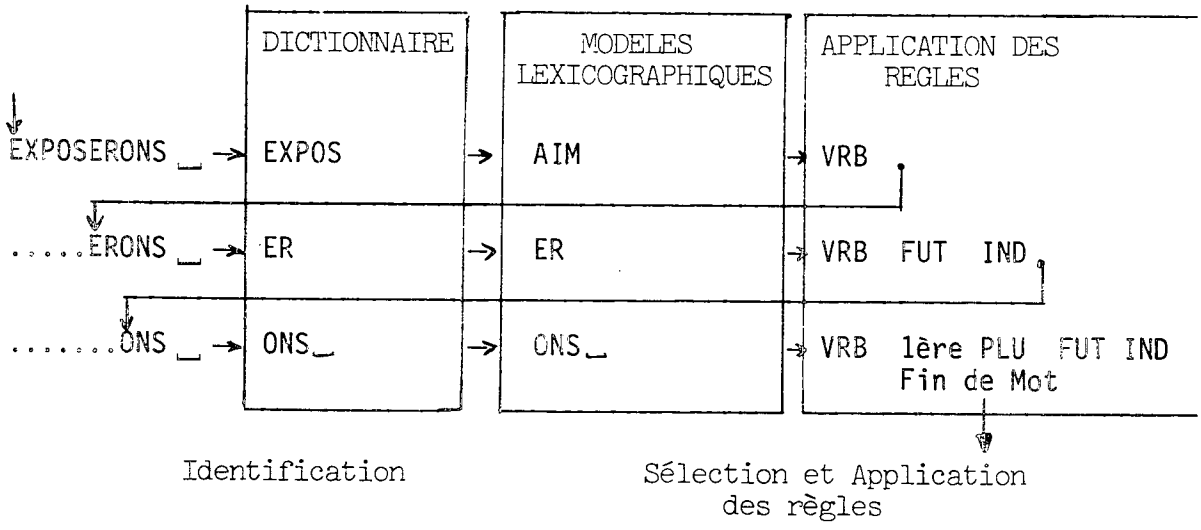
Il serait évidemment coûteux de disposer en machine d'un dictionnaire contenant toutes les formes d'une langue naturelle (verbes conjugués, noms dérivés,...).

Le modèle morphologique est alors composé d'une grammaire et d'un dictionnaire. La grammaire contient des règles qui contrôlent la composition des formes à partir des éléments contenus dans le dictionnaire. Ces éléments sont des préfixes, des racines (radicaux ou bases), des suffixes et des désinences (terminaisons). Les règles assurent également la transduction des informations grammaticales [3] [4].

Plusieurs bases ayant le même comportement morphologique constituent une classe de référence appelée "Modèle Lexicographique".

Il apparaît que le dictionnaire sera constitué d'un ensemble d'éléments faisant toujours référence à un modèle lexicographique existant (voir fig. 2).

On peut schématiser l'analyse morphologique par les opérations suivantes :



Cette stratégie nous permet de séparer la grammaire de la lexicographie d'où la possibilité d'enrichir le dictionnaire (bases essentiellement) au fur et à mesure de son utilisation.

ANALYSE MORPHOLOGIQUE

IDENTIFICATION

$\alpha_1\ \alpha_2\ \alpha_3 \cdots\cdots\cdots\cdots \alpha_\ell$ MOT DU LANGAGE

$(\alpha_1\ \alpha_2\ \alpha_3 \cdots\cdots\cdots\cdots \alpha_\ell)$ FORMES- LOCUTIONS

$(\alpha_1\ \alpha_2 \cdots\cdots\cdots \alpha_i)\ (\alpha_{i+1} \cdots\cdots\cdots\cdots \alpha_\ell)$ (RADICAL)(DESINENCE)

$(\alpha_1\alpha_2\cdots\alpha_i)\ (\alpha_{i+1}\cdots\alpha_j)(\alpha_{j+1}\cdots\alpha_k)(\alpha_{k+1}\cdots\alpha_\ell)$ (RADICAL)(SUFFIXE)...
...(SUFFIXE) (DESINENCE)

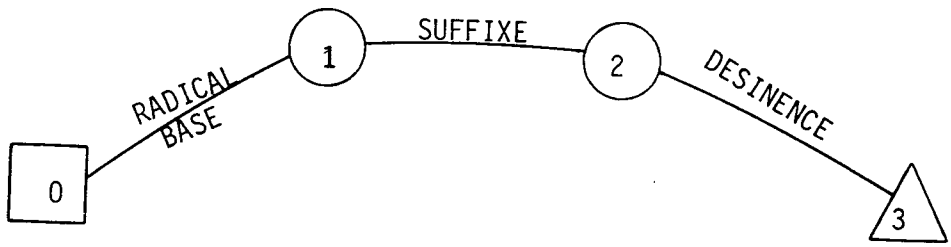
$\downarrow M_1$ $\downarrow M_2$ $\downarrow M_3$ $\downarrow M_n$

MODELE LEXICOGRAPHIQUE : REGLES DE COMPOSITION

NATURE DE LA CLASSE LEXICALE (CATEGORIE, VARIABLE)

APPLICATION DES REGLES

REGLES :



Calcul de l'état courant
Transduction des variables

CAS RENCONTRES :

- CATEGORIE 1 (Var 1, Var 2, Var 3) : Homographie Interne
- CATEGORIE 1 (Var)
  CATEGORIE 2 (Var) } Homographie Externe
- PLUSIEURS DECOUPAGES

Figure 2

Nous voulons obtenir tous les découpages possibles d'une forme (traitement des homographies), pour ce faire nous utilisons la technique du "Longest Match" qui consiste à rechercher la plus longue coïncidence possible entre la chaîne de données et les éléments du dictionnaire.

Ensuite, on pourra en déduire les chaînes plus courtes (cf. II-3).

Exemples :	COUVENT	COUVENT␣	Nom commun Masculin Singulier
		COUV+ENT␣	Verbe 1er groupe 3ème personne du présent de l'indicatif et du subjonctif

Le dictionnaire peut également contenir des locutions et des ensembles préconstruits tels que "au fur et à mesure"
"à gauche de"
"à gauche"

Du fait de cette possibilité, il n'est plus possible de considérer le caractère blanc (noté ␣) comme un séparateur de mots. On pourra stocker des expressions courantes du langage afin de minimiser l'analyse syntaxique ultérieure.

Notons enfin, que l'application de la grammaire morphologique est réversible; c'est-à-dire qu'elle peut être utilisée pour générer des formes du langage à partir des bases.

D'autre part, nous avons constaté au cours de nos expériences précédentes [7][13], qu'il était indispensable de disposer d'un système interactif facilitant les expériences et les mises au point, en particulier :

- validation des données linguistiques
- correction et extension du dictionnaire,
- mise au point des grammaires (règles et modèles)
- paramètres et options des algorithmes.

Cet aspect interactif qui est compatible avec la stratégie proposée précédemment va conditionner en priorité les choix ultérieurs pour l'organisation des données du dictionnaire.

I.2. CAHIER DES CHARGES DU DICTIONNAIRE

L'organisation des données devra être telle que l'utilisation normale du dictionnaire satisfasse aux conditions suivantes :

1°) IDENTIFICATION rapide des éléments pour garantir un traitement conversationnel (temps de réponse minimal)
2°) INTERACTION avec le DICTIONNAIRE pour assurer les mises à jour pendant l'exécution d'une analyse
3°) IDENTIFICATION de la plus longue chaîne possible en premier lieu afin d'éviter la recherche de chaînes plus courtes
4°) Le caractère blanc (␣) n'est plus considéré comme un caractère de fin de mot
5°) Rendre possible la fonction de GENERATION MORPHOLOGIQUE.

D'un point de vue pratique, nous utilisons les matériels et les logiciels standards actuellement en service au Centre Interuniversitaire de Calcul de Grenoble (CICG) : Ordinateur IBM 360/67, Système CP/CMS et Langage de programmation PL/360.

Etant donné qu'une machine virtuelle standard a une taille de 256 K octets, le dictionnaire sera placé entièrement en mémoire. En conséquence, les problèmes liés au chargement et à la pagination ont été supprimés, mais l'implantation du dictionnaire sur une petite machine est tout à fait possible et a été envisagée.

Par interaction, nous entendons de pouvoir modifier le contenu du dictionnaire, par intervention de l'utilisateur au cours de l'analyse morphologique ou syntaxique d'un texte. On obtiendra ainsi, un enrichissement progressif du vocabulaire en fonction des différents "corpus" analysés.

Les données sont de type "chaîne de caractères".

La longueur prévue varie de 1 à 24 caractères alphanumériques. L'emploi des pointeurs permet les références (indexations) aux 3 ensembles de modèles lexicographique, syntaxique et sémantique.

I.3. PREMIERE APPROCHE SUR LES STRUCTURES D'INFORMATIONS POSSIBLES

Nous avons tout d'abord cherché parmi les structures d'informations connues [1][8][9][10][11], s'il était possible d'en sélectionner une qui nous permette de concilier les différentes contraintes du cahier des charges.

La structure de données devait être telle que l'algorithme d'identification soit très efficace pendant le traitement normal d'analyse et que l'algorithme de modification (extension) des éléments du dictionnaire reste rapide en évitant les mouvements d'informations trop importants.

On ne connait pas la longueur de l'élément à identifier ($1 < \ell < 24$).

Les principales structures d'informations que sont les vecteurs, les listes chaînées et les arbres nous ont conduit à examiner quelques méthodes d'utilisation propres à chaque type.

1) Tables ordonnées. Les (N) éléments sont rangés par ordre alphabétique dans la table.

Les méthodes d'accès sont :

- la recherche séquentielle très lente ($\frac{N}{2}$)
- la recherche dichotomique, c'est la méthode la plus rapide ($\log_2 N$)

Mais la mise à jour d'une table ordonnée demande des mouvements d'informations qui peuvent être très grands dans l'application considérée. Les temps de réponse risquent d'être longs ce qui interdit pratiquement l'utilisation de cette méthode en mode interactif.

2) Tables mélées: L'utilisation d'une technique d'adressage dispersé ("Hash-Coding") est évidemment séduisante. On peut citer des fonctions d'adressage telles que :

- le carré du nombre binaire que représente la suite de lettres
- l'adressage par la 1ère et la 3ème lettre
- l'ensemble des lettres impaires.
- etc...

Les extensions et mises à jour sont possibles en mode interactif. Il s'agit seulement de calculer la clé en fonction de la valeur de l'élément, de parcourir des pointeurs et de les modifier. Mais, au moment de l'identification, nous ne connaissons que le début de la chaîne de caractères (phrase ou reste de phrase). La longueur de l'élément recherché est inconnue et par suite le calcul de la clé est impossible sauf si l'on prend comme fonction de "hash-code" la première lettre. Cette solution n'a pas été retenue car il est bien connu que les classes d'équivalence obtenues, par un tel procédé, sont disproportionnées.

3) Arborescences :

Les éléments du dictionnaire peuvent être écrits sous la forme d' "ARBRES". Le parcours est réalisé en fonction de chaque lettre du mot cherché, [12] (Analogie avec le "hash code" précédent).

La méthode d'accès a l'avantage d'être rapide et adaptée au problème, mais le volume est trop important et la mise à jour est coûteuse.

4) Listes chaînées : Tous les éléments du dictionnaire sont chaînés entre eux en respectant l'ordre alphabétique. La mise à jour (ajout, suppression, remplacement) est tout à fait possible en mode interactif.

L'identification des éléments peut se réaliser par une méthode dichotomique sur un vecteur d'adresses, suivie d'un appel séquentiel. Il sera nécessaire de réorganiser fréquemment la liste et le vecteur d'adresses afin de conserver des classes d'équivalence de même taille.

C'est à partir de cette dernière solution que nous avons conçu la méthode décrite au chapitre II dans laquelle nous tenons compte du fait que les éléments du dictionnaire ont des fréquences d'appel variables.

C H A P I T R E II

STRATEGIE CHOISIE

II.1. ETUDE DES DONNEES : Pour organiser le dictionnaire.

Nous avons utilisé les résultats d'une enquête statistique sur la langue française (parlée) en vue de la définition d'un vocabulaire de base du français fondamental.

Nous avons constaté que les mots de liaisons (articles, prépositions, conjonctions, pronoms, etc...) et les formes des auxiliaires (avoir, être, sembler,...) avaient une fréquence d'utilisation très élevée par rapport aux verbes, noms, adverbes et adjectifs.

Ces mots de caractère général (liaisons et auxiliaires) représentent près de 30 % du discours et sont généralement invariables. C'est-à-dire qu'ils seront indexés par leur forme complète dans le dictionnaire et ils seront peu sujets à des modifications ultérieures.

Nous avons donc cherché à utiliser cette propriété intéressante des données pour organiser le dictionnaire. D'une manière générale, les mots d'une langue naturelle suivent la loi de ZIPF, [9 vol.3] c'est-à-dire que si l'on considère l'ensemble des mots ordonnés par fréquence décroissante, le mot de rang n a une fréquence $\frac{1}{n}$.

On trouvera en annexe chapitre II, la liste des mots classés par fréquences décroissantes triés du livre de Sauvageot et Michéa [6].

II.2. PRINCIPE

Nous avons utilisé une structure de données hybride qui nous permet de tenir compte des contraintes du cahier des charges.

Le dictionnaire est constitué de 2 ensembles de données ayant des structures différentes.

1) Un arbre binaire, composé d'éléments fréquents et ordonnés alphabétiquement, permet l'accès rapide à ces éléments et fournit des points d'entrée sur la seconde partie.

2) Une liste chaînée composée de l'ensemble des éléments ordonnés alphabétiquement permet l'accès par balayage aux éléments moins fréquents à partir du point d'entrée précédent.

- L'ordre séquentiel permet de satisfaire à la contrainte du "Longest Match",
- Le chaînage assure la mise à jour interactive,
- Toute modification se traduit par la modification d'un pointeur et l'utilisation d'une seule zone de débordement contiguë
- Cette structure à 2 niveaux est un certain type de "Hash-code"
- L'arbre binaire détermine des classes d'équivalence sur la liste chaînée.

Evaluation de cette stratégie :

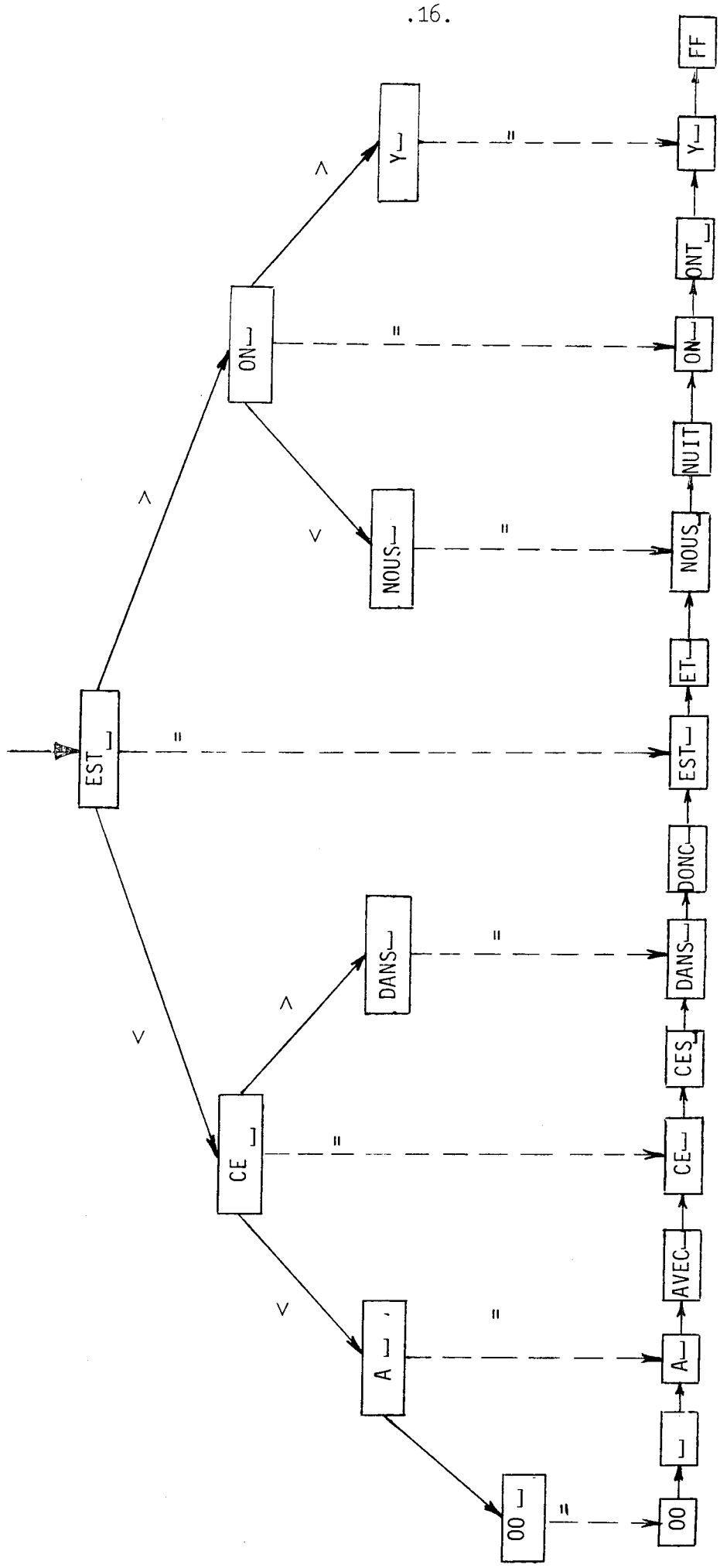
Si n est le nombre d'éléments du dictionnaire

Si p est le nombre d'éléments de l'arbre.

Le parcours de l'arbre est proportionnel, en moyenne à $\text{Log}_2\ p$

et le parcours moyen d'une classe est proportionnel à $\frac{n}{2p}$ à condition que le nombre d'éléments des classes soit identique et égal à $\frac{n}{p}$

et que la fréquence d'appel des éléments soit équiprobable.


EST_
>
=
<
ON_
Y_
NOUS_
EST_
ET_
NOUS_
NUIT
ON_
ONT_
Y_
FF
CE_
DANS_
DONC
CES_
CE_
AVEC
A_
_
OO

II.3. STRATEGIE GENERALE DU DICTIONNAIRE

Il n'était pas possible d'utiliser l'algorithme de KNUTH [9][10] sur l'ensemble des éléments du dictionnaire (2000 éléments actuellement) étant donné les coûts de construction de l'arbre proportionnel à n^2 en temps et en volume d'une part, et étant données les contraintes d'interaction, d'autre part, car l'ajout ou la suppression d'un élément implique la reconstruction de l'arbre complet.

Mais par analogie avec l'exemple de KNUTH [10], nous avons la possibilité de choisir les mots de caractère général de sorte que :

- la fréquence α_i de ces mots soit élevée,
- leur nombre permette l'application de l'algorithme de KNUTH dans l'espace mémoire disponible. Actuellement, il est possible de construire un arbre binaire de 120 noeuds, ce qui donne des classes d'une taille inférieure à 20 éléments.

Au départ, nous avions peu d'informations sur les fréquences β_i des classes. Un premier passage d'identification d'un échantillon de textes avec comptage nous permet de les obtenir.

Il nous reste à satisfaire la condition (3) d'obtention de la plus longue coïncidence entre la chaîne d'entrée et le contenu du dictionnaire.

Organisation de la seconde partie du dictionnaire. Liste chaînée [2]

Définition : Soit V un ensemble fini appelé vocabulaire.

avec V = {␣ ,.␣ ,?␣,...,A,B,C,...,Z,0,...9}

␣ : désigne le caractère blanc

On appelle chaîne ou "élément" sur V une suite finie de symboles $X_1\ X_2\ X_3\ \ldots\ X_n$ avec $X_i \in V$

1) On définit la relation INFERIEURE notée "<" (relation partielle)

Soit 2 éléments α et β de V

avec $\alpha = X_1 X_2 \ldots X_n$ $\quad n > 1 \quad$ $X_i \in V$

$\beta = Y_1 Y_2 \ldots Y_m$ $\quad m > 1 \quad$ $Y_i \in V$

On dit que $\alpha < \beta$ si $X_1 < Y_1$

ou s'il existe un i $(1 < i < n-1)$ tel que

$\beta = X_1 X_2 \ldots X_i \; Y_{i+1} \ldots Y_m$ avec $X_{i+1} < Y_{i+1}$

Quand cette relation est vérifiée on dit que β est un élément frère de α

Exemple : Frère (PARTIEL) = PARTIES

Soit α et β : 2 éléments frères.

On dit que β est frère direct de α si $\alpha < \beta$ et s'il n'existe pas d'élément frère γ tel que $\alpha < \gamma < \beta$

2) De même, on définit la relation CONTENUE notée "C" (relation partielle).

Soit 2 éléments α et β de V

avec $\alpha = X_1 X_2 \ldots X_n$ $\quad X_i \in V$

$m > n > 1$

$\beta = Y_1 Y_2 \ldots Y_m$ $\quad Y_i \in V$

On dit que $\alpha \; C \; \beta$ si $\beta = \alpha \; Y_{n+1} \ldots Y_m$

Quand cette relation est vérifiée, on dit que α est un élément fils de β

Exemple : Fils(PARTIEL) = PARTIE, PARTI, PART

On dit que α est fils direct de β

si $\alpha \in$ Fils(β)

et s'il n'existe pas d'élément $\gamma \in$ Fils(β) tel que $\alpha \in$ Fils (γ)

3) On dira qu'un élément α du dictionnaire(D) est un "élément fils" s'il existe un élément β de D tel que α C β

Les autres éléments de D seront appelés "élément frères"

On peut ainsi organiser les chaînages entre les éléments de la seconde partie du dictionnaire en tenant compte des relations "frère direct" et "fils direct" pour satisfaire la condition (3).

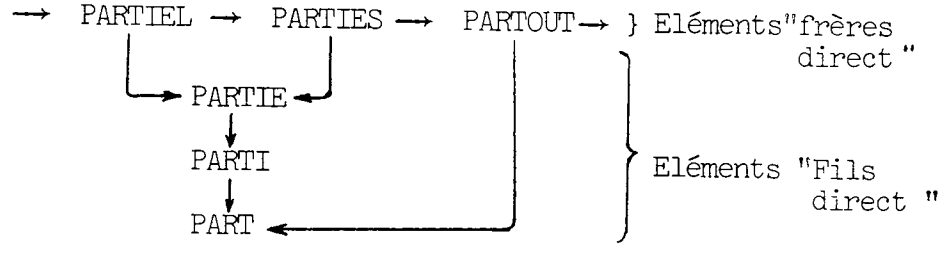
La liste chaînée sera réalisée entre les éléments "frères direct"

Chaque élément "frère" contient un pointeur sur son "frère direct" et un pointeur sur son "fils direct"

Ce 2ème pointeur nous permet d'accèder à une sous-liste chaînée qui est réalisée entre les éléments "fils direct".

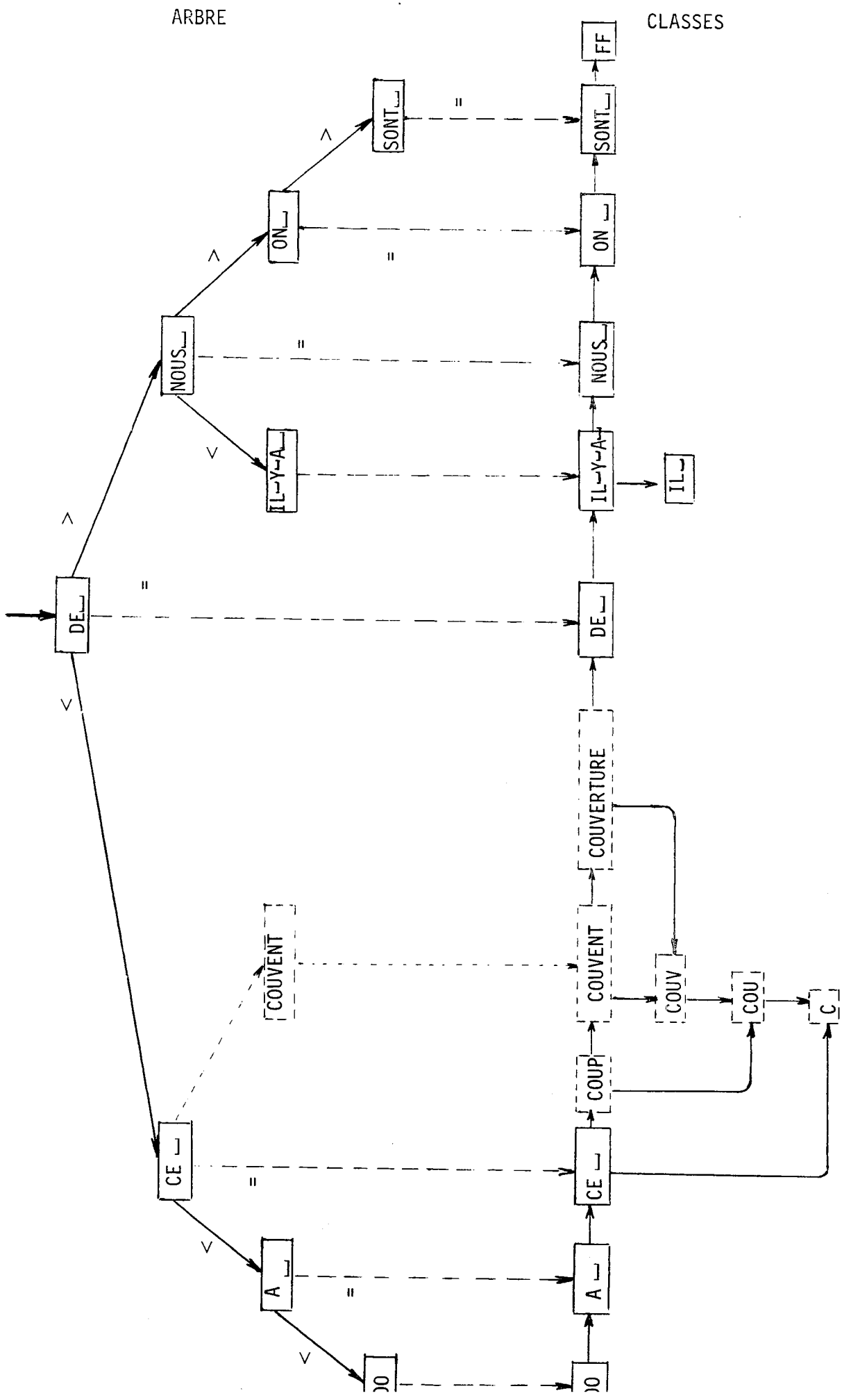
Chaque élément "fils" contient un pointeur sur son "fils direct". Il y a autant de pointeurs de sous-listes que d'éléments "frères direct".

Exemple :



CONDITION :

Afin de s'assurer un ensemble continu sur les éléments "frères directs", les nœuds de l'arbre sont toujours des éléments "frères" et ils sont simplement recopiés dans les 2 ensembles. (Recherche d'éléments "fils" aux limites de classes d'équivalence). L'augmentation de volume est faible si on se limite à un arbre binaire de 120 éléments.


ARBRE
CLASSES
DE
NOUS
ON
SONT
IL Y A
CE
A
DO
COUVENT
COUVERTURE
COUP
COUV
COU
C
FF
IL

II.4. ALGORITHME D'IDENTIFICATION ET ALGORITHME DE RECHERCHE

1) **L'algorithme d'identification** est activé par l'analyseur morphologique.

Les données constituent une chaîne de caractères dont on connaît seulement le rang du 1er caractère.

. La longueur de la sous-chaîne à identifier est inconnue.

En effet, on peut identifier indifféremment une base, un suffixe, une désinence ou une locution complète.

Le blanc n'est plus considéré comme séparateur de mots.

. Les comparaisons entre la chaîne d'entrée et les éléments du dictionnaire (arbre, frères ou fils) sont effectuées sur la longueur indiquée au début de chaque élément du dictionnaire.

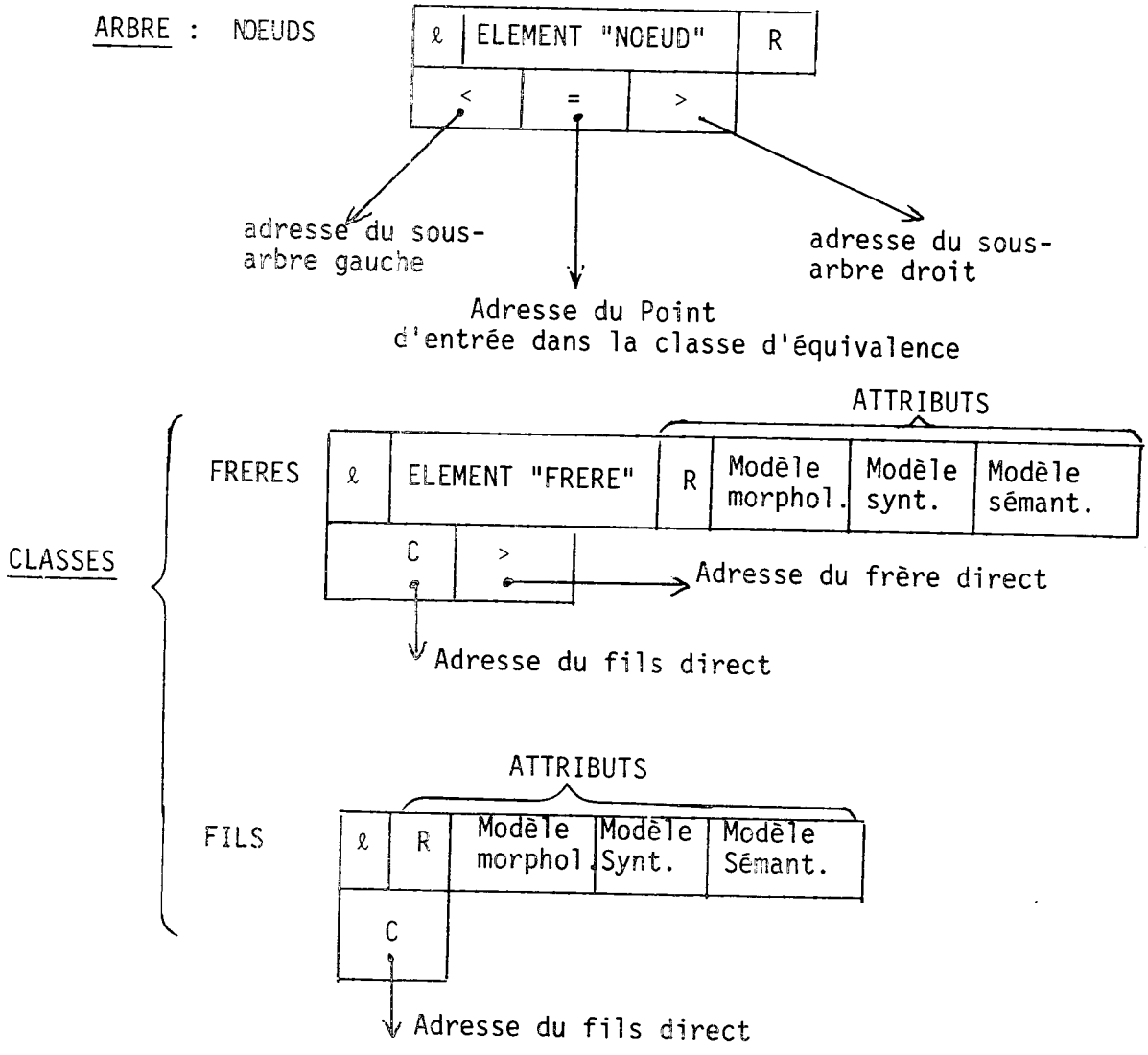
2) **L'algorithme de recherche** est activé par le programme d'extension et de mise à jour.

. Les données sont des chaînes de caractères de longueur connue.

. Les comparaisons s'effectuent sur le minimum des 2 longueurs (données-dictionnaire).

La méthode de parcours est sensiblement identique pour les 2 algorithmes.

II.5. STRUCTURE DES ELEMENTS DU DICTIONNAIRE



Chaque chaîne de caractères représentant un élément "noeud" ou "frère" est précédé de sa longueur (ℓ). Dans le cas des éléments "fils", seule la longueur est indiquée.

Les éléments des classes sont complétés par une zone ATTRIBUTS contenant les adresses de références aux 3 modèles morphologique, syntaxique et sémantique et un octet de renseignements indiquant la nature de l'élément et permettant certaines optimisations (indicateur noté *) au moment de la recherche des éléments "fils" en morphologie (voir exemples de parcours).

L'algorithme de parcours découle naturellement de cette structure de données (les schémas de programme sont donnés au paragraphe II.8).

1) Parcours de l'arbre binaire

Le point d'entrée est la racine de l'arbre.

La comparaison entre la chaîne d'entrée et le noeud, sur la longueur ℓ, nous indique :

- soit un point d'entrée dans la classe d'équivalence
  si le résultat est =
- soit un nouveau parcours de l'arbre
  si > → sous-arbre droit - conserver adresse courante
  si < → sous-arbre gauche.

Dans le cas où l'adresse de parcours est nulle; le dernier noeud pour lequel la comparaison avait fourni le résultat >, contient l'adresse du point d'entrée de la classe d'équivalence.

2) Parcours de la classe d'équivalence

Le point d'entrée est donné par le parcours de l'arbre. Tant que la comparaison entre la chaîne d'entrée et l'élément "frère", sur la longueur ℓ, indique un résultat >, on parcourt les éléments "frères" (adresse frère direct).

Quand on trouve égalité, on obtient les attributs de cet élément et l'adresse de son fils direct, donc de l'ensemble de ses fils.

Sinon, il faut chercher à identifier une chaîne plus courte. Elle appartient : - soit à l'ensemble des fils du "frère" courant,
- soit à l'ensemble des fils du "frère" précédent.

Le parcours de ces 2 ensembles de fils utilise la même technique que pour les éléments "frères".

Quand on trouve égalité, la plus longne chaîne trouvée est le résultat avec ses attributs.

Sinon, il n'y a aucune coïncidence entre la chaîne d'entrée et le dictionnaire.

Exemples de parcours :

1) Soit la chaîne d'entrée : COUVENT␣DE␣JEUNES␣FILLES␣ ...

racine

.Parcours de l'arbre : /COU/ < /DE␣/ → Appel du sous-arbre gauche

/COU/ > /CE␣/ → Appel du sous-arbre droit

Il est nul - appel du dernier > : /CE␣/ → Appel de la classe "CE␣"

.Parcours de la classe : /COU/ > /CE␣/ → Appel frère direct

/COUV/ > /COUP/ → Appel frère direct

/COUVENT/ = /COUVENT/ O.K.

Appel du fils /COUV/

2) Soit la chaîne : COUVONS␣␣␣

.Parcours de l'arbre : identique

.Parcours de la classe : /COU/ > /CE␣/

/COUV/ > /COUP/

/COUVONS/ > /COUVENT/

/COUVONS␣␣␣/ > /COUVERTURE/

/COU/ < /DE␣/ → Appel fils direct de /DE / = 0

Appel fils direct de /COUVERTURE/

/COUV/ = /COUV/ O.K.

Il y a un indicateur * sur /COUV/ donc pas d'appel du fils /COU/

3) Soit la chaîne : COUS ␣␣

.Parcours de l'arbre : identique

.Parcours de la classe : /COU/ > /CE␣/
/COUS/ > /COUP/
/COUS..../ < /COUVENT/ a) appel fils de /COUVENT
b) appel fils de /COUP/
a) /COUS/ < /COUV/
/COU/ = /COU/ 3 lettres
=
b) /COU/ = /COU/ 3 lettres
} O.K.

4) Soit la chaîne : IL␣Y␣A␣

.Parcours de l'arbre : /IL␣/ > /DE␣/
/IL␣Y␣/ < /NOUS␣/
/IL␣Y␣A␣/ = /IL␣Y␣A␣/

.Parcours de la classe : /IL␣Y␣A/ = /IL␣Y␣A␣/ O.K.

Généralement, il y a un indicateur (*) associé à la locution IL␣Y␣A␣ donc pas d'appel du fils direct /IL␣/.

On évite la génération de structures ou de solutions parasites.

5) Soit la chaîne : DONC␣␣

.Parcours de l'arbre : /DON/ > /DE␣/
/DONC␣/ < /NOUS␣/
/DONC␣␣/ </IL␣Y␣A␣/ Pas de sous-arbre gauche
appel de l'adresse du dernier >:/DE␣/

.Parcours de la classe : /DON/ > /DE␣/

On voit que le noeud le plus à gauche de l'arbre doit être égal au 1er frère des classes.

Cette condition est réalisé par le programme au moment de la création du dictionnaire initial.

L'arbre initial contient un seul noeud ayant pour valeur 00.

Les classes contiennent 2 éléments "frères" de valeur 00 (borne minimum) et de valeur FF (borne maximum).

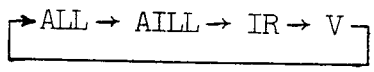
Il est possible de stocker toutes les valeurs possibles de caractères telles que les signes de ponctuation, le blanc, les chiffres, ... entrées à partir d'un clavier et également toutes valeurs non nulles créées par programme.

POINTEURS LOGIQUES

La génération morphologique est assurée par l'existence de pointeurs "logiques".

Ces pointeurs assurent un chaînage circulaire entre plusieurs éléments "frères" ou "fils" qui appartiennent à un même ensemble logique dans une application linguistique.

Exemple : On "chainera" entre elles les diverses bases du verbe aller.



La génération morphologique du verbe aller, nous fournira les résultats:

ALLons,ALLez ...

AILLes,...

IRa,IRons,...

Va,Vont,...

D'autres familles peuvent être imaginées telles que synonymes, homophones, articles, pronoms, ...

Le nombre de pointeurs logiques est un paramètre du programme, il est constant pour une application donnée.

II.6. ALGORITHMES D'EXTENSION et de mise à jour du dictionnaire

On ne connaît pas d'algorithme qui permette d'ajouter, remplacer ou supprimer un noeud de l'arbre binaire en conservant l'optimalité. Nous sommes obligés d'appliquer globalement l'algorithme de Knuth.

Mais les modifications de l'arbre sont peu fréquentes et l'algorithme est relativement rapide (1 seconde pour 100 noeuds).

Nous avons rencontré 2 types de difficultés à ce niveau :

1) La réorganisation de l'arbre demande un volume mémoire important, elle ne peut être réalisée au cours d'une application morphologique.
2) Le choix des noeuds est manuel et demande une certaine connaissance du mécanisme d'identification.

Nous verrons au chapitre III comment nous avons supprimé ces inconvénients.

1) Extension de l'arbre

Le mécanisme est le même que celui des classes, car en pratique ce sont les éléments "frères" pointés par l'arbre qui sont modifiés. Ils sont porteurs d'un indicateur signifiant leur appartenance à l'arbre. Ils seront pris en compte par l'algorithme de Knuth pour la réorganisation réelle de l'arbre ultérieurement.

Cette stratégie nous permet de réaliser des modifications du contenu de l'arbre au cours de sessions différentes et de ne faire qu'une seule réorganisation générale.

2) Extension des classes

Cet algorithme nous permet d'ajouter, remplacer ou supprimer des éléments "frères" ou "fils" en tenant compte des relations "inférieure" ou "contenue". La longueur des chaînes de données est connue.

- On utilise donc l'algorithme de recherche.
- On modifie les différents pointeurs "frère direct" et "fils direct" après avoir rangé, dans la zone de débordement, le nouvel élément ou effacé un ancien élément.
- On modifie les pointeurs logiques.
- Quand il s'agit seulement d'un remplacement : la modification concerne les attributs et/ou les pointeurs logiques.
- On utilise donc les techniques de parcours de listes et de modification des chaînages
- Le nombre de cas de figure est très varié mais ne présente pas de difficultés particulières.

Cet algorithme est rapide et il demande peu de place en mémoire, il est appliqué lors d'un arrêt en fin de mot ou d'une erreur pendant l'analyse morphologique.

Quand cette erreur provient du dictionnaire - généralement une base non indexée - le langage de commande et la stratégie adoptée dans la conception de l'éditeur permettent à l'utilisateur d'indiquer une Equivalence Morphologique et de demander ensuite (si l'utilisateur est satisfait) l'indexation Automatique de la base - c'est-à-dire l'ajout automatique d'un nouvel élément "frère" ou "fils" dans le dictionnaire.

Cette stratégie nous permet de commencer l'édition d'un nouveau corpus avec un dictionnaire réduit, ne contenant que les éléments du vocabulaire général.

L'enrichissement du dictionnaire est réalisé progressivement de manière semi-automatique.

Les mots rares et les noms propres ne seront indexés dans le dictionnaire que si leur fréquence d'appel le justifie. On évite ainsi des encombrements inutiles.

Avec ce type de fonctionnement, le volume des classes va bien entendu augmenter car on effectue essentiellement des ajouts, quelques remplacements et suppressions (voir la solution proposée au chapitre III).

II.7. ALGORITHMES DE REORGANISATION

1) Réorganisation de l'arbre

- Il s'agit de balayer l'ensemble des éléments "frère" et de sélectionner ceux qui possèdent l'indicateur d'appartenance à l'arbre.

- Puis d'appliquer à ces noeuds, auxquels on a pu associer des fréquences (α et β) par comptage automatique ou manuel, l'algorithme de KNUTH.

Un nouvel arbre est généré sur un nouvel espace mémoire.

2) Réorganisation des classes

Cet algorithme consiste à balayer l'ensemble des éléments "frères" et "fils", en parcourant les chemins possibles, et à recopier ceux-ci sur un nouvel espace mémoire de telle sorte que les nouveaux pointeurs obtenus indiquent des adresses croissantes par rapport à leur position. Il reste à réaliser la liaison arbre-classes en indiquant les nouvelles adresses des classes dans le nouvel arbre.

Optimisation

- Il est possible d'introduire manuellement les fréquences α et β des noeuds au cours de l'extension de l'arbre. Ce qui permet d'initialiser un premier arbre (ce que nous avons fait avec les statistiques sur la langue parlée).

- Il est ensuite possible de compter automatiquement ces fréquences α et β dans l'arbre pendant le traitement d'un échantillon de corpus.

On peut alors réorganiser le dictionnaire et activer l'analyse du corpus complet.

Nous avons fait des mesures de coût (voir chapitre III).

II.8. PRESENTATION SOUS FORME DE SCHEMAS DE PROGRAMME [5]

ANALYSE DESCENDANTE DU SYSTEME

ACTION ANALYSER UN TEXTE

```
- LIRE UNE PHRASE
- TANT QU'IL RESTE UNE PHRASE
  FAIRE . ANALYSER LA PHRASE
  |     . LIRE UNE PHRASE
  FIN FAIRE
```

FIN ACTION

ACTION LIRE UNE PHRASE

```
- EFFACER CHAINE LOGIQUE
- LIRE UN CARACTERE
- TANT QUE NON FIN DE PHRASE
  FAIRE - RANGER LE CARACTERE
  |       dans la CHAINE LOGIQUE
  |     - LIRE UN CARACTERE
  FIN FAIRE
```

FIN ACTION

ACTION ANALYSER LA PHRASE

```
- Etat initial (1er caractère)
- TANT QUE NON FIN DE CHAINE LOGIQUE
  FAIRE - ANALYSE MORPHOLOGIQUE
  |     - AVANCER D'UN CARACTERE
  FIN FAIRE
- ANALYSE SYNTAXIQUE
```

FIN ACTION

ACTION ANALYSE MORPHOLOGIQUE

- IDENTIFIER UN MOT DU LANGAGE
- VERIFIER QU'IL Y A AU MOINS UN RESULTAT
  SI OUI RANGER LES RESULTATS (MOT - CATEGORIE - VARIABLE)
  SI NON INTERRUPTION MORPHOLOGIE et REPRISE ANALYSE MORPHOLOGIQUE après chaque commande

FIN ACTION

ACTION IDENTIFIER UN MOT DU LANGAGE

- ETAT INITIAL
- TANT QUE NON FIN DE MOT
  FAIRE - IDENTIFIER UNE CHAINE
  - (RECHERCHER la plus longue coïncidence entre un élément du dictionnaire et le début de la chaîne courante)
  - SELECTIONNER et APPLIQUER une REGLE MORPHOLOGIQUE
  - TRANSDUCTION DES RESULTATS
  - EVALUER L'ETAT COURANT
  FIN FAIRE

FIN ACTION

ACTION INTERRUPTION MORPHOLOGIE

- Commande DICT : INTERACTION avec le dictionnaire (ajout, suppression, interrogation)
- Commande EQUIV : EQUIVALENCE MORPHOLOGIQUE (fournir des résultats sans indexer)
- Commande INDEXER : Ajout automatique de la base définie par EQUIV
- Commande SUPPRIMER : Suppression automatique de la base définie par EQUIV
- Commande ORTHO : Modification du mot courant (correction de fautes de frappe)
- Commande Vide (R.C.) : Passage au mot suivant (Avancer au 1er blanc)

FIN ACTION

ACTION IDENTIFIER UNE CHAINE

```
Adr Noeud ← Adr. Racine
TANT QUE Adr.Noeud ¬= 0 ET Chaîne  ¬= Noeud
FAIRE
     PARCOURS DE L'ARBRE
FINFAIRE
SI adr. Noeud = 0
ALORS Adr. Frère ← Adr. dernier Noeud(>)
|    TANT QUE  chaîne > frère
|    FAIRE  PARCOURS DES FRERES
|
|    FIN FAIRE
|    SI Chaîne < frère
|    ALORS PARCOURS des FILS  du frère courant
|    |     PARCOURS des FILS  du frère précédent
|    |     MAXI des 2 longueurs
|    FINSI Adr ← Adr Maxi
FINSI
Adresse Elément ← Adr
```

FIN ACTION

ACTION PARCOURS DE L'ARBRE

```
SI Chaîne > Noeud
ALORS  Adr. dernier Noeud(>) ← Adr. Noeud
|      Adr. Noeud ← Adr (>)
SINON  Adr. Noeud ← Adr (<)
FINSI
```

FIN ACTION

ACTION REORGANISATION du DICTIONNAIRE

```
LIRE les noeuds de l'arbre et
      leurs fréquences α et β
CONSTRUIRE L'ARBRE BINAIRE (alg. KNUTH)
ORGANISER l'ensemble des classes
```

FINACTION

ACTION CONSTRUIRE L'ARBRE BINAIRE

```
REMPLIR la matrice des fréquences α des noeuds(P)
CONSTRUIRE la matrice des fréquences cumulées(W)
DETERMINATION du coût mini des sous-arbres(P)
                et
CONSTRUCTION de la matrice des racines(R)
CREER L'ARBRE BINAIRE OPTIMAL
```

FINACTION

ACTION CREER L'ARBRE BINAIRE OPTIMAL

```
APPELER la racine de l'arbre(ou d'un sous-arbre)(R)
APPELER la branche gauche
 - CREER ARBRE BINAIRE
APPELER la branche droite
 - CREER ARBRE BINAIRE
```

FINACTION

ANNEXE : CHAPITRE II

LISTE DES MOTS PAR FREQUENCES DECROISSANTES

N°d'ordre	Mots	Fréquence
1	être	14.083
2	avoir	11.552
3	de	10.503
4	je	7.905
5	il(s)	7.515
6	ce	6.846
7	la	5.374
8	pas	5.308
9	à	5.236
10	et	5.082
11	le	4.957
12	on	4.266
13	vous	4.202
14	un	4.188
15	çà	3.972
16	les	3.815
17	que	3.537
18	ne	3.283
19	faire	3.174
20	qui	3.096
21	oui	2.935
22	alors	2.854
23	une	2.780
24	mais	2.768
25	des	2.646

N° d'ordre	Mots	Fréquence
26	elle(s)	2.462
27	en	2.405
28	dire	2.391
29	y	2.391
30	pour	2.076
⋮		
60	enfin	1.001
61	par	965
62	quand	964
63	le	894
64	vouloir	881
65	petit	863
66	si	837
67	plus	832
68	même	810
69	sur	801
70	ce	705
⋮		
100	trouver	439
101	quoi	437
102	ma	432
103	grand	428
104	temps	426
105	donner	426
106	après	425
107	fois	423
108	eh bien !	417
109	te	413
110	an	407
⋮		

N° d'ordre	Mots	Fréquence
:		
200	pendant	181
201	école	178
202	acheter	178
203	français	178
204	près	176
205	laisser	175
206	quelques	175
207	écouter	175
208	entendre	174
209	un	171
210	gros	165
:		
300	cinquante	101
301	sentir	101
302	presque	99
303	idée	99
304	tomber	99
305	loin	99
306	vers	99
307	puisque	98
308	argent	98
309	train	98
310	vendre	98
:		
400	autrement	73
401	cas	73
402	partie	73
403	vite	73
404	fait	72
405	simplement	72
406	porter	72
407	quelquefois	72
408	gagner	72
409	plusieurs	71
410	normal	71
:		

N° d'ordre	Mots	Fréquence
500	tout à l'heure	55
501	état	55
502	cuisine	55
503	préparer	55
504	eh	55
505	chien	55
506	médecin	55
507	en train de	54
508	enlever	54
509	dehors	54
510	porte	54

C H A P I T R E III

THEORIE ET MESURES SUR LES ALGORITHMES D'ORGANISATION DE L'ARBRE

III.1. ALGORITHME DE KNUTH [9] [10]

Si A_i est l'ensemble des Noeuds de l'arbre.

Si B_i est l'ensemble des éléments du dictionnaire n'appartenant pas à l'arbre.

Cet algorithme permet de définir un arbre binaire qui minimise le temps de recherche global des éléments A_i en fonction de :

- l'ordre alphabétique des éléments A_i : $A_i < A_{i+1}$
- la fréquence α_i des éléments A_i
- la fréquence β_i des éléments B_i

tels que : $A_i < B_i < A_{i+1}$ pour i = 1,2,...,n-1 (ordre alphabétique)

et avec $\left\{ \begin{array}{c} B_0 < A_1 \\ \\ A_n < B_n \end{array} \right\}$ aux limites

Le coût de recherche total est :

$$C = \sum_{i=1}^{n} \ell(A_i) \cdot \alpha_i + \sum_{i=0}^{n} \ell(B_i) \cdot \beta_i$$

avec $\ell(A_i)$: Niveau du noeud A_i dans l'arbre

$\ell(B_i)$: Niveau de la classe de B_i

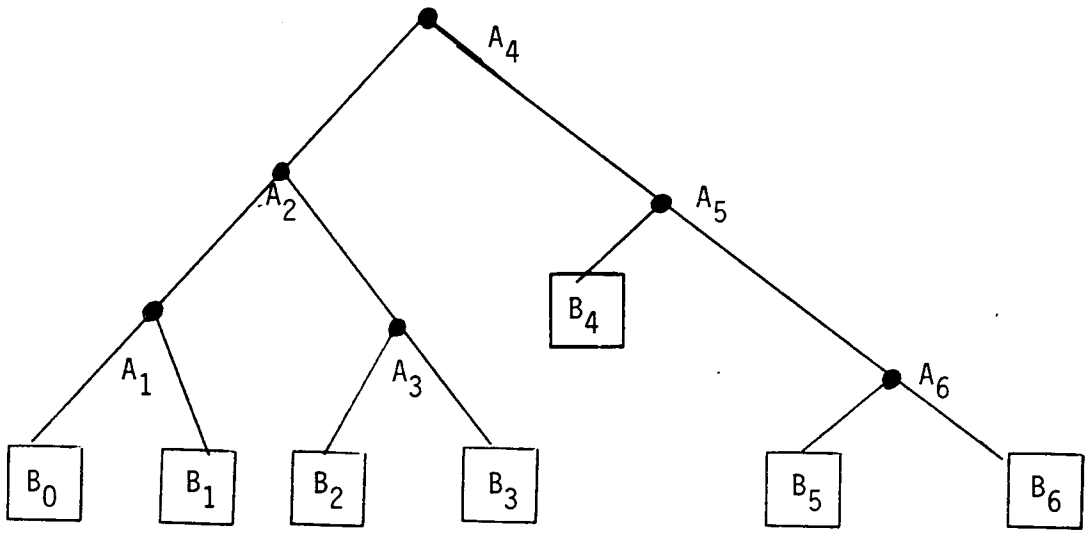
On montre que le coût C = Cg + Cd + S

Avec Cg : coût du sous-arbre gauche

Cd : coût du sous-arbre droit

S : somme des fréquences $\alpha_i + \beta_i$

ce qui donne sur un exemple :



$$C = 3.\alpha_1 + 2.\alpha_2 + 3.\alpha_3 + \alpha_4 + 2.\alpha_5 + 3.\alpha_6 + 4.\beta_0 + 4.\beta_1 + 4.\beta_2 + 4.\beta_3 + 3.\beta_4 + 4.\beta_5 + 4.\beta_6$$

$$Cg = 3.\beta_0 + 2.\alpha_1 + 3.\beta_1 + \alpha_2 + 3.\beta_2 + 2.\alpha_3 + 3.\beta_3$$

$$Cd = 2.\beta_4 + \alpha_5 + 3.\beta_5 + 2.\alpha_6 + 3.\beta_6$$

$$S = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 + \alpha_6 + \beta_0 + \beta_1 + \beta_2 + \beta_3 + \beta_4 + \beta_5 + \beta_6$$

Il faut construire un arbre tel que C soit minimum

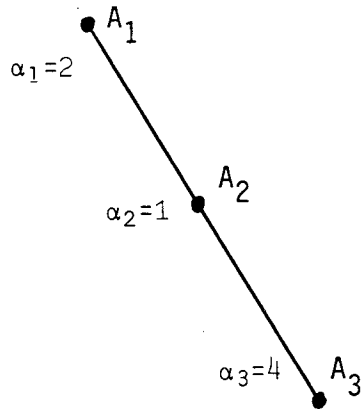
$$\min(C) = \min(Cg + Cd) + S$$

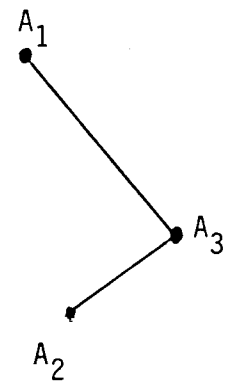
Les sous-arbres gauche et droit d'un arbre possédant un coût minimum, ont un coût minimum.

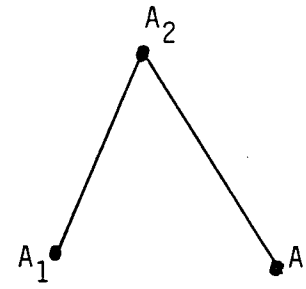
Le principe de l'algorithme est donc de construire les sous-arbres ayant un coût minimum, en partant de gauche à droite et de bas en haut, jusqu'à l'obtention de l'arbre complet.

L'algorithme de parcours est analogue à celui qui a été utilisé dans l'analyse syntaxique ascendante - Algorithme de Coke [9]

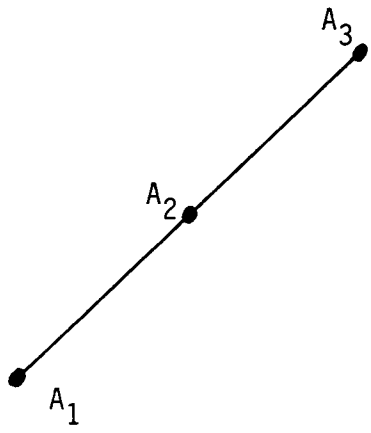
Quelques exemples de recherche de l'arbre de coût minimal avec 3 noeuds A_1, A_2, A_3 de fréquences $\alpha_1 = 2$, $\alpha_2 = 1$ et $\alpha_3 = 4$ et avec $\beta = 0$



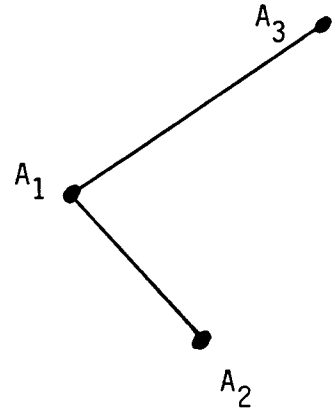




C=(2×1)+(1×2)+(4×3)=16 C=(2×1)+(4×2)+(1×3)=13 C=(1×1)+(2×2)+(4×2)=13



C=(4x1)+(1x2)+(2x3) = 12



C=(4x1)+(2x2)+(1x3) = 11

arbre de coût minimal

EVALUATION DE CET ALGORITHME

Le temps de calcul est proportionnel à n^3 : $(\frac{n(n+1)(n+2)}{6})$

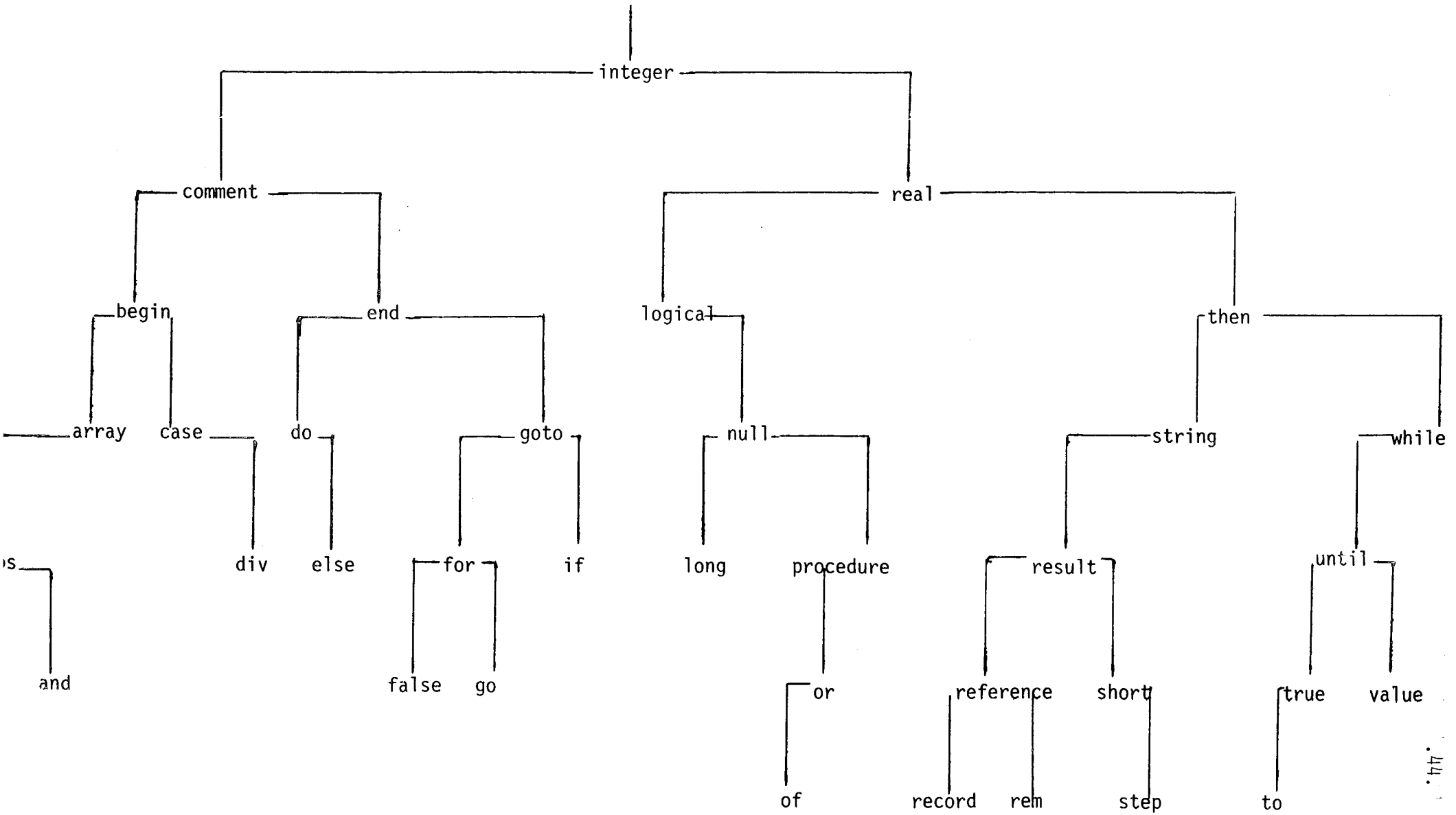
KNUTH montre que si on ajoute un nouveau noeud $A_{n+1} > A_n$, la racine ne se déplace jamais à gauche. En conséquence, KNUTH démontre que le coût de l'algorithme peut être de $O(n^2)$: $(\frac{3n^2-n}{2})$.

Le volume nécessaire est de $6n^2$ octets mémoire en ne stockant que les matrices triangulaires.

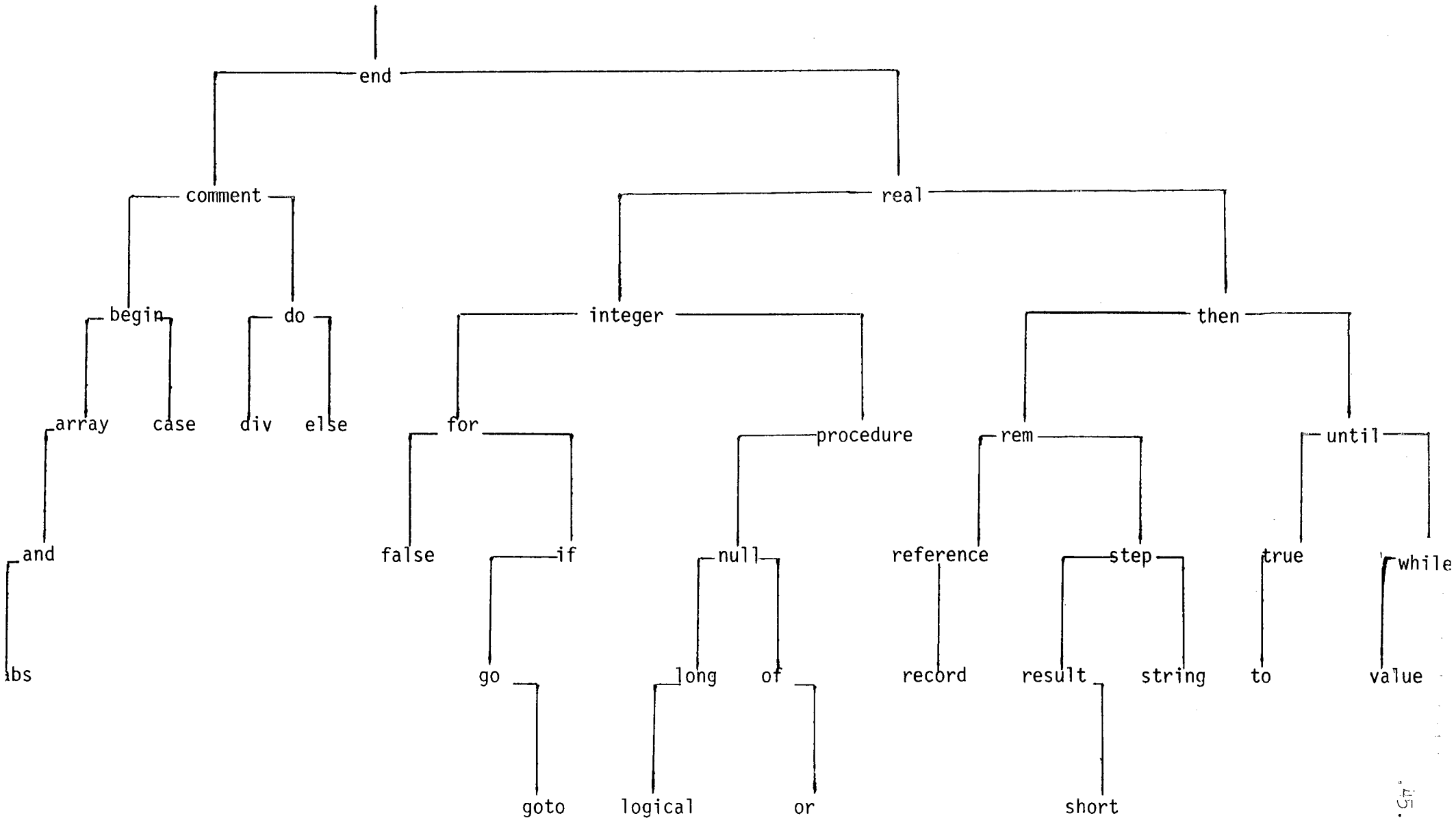
Nous donnons l'exemple fourni par KNUTH.

Soit un corpus formé de 25 programmes ALGOL W et l'ensemble A_i des noeuds de l'arbre constitué des 36 symboles de base du langage ALGOL

α	symboles de base	β		α	symboles de base	β
33	abs	1		113	null	8
5	and	6		2	of	5
0	array	9		0	or	5
26	begin	77		30	procedure	16
37	case	5		38	real	29
12	comment	95		0	record	2
54	div	12		0	reference	13
0	do	50		0	rem	9
23	else	16		0	result	0
0	end	77		23	short	0
15	false	2		11	step	5
0	for	35		0	string	5
36	go	1		99	then	34
0	goto	1		2	to	1
57	if	34		4	true	8
7	integer	37		5	until	34
142	logical	2		4	value	8
0	long	5		0	while	16
113				111		



Arbre binaire optimal obtenu en tenant compte des fréquences des symboles de base et des fréquences des identificateurs



Arbre binaire optimal obtenu en tenant compte de la fréquence des symboles de base

III.2. MESURES ET ESSAIS

Toutes les mesures de temps que nous avons effectuées concernent un corpus de 6520 mots constitués de 10200 éléments. L'identification est réalisée dans un dictionnaire de 2000 éléments dont 1600 éléments "frères".

Etant donné, la stratégie employée dans les modèles et la grammaire morphologiques, les éléments les plus fréquents sont bien entendu, les mots invariables de caractère général, mais également les suffixes (AL,ER,IONN,ISME,...) et les désinences (␣,E␣,S␣,T␣,...).

D'autre part, nous avons vu que seuls les éléments "frères" peuvent être des noeuds de l'arbre, les fréquences des éléments "fils" sont donc cumulées à celle de l'élément "frère" correspondant.

Une identification de notre échantillon avec comptages a été nécessaire pour obtenir ces fréquences.

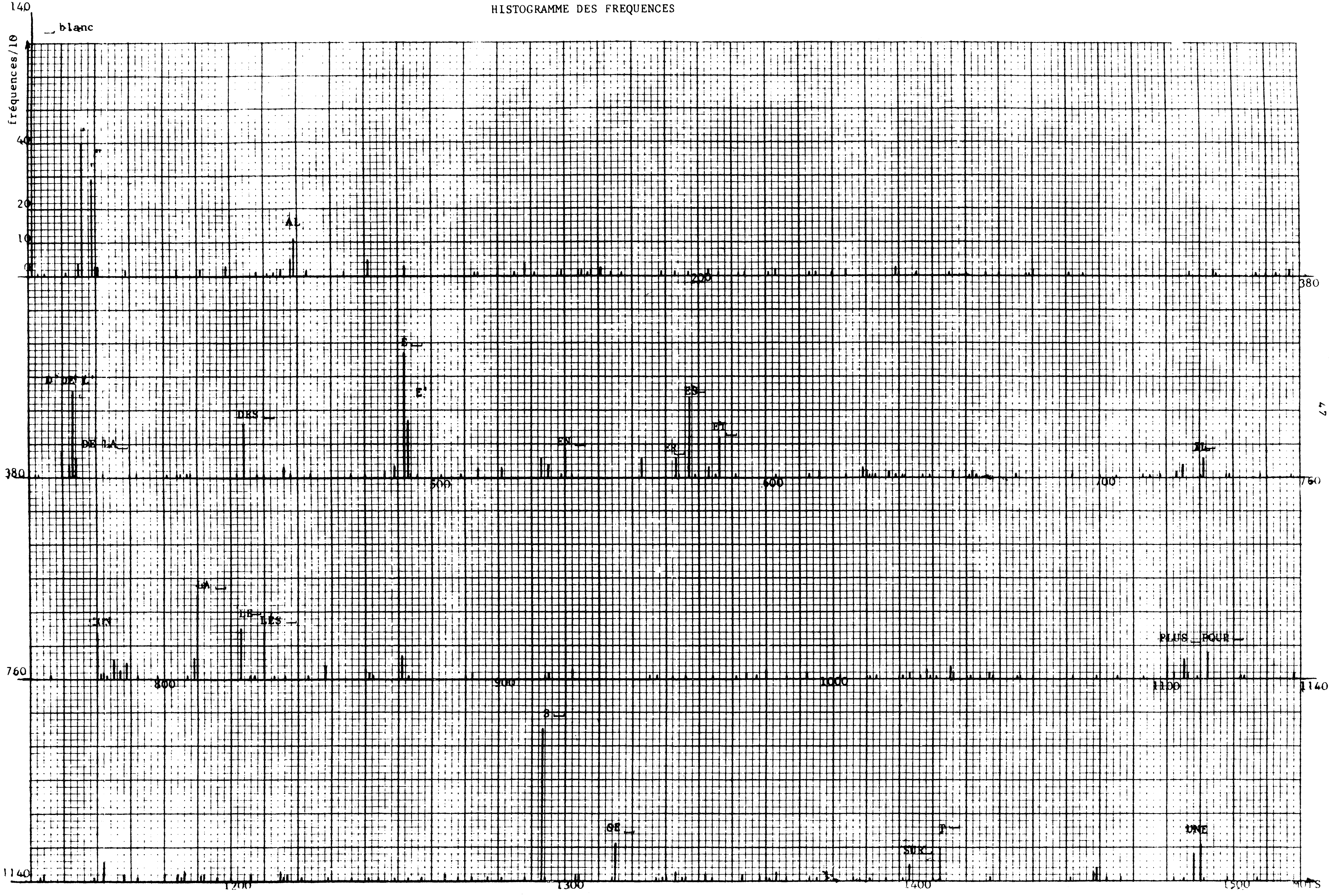
L'histogramme, ci-après, nous indique la répartition des fréquences sur l'ensemble des 1600 éléments "frère" du dictionnaire, classés alphabétiquement.

Pour effectuer des mesures différentes et pouvoir les comparer, le seul objet qui puisse évoluer est l'arbre d'entrée dans le dictionnaire.

Cet arbre possède en fait 3 paramètres :

1) le nombre de noeuds,
2) la position de ces noeuds par rapport à l'ensemble des éléments "frère",
3) la forme de l'arbre binaire (équilibré ou non).

HISTOGRAMME DES FREQUENCES



III.3. COUT THEORIQUE

Ceci nous servira de méthode de référence pour les évaluations.

On sait que le nombre de comparaisons pour effectuer le parcours d'une branche d'un arbre binaire est de l'ordre de Log_2 p si p est le nombre de noeuds.

La longueur moyenne de recherche dans une file séquentielle ordonnée, de longueur q, est : $\frac{q}{2}$. Dans notre cas, les classes ont une longueur moyenne de $q = \frac{n}{p}$ si n est le nombre d'éléments "frère" du dictionnaire (hypotnèse équiprobable sur chaque classe)
Le programme d'identification contient des instructions d'appel des chaînes de données et de renvoi des résultats (constante γ).

Le coût total d'identification des 10200 éléments du corpus (N) peut s'exprimer par la formule :

$$\text{Coût total} = N(\alpha \, Log_2 \, p + \beta \frac{n}{2p} + \gamma)$$

Nous avons d'abord mesuré avec toute la précision possible les coefficients α, β et γ. Pour cela nous avons établi un corpus contenant certains noeuds de l'arbre et certains éléments "frère" répétés 1000 fois Par différence des temps, nous avons évalué un changement de niveau dans l'arbre à 25 µs environ.

Soit N $\alpha = 25 \times 10^{-6} \times 10200 \simeq 0{,}25$ seconde car on passe toujours par l'arbre.

Les mesures de temps nécessaires à un changement d'élément "frère" donnent 17 µs.

Soit N $\beta = 17 \times 10^{-6} \times 10200 \simeq 0{,}17$ seconde.

Pour déterminer la constante γ, il suffit d'exécuter le programme sur le texte complet en enlevant les instructions d'identification propres (laisser les appels de procédure).

On obtient un temps total de 2,6 secondes.

$$N\gamma = 2,6$$

Une mesure globale d'identification du corpus avec un seul noeud (00) dans l'arbre, c'est-à-dire pour un balayage séquentiel des éléments "frère" nous donne un temps de 140 secondes.

Ce qui confirme le coefficient β :

$$C_T = N(\alpha \, \mathrm{Log}_2 \, p + \beta \frac{n}{2p} + \gamma)$$

$$140 = 0,25 \, . \, 0 + N\beta \frac{1600}{2} + 2,6$$

$$\underline{N\beta = 0,172 \text{ seconde}}$$

La tabulation du coût total pour quelques points particuliers, nous permet de tracer la courbe moyenne théorique en supposant une répartition homogène des fréquences,(Courbe 0).

III.4. COURBES DE COUT TOTAL

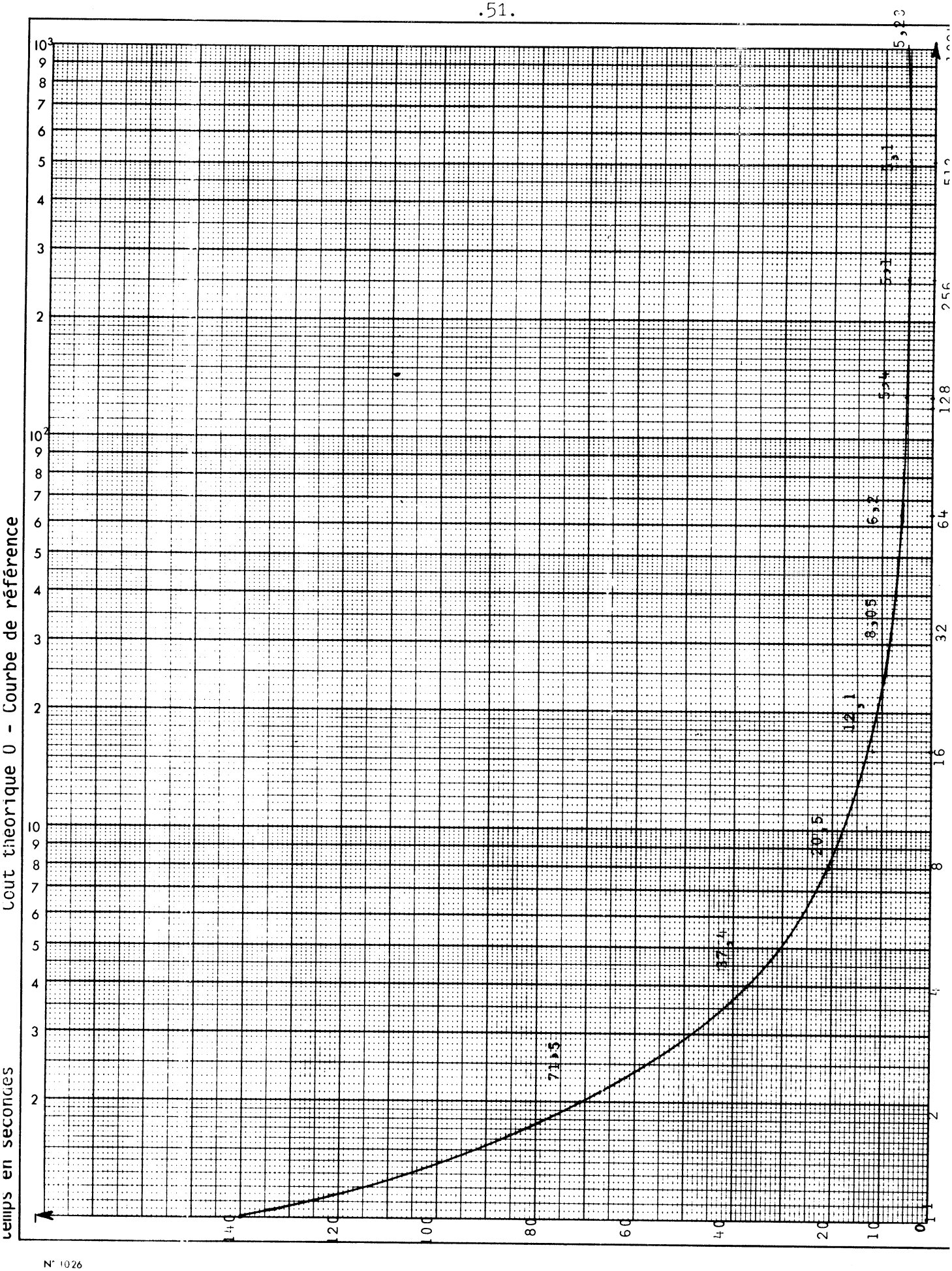
Tabulation pour la courbe 0

P Noeuds	Log_2 P	$N\alpha$ Log_2 P	$\frac{n}{2p}$	$N\beta$ $\frac{n}{2p}$	coût total en secondes
1	0	0	800	137,4	140
2	1	0,25	400	68,7	71,5
4	2	0,5	200	34,3	37,4
8	3	0,75	100	17,1	20,5
16	4	1	50	8,5	12,1
32	5	1,25	25	4,2	8,05
64	6	1,5	12	2,1	6,2
128	7	1,75	6	1,05	5,4
256	8	2	3	0,52	5,1
512	9	2,25	2	0,26	5,1
1024	10	2,5	1	0,13	5,23

On peut déjà constater que la courbe passe par un minimum entre 256 et 512 noeuds dans l'arbre.

Le gain est très faible à partir de 128 noeuds.

Il se trouve, que pour des raisons de taille mémoire avec une machine virtuelle standard de 256 K octets, l'algorithme de KNUTH nous autorise la construction d'arbres de 120 noeuds au plus.


Temps en secondes
Cout theorique O - Courbe de référence
N° 1026
71,5
87,4
20,5
12,1
8,05
6,2
5,1
5,1
5,1
5,23

Nous avons effectué plusieurs séries de mesures en faisant varier le nombre de noeuds de 2 à 120 et en tenant compte des fréquences dans :

1) Le choix des noeuds - Influence sur la taille des clas
2) La forme de l'arbre binaire : équilibré ou non.

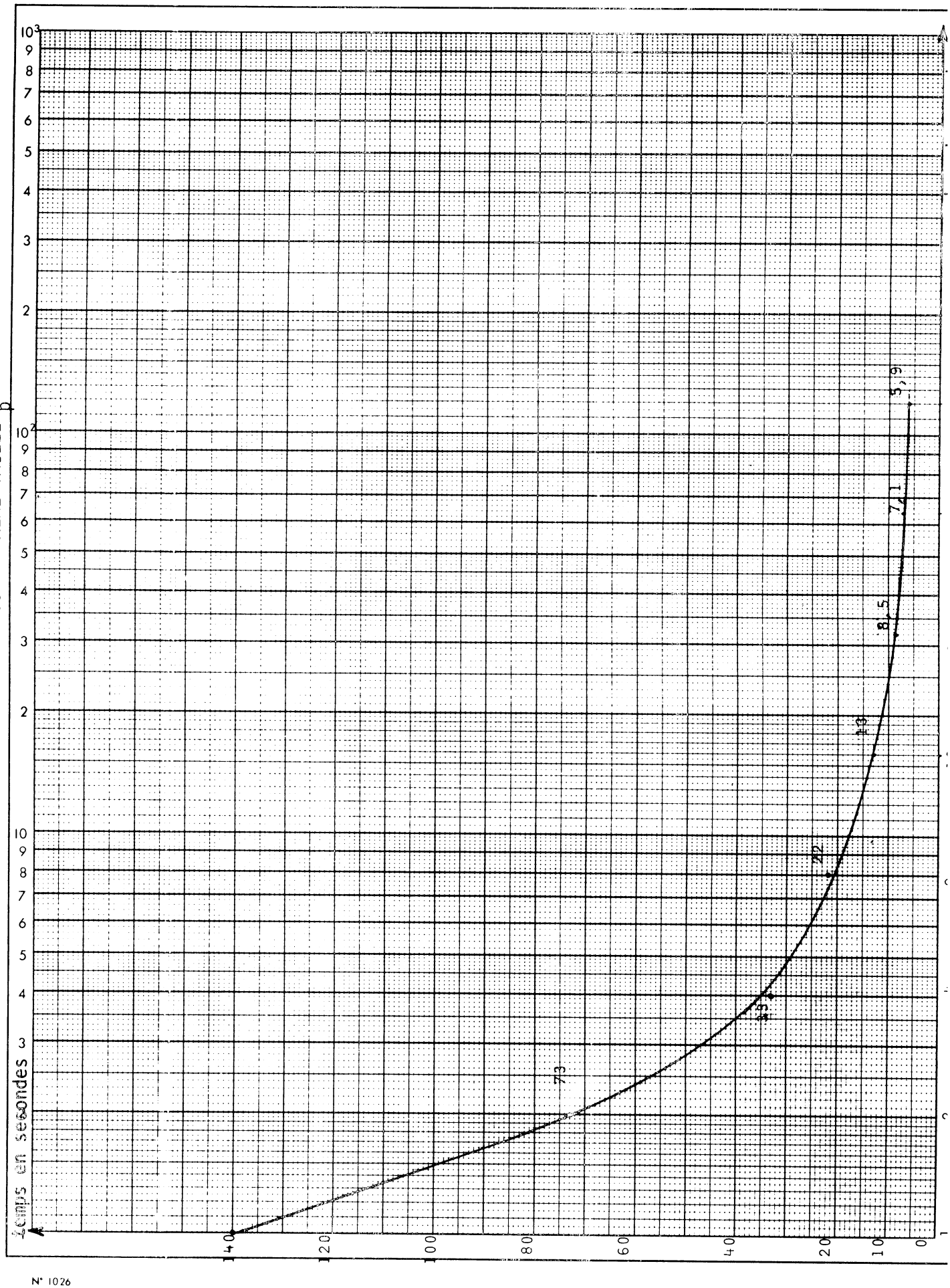
Courbe 1: Nous avons découpé l'ensemble des éléments "frère" en classes de même taille.

Nous avons supposé une répartition homogène des fréquences d'appel des éléments. Nous avons donc construit un arbre binaire équilibré. Ce résultat est obtenu avec l'algorithme de KNUTH si les fréquences α et β sont égales ($\alpha = \beta = 1$) ou plus simplement en faisa une dichotomie simple.

On constate que la courbe obtenue expérimentalement avec les mêmes hypothèses que pour la courbe théorique, suit de très près celle-ci.

Les temps obtenus sont du même ordre de grandeur que ceux de la courbe théorique. Ils sont légèrement supérieurs car les classe sont de même taille mais les fréquences (β) sont très inégales.

Ceci montre la validité de l'étude théorique et la relativ régularité de la répartition des fréquences.


COURBE 1
DICHOTOMIE SIMPLE - CLASSE DE MEME TAILLE n/p (frequences inégales)(KNUTH α=β= 1)
temps en secondes
N° 1026
140
120
100
80
60
40
20
10
0
73
45
27
13
8,5
7,1
5,9

Courbes 2 et 3 : Cas normal (1ère stratégie choisie)

On sélectionne manuellement les éléments "frère" les plus fréquents et on construit l'arbre binaire avec l'algorithme de KNUTH en tenant compte de :

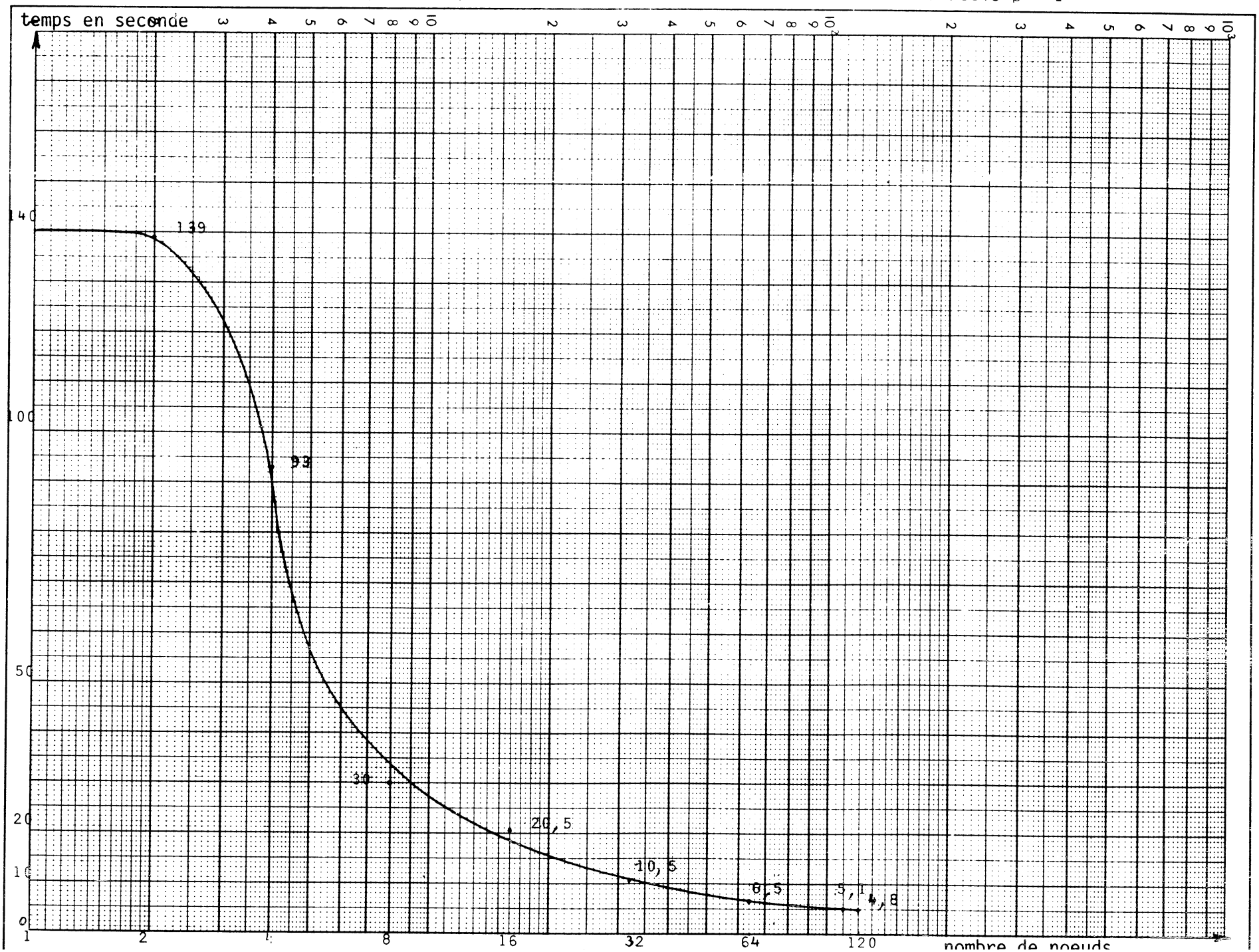
- la fréquence α des noeuds pour tracer la courbe 2
- la fréquence α et de la fréquence β de chaque classe (ΣF_i) pour tracer la courbe 3.

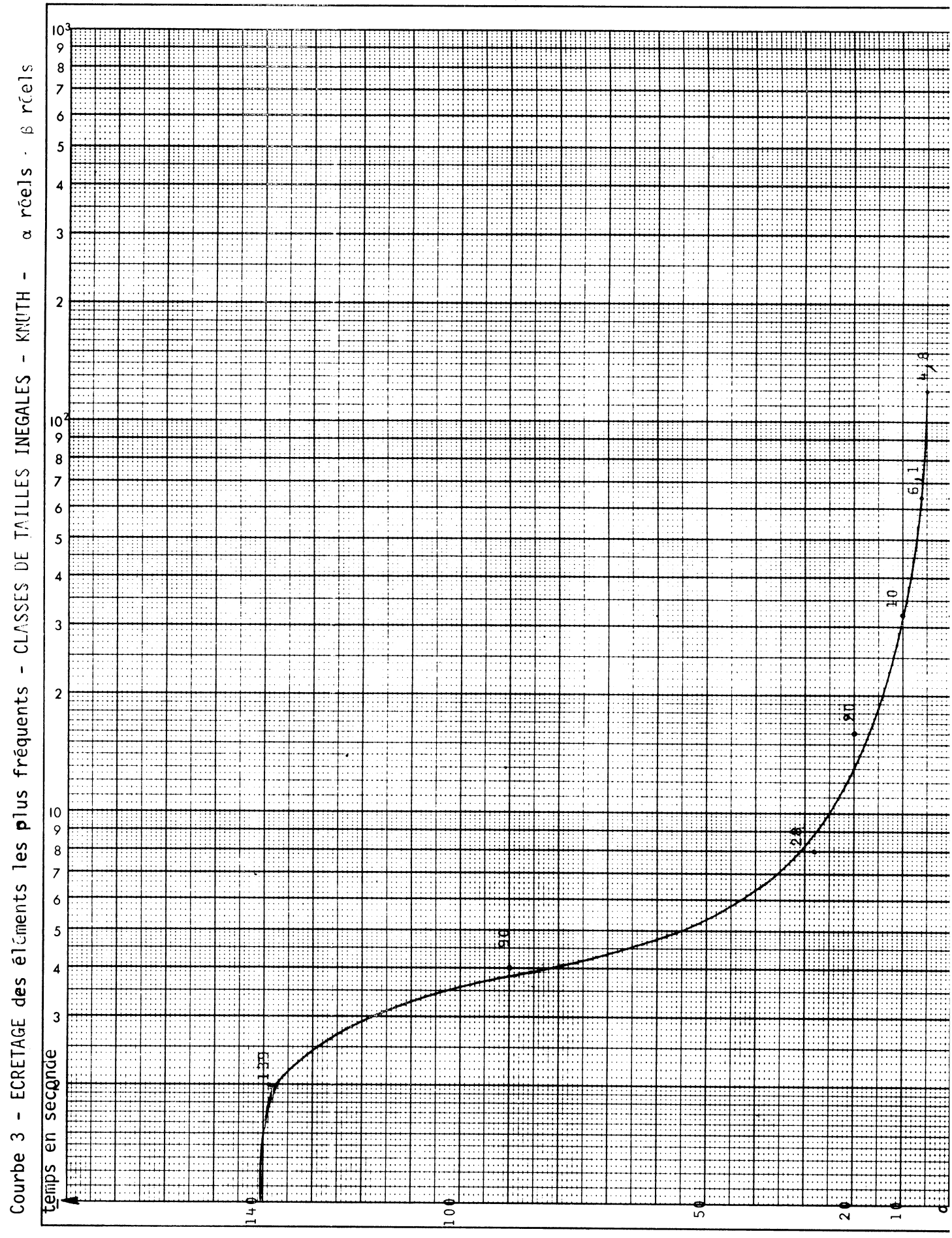
On constate, qu'étant donné la répartition très hétérogène des fréquences et les écarts de valeurs importants, les temps obtenus sont beaucoup plus élevés que ceux de la courbe théorique quand il y a peu de noeuds dans l'arbre. Les classes sont très inégales en taille et en fréquence.

Les temps deviennent rapidement égaux, puis inférieurs à ceux de la courbe théorique quand p augmente.

Pour un arbre de 64 noeuds, le temps est égal à celui de la courbe 0 et pour 120 noeuds, il est inférieur d'environ 10 %

Courbe 2 - ECRETAGE des éléments les plus fréquents - CLASSES DE TAILLES INEGALES - KNUTH α réels $\beta = 1$




Courbe 3 - ECRETAGE des éléments les plus fréquents - CLASSES DE TAILLES INEGALES - KNUTH - α réels · β réels
temps en seconde
10^3
10^2
140
100
50
20
10
1,3m
5a
2a
80
10
6,1
4,6
N° 1026

Courbes 4 et 5 :

Dans ces 2 séries de mesures, on a cherché à rétablir l'égalité des classes.

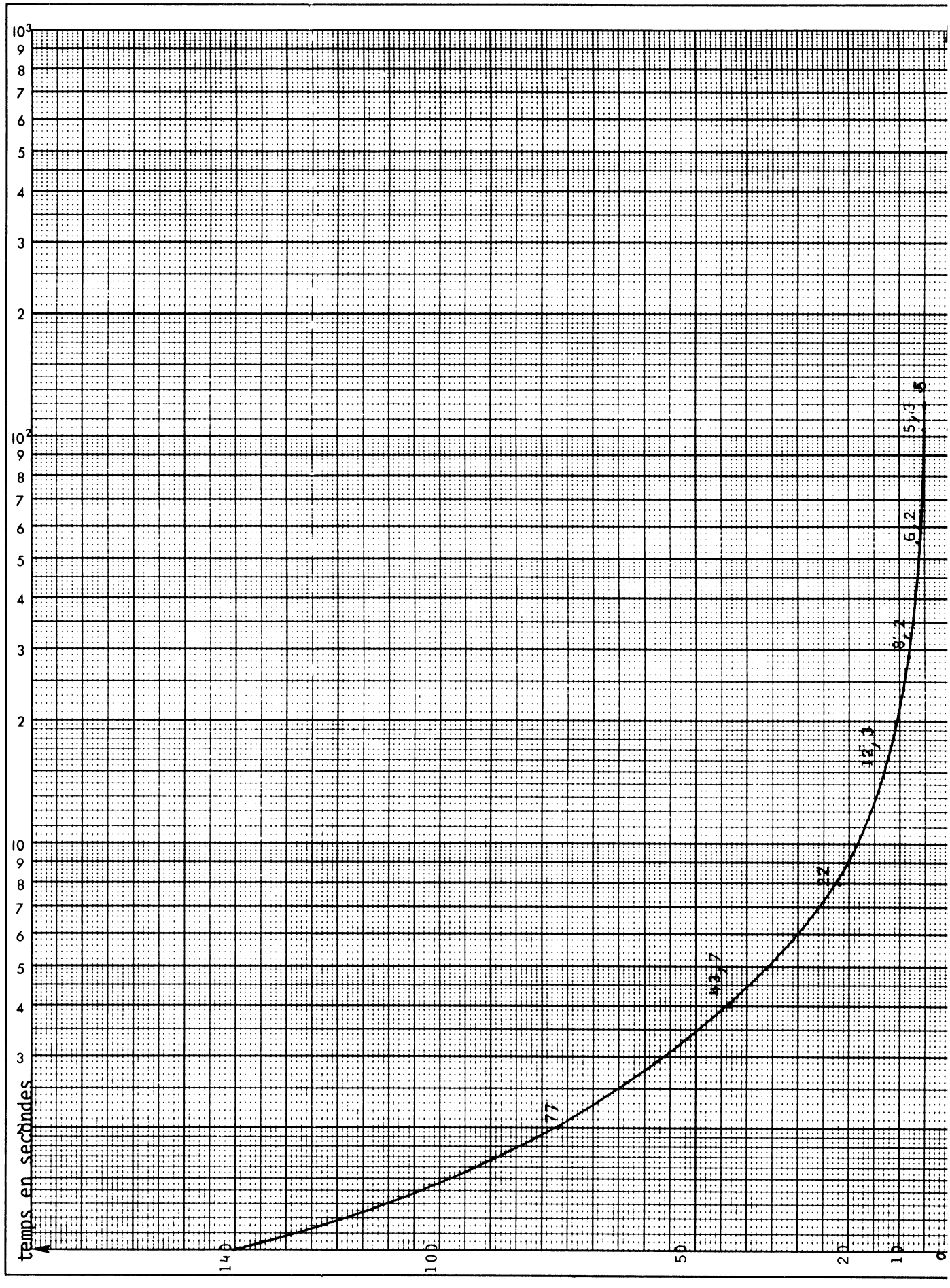
- On sélectionne les noeuds de l'arbre de telle sorte que les classes réalisées possèdent sensiblement la même fréquence :

$$\Sigma F_i = \Sigma F_j$$

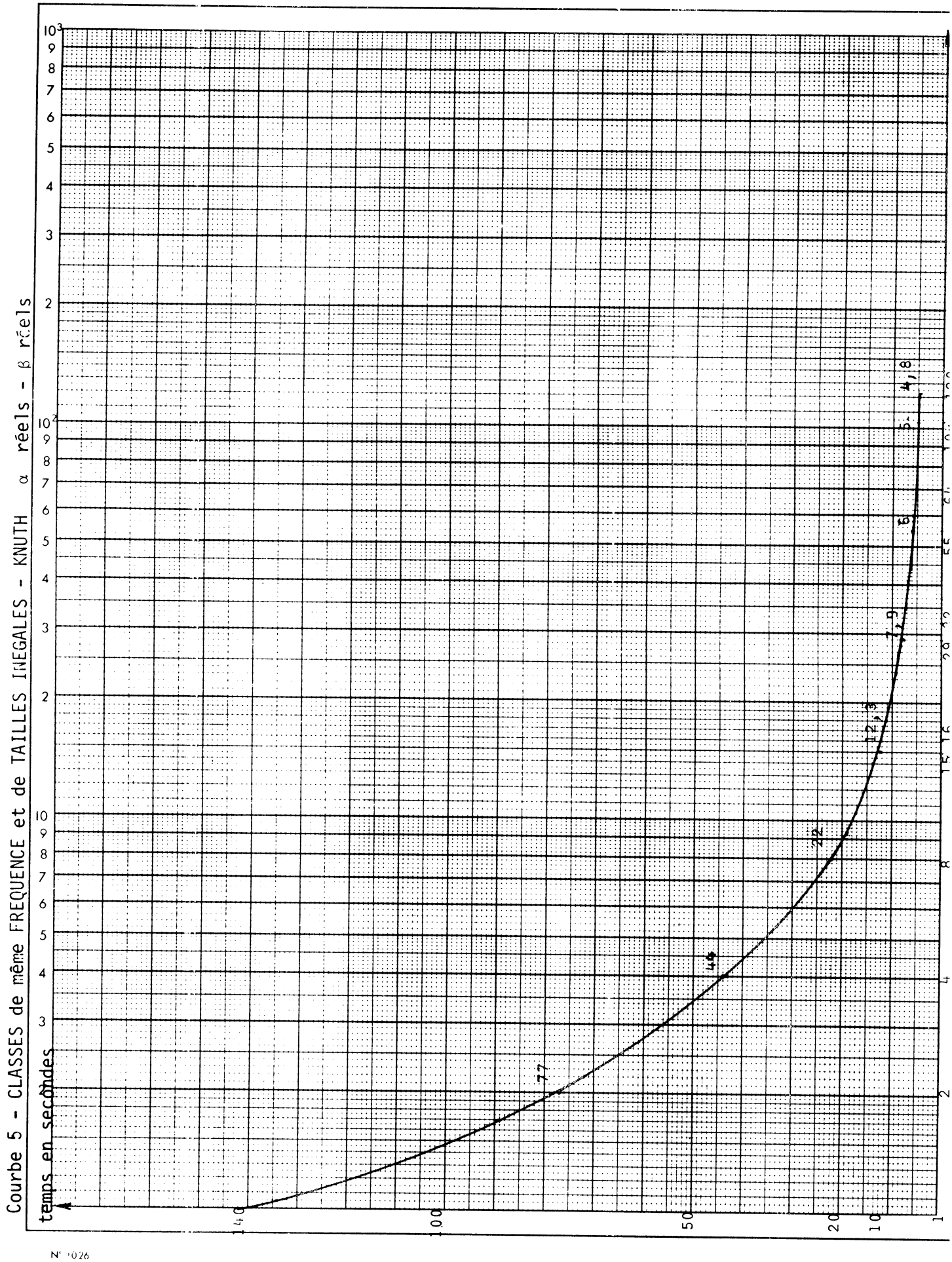
- On construit l'arbre binaire à l'aide de l'algorithme de KNUTH en tenant compte de :
  - la fréquence α des noeuds pour obtenir la courbe 4
  - la fréquence α, de la fréquence β de chaque classe pour obtenir la courbe 5

On constate effectivement une diminution des temps entre 2 et 32 noeuds par rapport aux courbes 2 et 3 précédentes.

Les courbes sont très proches de la courbe théorique et de plus on retrouve un gain de 10 % pour l'arbre maximal.


Courbe 4 - CLASSES de même FREQUENCE (tailles inégales) - KNUTH α = réels - β = 1
temps en secondes
10^3
9
8
7
6
5
4
3
2
10^2
9
8
7
6
5
4
3
2
10
9
8
7
6
5
4
3
140
100
50
25
1
d
77
43,7
22
12,3
8,2
6,2
5,3
5

N° 1026


Courbe 5 - CLASSES de même FREQUENCE et de TAILLES INEGALES - KNUTH α réels - β réels
temps en secondes
77
44
22
12,3
7,9
5,6
4,8

N° 1026

Courbes 6 et 7 :

Nous avons cherché à nous rapprocher de la réalité en tenant compte, non seulement des fréquences, mais aussi de leur répartition dans le dictionnaire.

- On sélectionne les noeuds de l'arbre de telle sorte que les classes obtenues possèdent sensiblement le même coût :

$$\Sigma\ i.F(i) = \Sigma\ j.(F(j)$$

- On construit l'arbre binaire avec l'algorithme de KNUTH en tenant compte de la fréquence (α) des noeuds et du coût des classes (β) pour tracer la courbe 6.

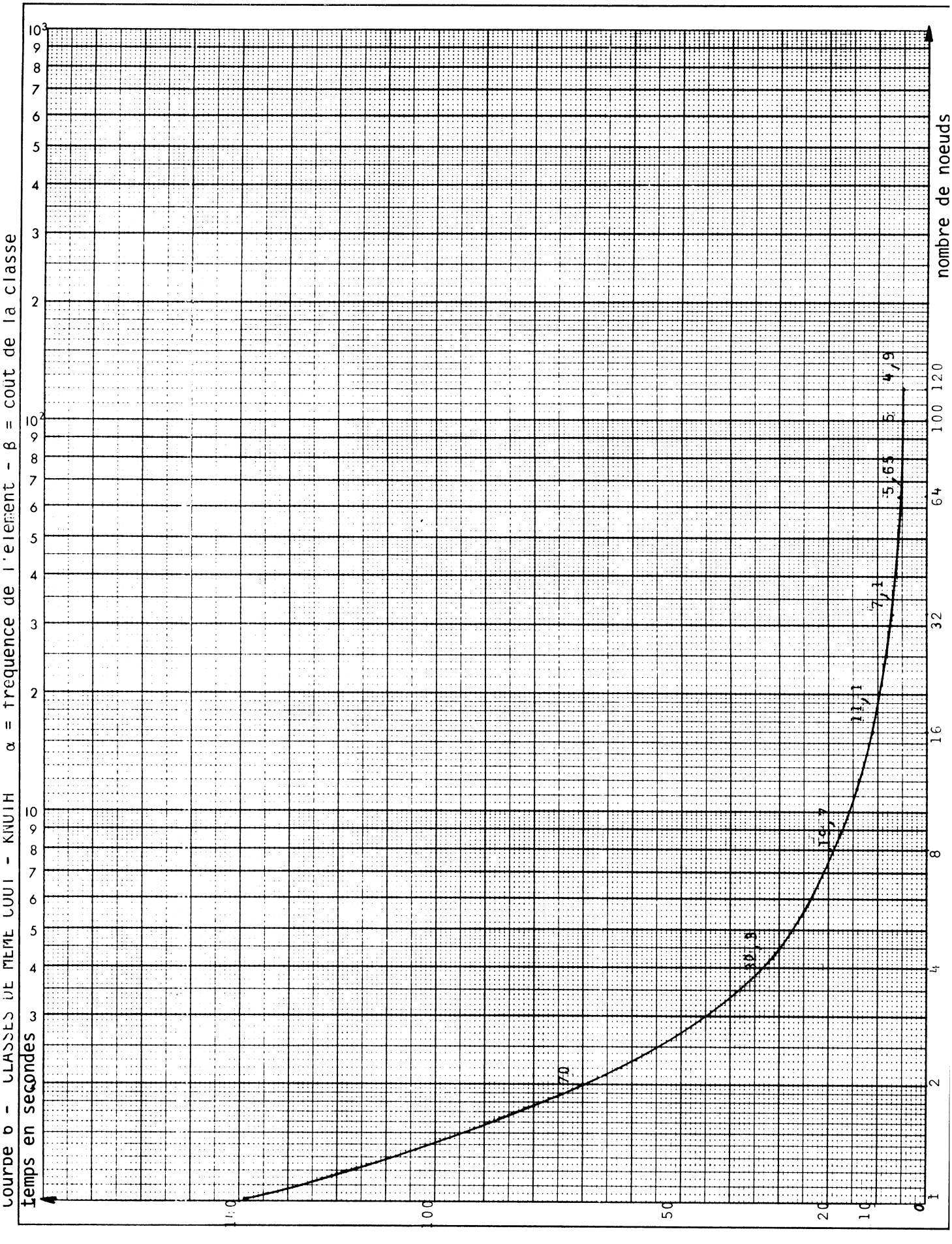
Comme, par définition, les coûts sont égaux et importants devant la fréquence des noeuds, on a construit un arbre binaire équilibré pour obtenir la courbe 7.

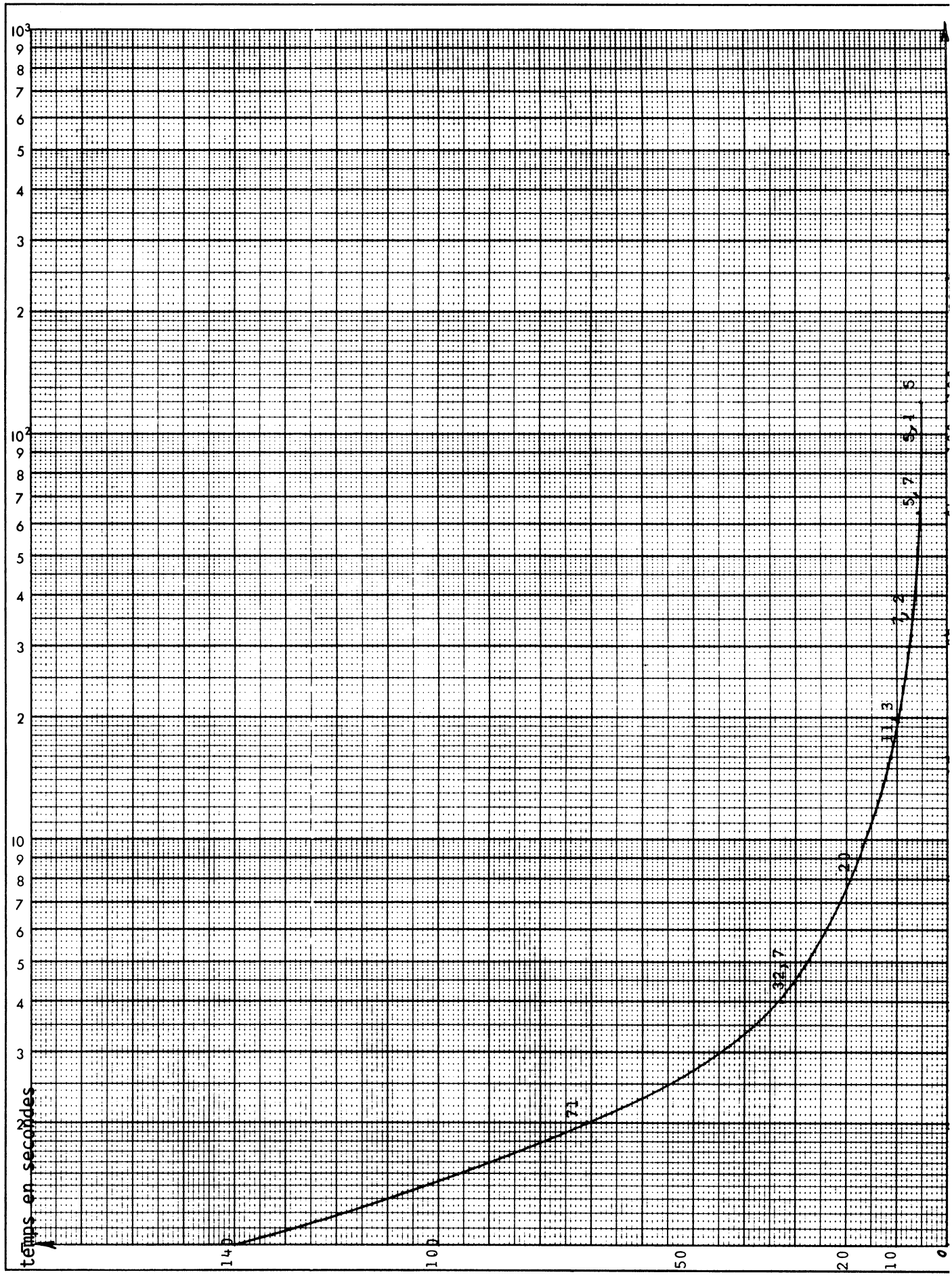
. Les temps obtenus avec l'algorithme de KNUTH - courbe 6- sont tous inférieurs de près de 15 % à ceux de la courbe théorique.

. Les temps obtenus avec un arbre binaire équilibré - courbe 7 - sont sensiblement identiques à ceux de la courbe 6.

On voit donc que le gain par rapport à la courbe 1 - Dichotomie simple - est de l'ordre de 20 à 25 %.

Ces derniers résultats confirment bien les évaluations précédentes.


courbe b - CLASSES DE MEME COUT - KNUTH α = frequence de l'element - β = cout de la classe
temps en secondes
nombre de noeuds
170
80,8
19,7
11,1
7,1
5,65
4,9
1
2
4
8
16
32
64
100
120
10
20
50
100
1000
N° 1026



Courbe 7 - CLASSES DE MEME COUT - ARBRE BINAIRE EQUILIBRE

N° 1026

Le calcul manuel des coûts gauche et droit devenant assez long et fastidieux, il a été nécessaire d'étudier et d'écrire un programme qui sélectionne automatiquement les p noeuds à placer dans l'arbre.

Cet algorithme consiste à itérer sur la décomposition d'une liste en 2 sous-listes de sorte que :

coût gauche ≈ coût droit

jusqu'à l'obtention de p sous-listes.

La recherche du point milieu qui satisfait à la condition ci-dessus est réalisée en effectuant une recherche dichotomique qui limite le nombre d'essais, sur la liste.

$$\Delta = \left(\sum_{i=1}^{k-1} i.F(i) - \sum_{i=k+1}^{n-1} (i-k) \, . \, F(i) \right)$$

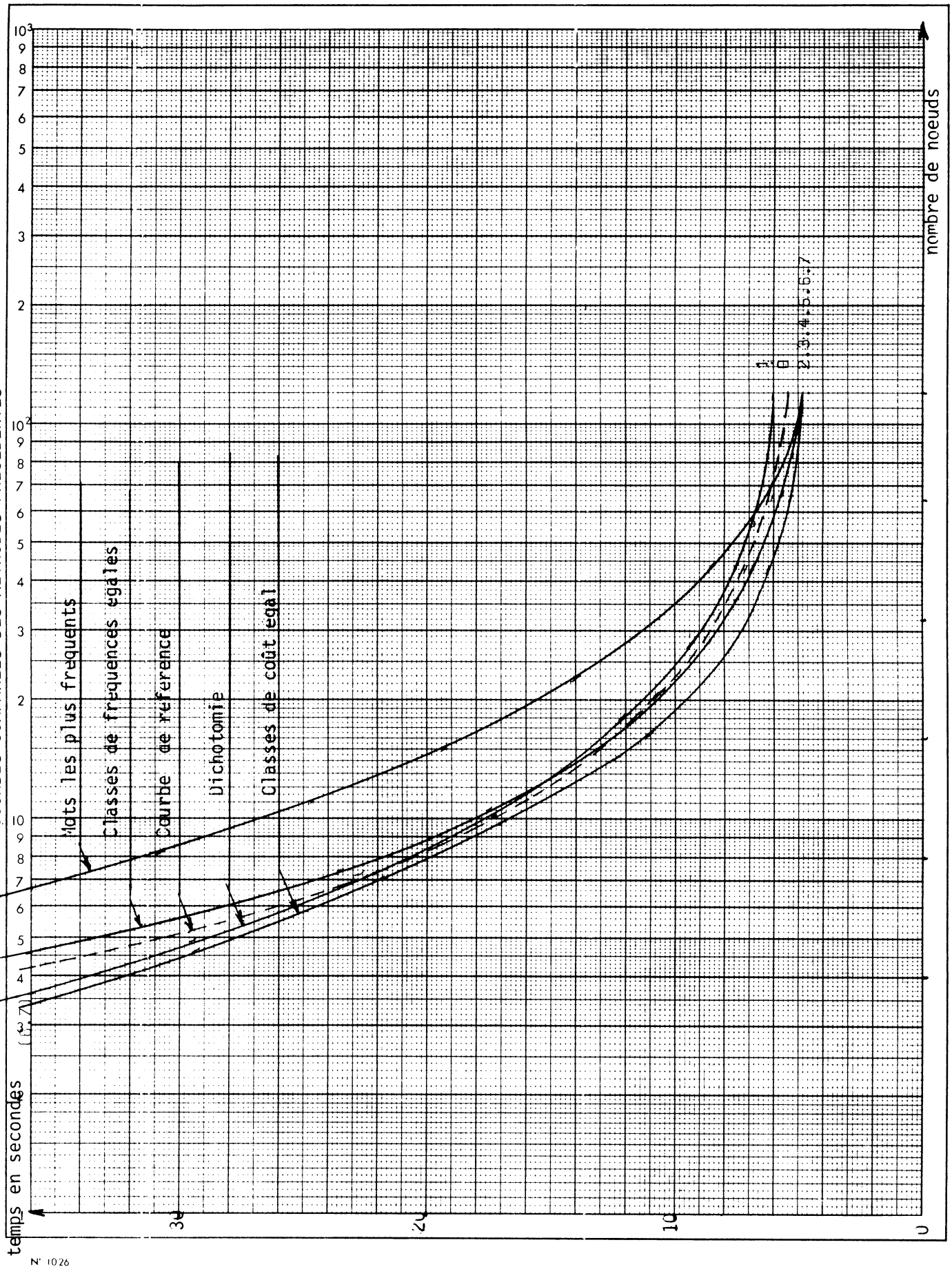
on prend l'élément k pour lequel on a Δ minimum.

- La règle est donc de tenir compte du coût des classes en utilisant la fréquence d'appel des éléments et la répartition de ces fréquences.

- En conséquence, il suffit de construire un arbre binaire équilibré qui évite l'utilisation de l'algorithme de KNUTH qui demandait $\underline{6\ p^2\ \text{octets}}$ de mémoire.

Il suffit de faire une recherche dichotomique sur les noeuds sélectés pour construire l'arbre binaire équilibré.

Toutes ces observations, nous conduisent à proposer un algorithme d'élaboration automatique de l'arbre, qui va encore améliorer les gains, en recherchant les classes de coût minimal.


COURBES COMPAREES DES METHODES PRECEDENTES
temps en secondes
nombre de noeuds
Mots les plus frequents
Classes de fréquences égales
Courbe de référence
Dichotomie
Classes de coût égal
1 (4.5) (2.3)
1
2.3.4.5.6.7
N° 1026

III.5. PRINCIPE DE L'ALGORITHME D'ELABORATION AUTOMATIQUE DE L'ARBRE

Cet algorithme a pour but de décomposer une liste d'objets ordonnés, auxquels on accède par une recherche séquentielle, en p sous-listes de sorte que la durée moyenne de recherche dans les sous-listes soit optimale pour une valeur p donnée de manière à minimiser le coût global.

On tient compte de la distribution des fréquences d'appel des objets.

Un arbre d'accès contenant p noeuds est généré.

Le coût total d'une recherche séquentielle dans une liste de n objets de fréquence F(i) est :

$$\text{Coût total} = \sum_{i=1}^{n-1} i.F(i)$$

Remarque : Si on suppose une fréquence d'appel constante F, le coût total est :

$$C_T = \frac{(n-1)n}{2} F \quad \text{soit environ} \quad \frac{n^2}{2} F$$

Le problème consiste à décomposer une liste de n objets en p segments de sorte que la somme descoûts des p segments soit minimale.

Soit chercher $k_1 \ldots k_{p+1} \in$ (k:k entier, $1 < k < n$) tels que:

$$\sum_{j=1}^{p} \sum_{i=k_j+1}^{k_{j+1}-1} (i-k_j).F(i) \text{ soit minimale}$$

Nous ne connaissons pas d'algorithme efficace qui nous permette d'obtenir cette décomposition optimale globalement.

La méthode serait combinatoire, elle entraînerait la construction des C_n^p intervalles et leur évaluation.

Comme nous ne cherchons qu'à minimiser l'identification, nous avons utilisé une méthode itérative simple.

Nous avons constaté un faible écart entre la méthode dichotomique (courbe 1) et la méthode de référence (courbe 0) ce qui laisse supposer une certaine régularité dans la distribution des fréquences.

Nous proposons donc de découper la liste de n objets en 2 sous-listes en supposant que :

$$\min(\text{Coût total}) \simeq \min(\text{coût gauche} + \text{coût droit})$$

Les coûts gauche et droit n'ont aucune raison de possèder la même valeur car nous calculons le point limite des 2 segments pour obtenir un gain maximal. Les coûts gauche et droit sont ensuite évalués.

- Nous recherchons alors la sous-liste ayant un coût maximal et nous découpons, à nouveau, celle-ci en 2 sous-listes.

Ce procédé est répété jusqu'à ce qu'on obtienne p sous-listes.

- Nous obtenons alors des classes ayant des coûts de même ordre de grandeur.

- Le fait de sélectionner la sous-liste de coût maximal ne fournit pas toujours le meilleur résultat quand les coûts sont proches et quand p est petit, mais ce phénomène disparaît rapidement quand on augmente p.

Calcul du gain en coupant en 2 :

$$\min(\text{coût total}) = \min\left(\sum_{i=1}^{k} i.F(i) + \sum_{i=k+1}^{n-1} (i-k)\, F(i) \right)$$

Mais en réalité, il faut tenir compte du fait que l'élément k qui peut être sélecté, n'entre plus dans le calcul du coût minimum.

$$\text{mini}\left(\sum_{i=1}^{k-1} i.F(i) + \sum_{i=k+1}^{n-1} (i-k)\, F(i) \right)$$

$$\text{mini}\left(\sum_{i=1}^{k-1} i.F(i) + \sum_{i=k+1}^{n-1} i.F(i) - k \sum_{i=k+1}^{n-1} F(i) \right)$$

$$\text{mini}\left(\sum_{i=1}^{n-1} i.F(i) - k\, F(k) - k \sum_{i=k+1}^{n-1} F(i) \right)$$

$$\text{mini}\left(\sum_{i=1}^{n-1} i.F(i) - k \sum_{i=k}^{n-1} F(i) \right)$$

Cette expression est minimum quand $k \sum_{i=k}^{n-1} F(i)$ est maximum (gain)

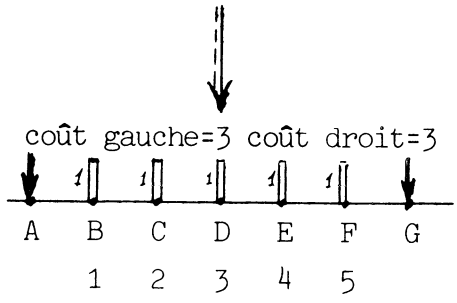
car le 1er terme $\sum_{i=1}^{n-1} i.F(i)$ est constant.

L'algorithme consiste alors à évaluer toutes les valeurs de $k \sum_{i=k}^{n-1} F(i)$ pour i = n-1 jusqu'à 1. L'indice k est obtenu quand le produit $k \sum_{i=k}^{n-1} F(i)$ est maximum.

Il suffit alors de calculer les coûts gauche et droit.

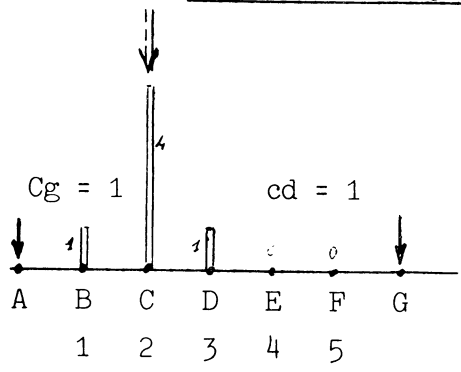
Nous donnons 3 exemples simples d'évaluation de l'indice k pour des cas typiques.

1er cas : Classe contenant des éléments de même fréquence



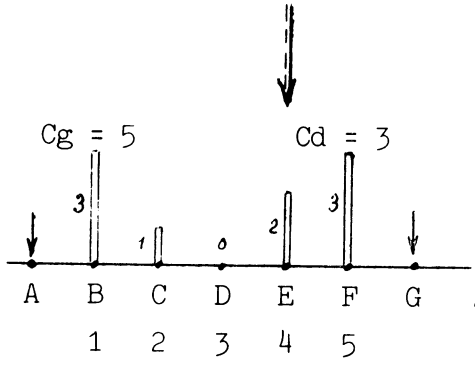
Nom	k	$\Sigma_5^1 F(i)$	$k \Sigma F(i)$
F	5	1	5
E	4	2	8
D	3	3	9
C	2	4	8
B	1	5	5

2ème cas : Classe contenant des éléments dont la fréquence est élevée dans la partie gauche



Nom	k	$\Sigma_5^1 F(i)$	$k\Sigma F(i)$
F	5	0	0
E	4	0	0
D	3	1	3
C	2	5	10
B	1	6	6

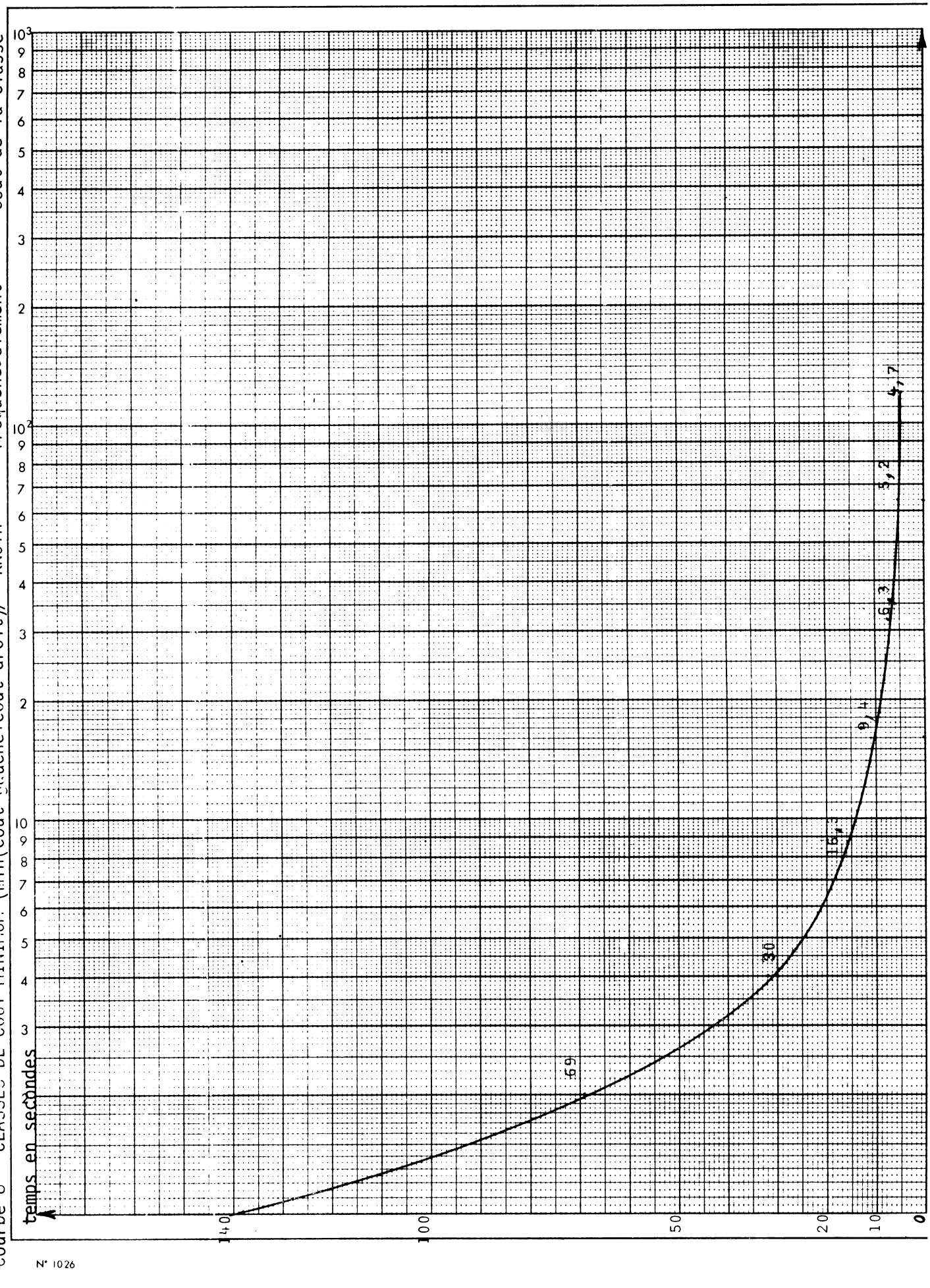
3ème cas : Classe contenant des éléments dont la fréquence est élevée près des extrémités

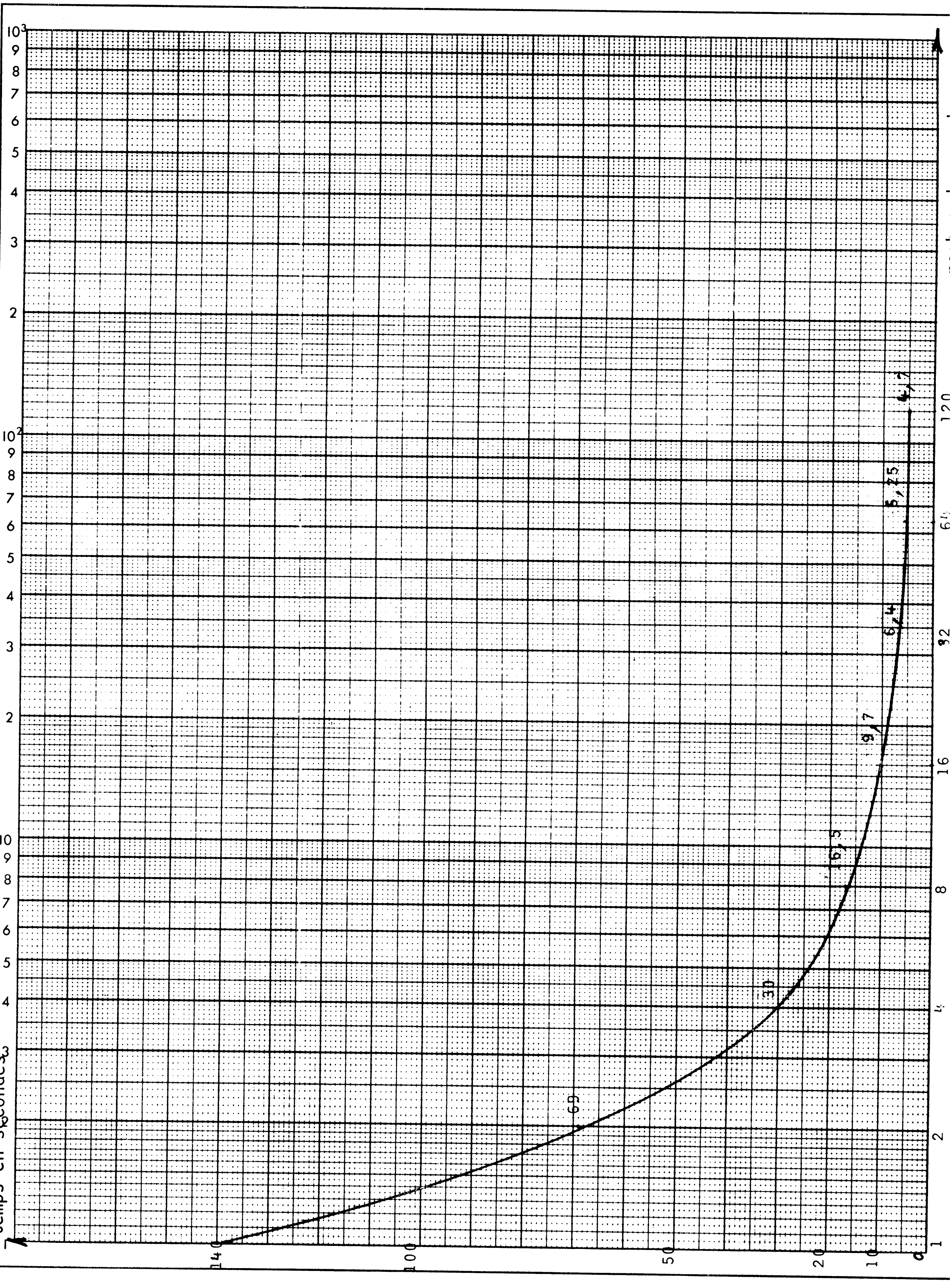


Nom	k	$\Sigma_5^1 F(i)$	$k\Sigma F(i)$
F	5	3	15
E	4	5	20
D	3	5	15
C	2	6	12
B	1	9	9

Nous avons tracé les courbes 8 et 9, analogues aux courbes 6 et 7 du chapitre précédent, avec ce nouvel algorithme et nous pouvons constater une diminution des temps d'identification, en particulier, quand les arbres d'accès sont petits. Cette constation est très importante car cette situation reflète le cas d'implantation du dictionnaire sur une petite machine ayant des pages de faible volume donc un arbre d'accès contenant peu de noeuds.

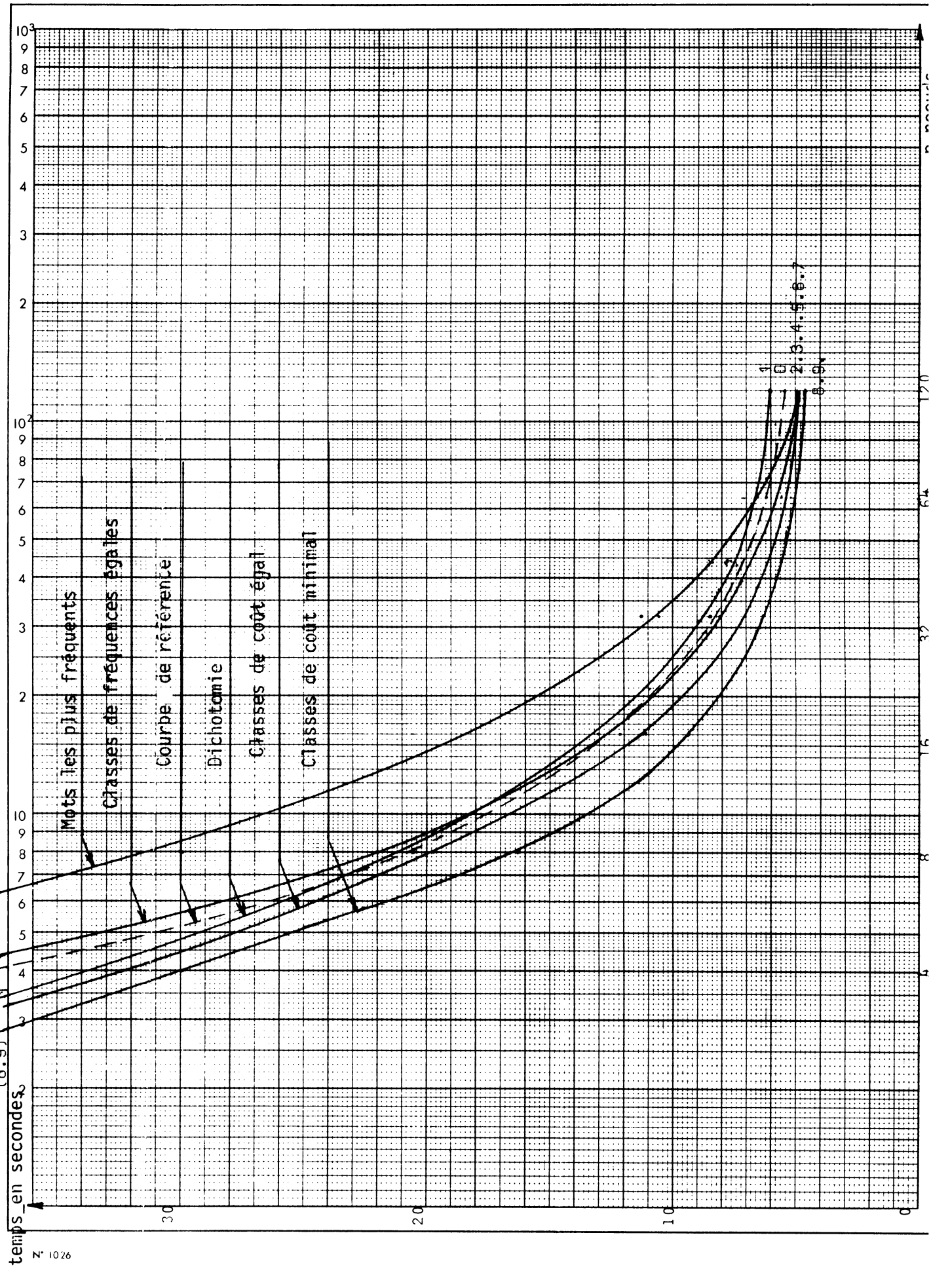
Nous avons résumé sur un même graphique les courbes des diverses méthodes précédentes : courbes comparées page 73.


Courbe 8 - CLASSES DE COUT MINIMUM (min(coût gauche+coût droit)) - KNUTH - fréquence élément - coût de la classe
temps en secondes
N° 1026
140
100
50
20
10
0
69
30
16,1
9,4
6,3
5,2
4,7



Courbe 9 - ARBRE AUTOMATIQUE - CLASSES de coût minimum min(coût gauche+coût droit) - ARBRE BINAIRE EQUILIBRE

N° 1026


COURBES COMPAREES
(6.7) (4.5) (2.3)
(8.9)
temps en secondes
Mots les plus fréquents
Classes de fréquences égales
Courbe de référence
Dichotomie
Classes de coût égal
Classes de coût minimal
1 2.3.4.5.6.7
8.9.
N° 1026

III.6. ALGORITHME, SCHEMAS DE PROGRAMME ET PROGRAMME

Nous allons décrire l'algorithme définitif d'élaboration automatique des noeuds de l'arbre, nous donnons ensuite sa représentation sous la forme de schémas de programme et enfin sa traduction sous la forme d'un programme complet en PL/360 [5]. Les déclarations contiennent les données de l'exemple 1 du paragraphe III.7.

L'algorithme de décomposition est simple, il consiste à itérer sur les 2 actions suivantes :

1) Recherche de la sous-liste de coût maximal
2) Décomposition de celle-ci en 2 sous-listes dont le coût total est minimal.

L'itération s'arrête quand on a obtenu p sous-listes. La décomposition est réalisée en recherchant l'indice k tel que :

$$k \quad \sum_{i=k}^{n-1} F(i) \quad \text{soit minimal}$$

On évalue alors les coûts gauche et droit des 2 sous-listes.

ACTION DECOMPOSER UNE LISTE EN P SOUS-LISTES

```
Initialiser . 1er noeud ← 1er élément gauche
            . coût(1) ← valeur maxi
            . Indice noeud suivant ← longueur de la liste
            . Nombre de noeuds ←1
TANT QUE nombre de noeuds < p
FAIRE . rechercher la sous-liste de coût maximum
  |     . SI coût maximum = 0 ALORS ALLER A FIN
  |     . Rechercher l'élément milieu tel que
  |           Coût minimum = min (coût gauche+coût  droit)
  |     . Ranger le nouveau noeud et les coûts gauche et droit
  |     . Progression du nombre de noeuds
FINFAIRE
FIN : Transfert des résultats
```

ACTION RECHERCHER LA SOUS-LISTE DE COUT MAXIMUM

```
Coûtmax ← 0
POUR I ← 1  JUSQU'A nombre de noeuds
FAIRE SI Coût(I) > coûtmax
  |     ALORS coûtmax ← coût(I)
  |       |      indicemax ← I
  |     FINSI
FINFAIRE
```

ACTION RECHERCHER L'ELEMENT MILIEU (min(coût gauche + coût droit))

```
INF ← indicemax
SUP ← indice noeud suivant-1
Somme fréquences ← 0
Produitmax ← 0          k ← SUP-INF
TANTQUE k > 0
FAIRE somme frequences ← somme fréquence + Fréquence(SUP)
   |      Produit ← k * somme fréquences
   |      SI Produit > Produitmaxi
   |          ALORS Produitmaxi ← Produit
   |           |     k max ← k
   |          FINSI
   |      SUP ← SUP-1        k ← k-1
FINFAIRE
```

$$\text{Calculer coût gauche} = \sum_{i=inf+1}^{k-1} (i-inf).F(i)$$

$$\text{Calculer coût droit} = \sum_{i=k+1}^{SUP-1} (i-k).F(i)$$

FIN ACTION

```
BEGIN COMMENT ELABORATION AUTOMATIQUE DES NOEUDS DE L' ARBRE ;
EXTERNAL PROCEDURE TYPE(R14); NULL;
SEGMENT BASE R11;
ARRAY 2000 LOGICAL ADRDICT=("   A","   B","   C","   D","   E","   F",
"   G","   H","   I","   J","   K","   L","   M","   N","   O","   P",
"   Q","   R","   S","   T","   U","   V","   W","   X","   Y","   Z");
CLOSE BASE;
ARRAY 400 LOGICAL TCOUT =400(0);
    SHORT INTEGER XINF SYN TCOUT, XSUP SYN TCOUT(2);
    LOGICAL COUT SYN TCOUT(4);
LOGICAL PNOEUD =8, LDICT =26;
LOGICAL COUTMAX,XCOUTMAX, LCOUT, NBNOEUD, MILMUN,
        MILMAX, PMAX, SFREQ, CG ,CD, BINF,BSUP;
INTEGER REGISTER MIL SYN R2, I SYN R3, X SYN R4, Y SYN R5,
                 INF SYN R6, SUP SYN R7;
ARRAY 2000 LOGICAL FREQ =(308,206,400,128,190,220,172,108,114,54,22,
  152,284,88,90,254,15,237,312,188,38,122,20,2,10,14);
COMMENT INITIALISATION ;
R0:=0; XINF:=R0; LCOUT:=R0;
X:=_1 SHLL 1 SHRL 1; COUT:=X;
X:=LDICT SHLL 2; XSUP:=X; X:=1; NBNOEUD:=X;
WHILE X < PNOEUD DO
BEGIN  COMMENT RECHERCHE DE LA SOUS-LISTE DE COUT MAXIMAL;
      X:=R0; COUTMAX:=X; XCOUTMAX:=X;
      FOR I:=R0 STEP 8 UNTIL LCOUT DO
      BEGIN Y:=COUT(I); IF Y > X THEN
            BEGIN  COUTMAX:=Y; X:=Y; XCOUTMAX:=I;
            END;
      END;
      IF X = R0 THEN GOTO FIN;
COMMENT RECHERCHE DE L'ELEMENT MILIEU MIN(COUT G + COUT D) ;
      I:=XCOUTMAX; INF:=XINF(I); SUP:=XSUP(I);
      Y:=INF+4; BINF:=Y; Y:=SUP-4; BSUP:=Y;
      PMAX:=R0; MILMAX:=R0; SFREQ:=R0; I:=BSUP-INF SHRL 2;
      SUP:=SUP-4;
      WHILE I > 0 DO
      BEGIN R1:=FREQ(SUP) + SFREQ; SFREQ:=R1; R1:=SFREQ * I;
            IF R1 > PMAX THEN
            BEGIN PMAX:=R1; MILMAX:=SUP;
            END;
            SUP:=SUP-4; I:=I-1;
      END;
COMMENT CALCUL DU COUT GAUCHE ;
            Y:=1; R0:=0; CG:=R0; CD:=R0; X:=MILMAX-4; MILMUN:=X;
            FOR X:=BINF STEP 4 UNTIL MILMUN DO
            BEGIN R1:=FREQ(X) * Y + CG; CG:=R1; Y:=Y+1;
            END;
COMMENT CALCUL DU COUT DROIT ;
            Y:=1;
            FOR X:=MILMAX+4 STEP 4 UNTIL BSUP DO
            BEGIN R1:=FREQ(X) * Y + CD; CD:=R1; Y:=Y+1;
            END;
COMMENT RANGER ET PROGRESSION DE NOEUD ;
       MIL:=MILMAX; X:=CG; Y:=CD;
       I:=XCOUTMAX; COUT(I):=X; R1:=XSUP(I); XSUP(I):=MIL;
       X:=LCOUT+8; LCOUT:=X; XSUP(X):=R1;COUT(X):=Y; XINF(X):=MIL;
       X:=NBNOEUD+1; NBNOEUD:=X;
END;
FIN:  FOR I:=R0 STEP 8 UNTIL LCOUT DO
      BEGIN X:=XINF(I); R1:=@ADRDICT+X; MVI("*",B1); END;
      R0:=@ADRDICT; R1:=105; TYPE;
END.
```

III.7. EXEMPLES

Exemple 1 : Nous avons choisi un exemple simple qui permet d'illustrer l'algorithme.

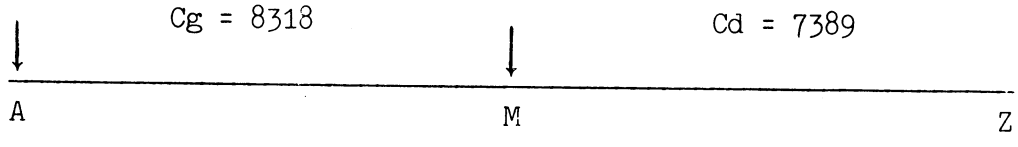
Dans un dictionnaire français en 4 volumes, nous avons compté les pages contenant les mots qui commencent par la lettre A, puis B,..., jusqu'à Z. Nous obtenons un ensemble ordonné de 26 éléments auxquels on associe une fréquence équivalente au nombre de pages trouvées.

Mot	Fréquence	Mot	Fréquence
A	308	N	88
B	206	O	90
C	400	P	254
D	128	Q	15
E	190	R	237
F	220	S	312
G	172	T	188
H	108	U	38
I	114	V	122
J	54	W	20
K	22	X	2
L	152	Y	10
M	284	Z	14

Le calcul de $k \quad \sum_{i=1}^{n-1} F(i)$ nous donne un maximum pour le mot M

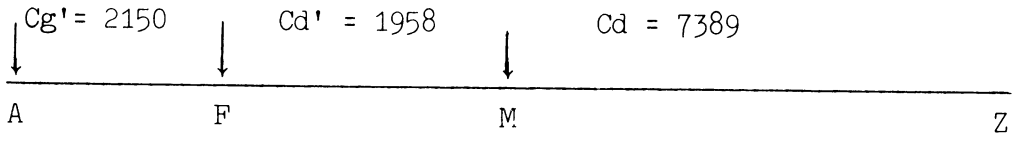
Soit un coût gauche = 8318 et un coût droit = 7389 pour 2 noeuds

coût total = 8318 + 7389 = 15707

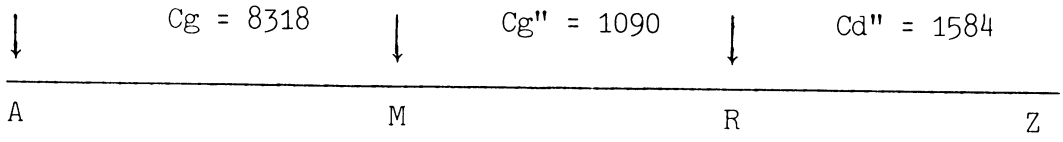


Quand on passe à un arbre d'accès de 3 noeuds, on détermine le mot F dans la liste de coût maximum, ce qui donne un coût total de

CT = 2150 + 1958 + 7389 = 11.497



On peut constater sur cet exemple que si on découpe la liste de coût minimum, on trouve le mot R et le coût total est alors inférieur au précédent : CT = 8318 + 1090 + 1584 = 11 002



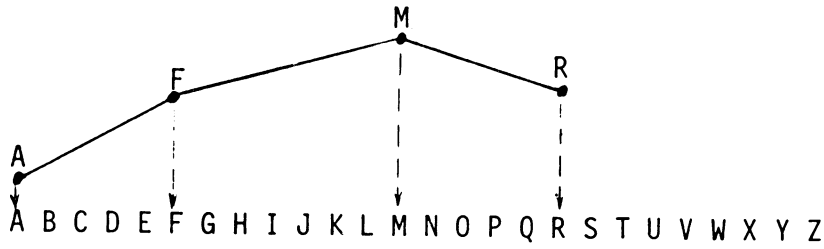
En continuant à ajouter des noeuds dans l'arbre d'accès, on constate qu'il est raisonnable de décomposer la sous-liste de coût maximum à chaque pas et que l'on tend vers un coût total minimum.

Arbre d'accès de 4 noeuds (A,F,M,R) en prenant les listes de coût maximum : coût total = 2150 + 1958 + 1090 + 1584 = 6782

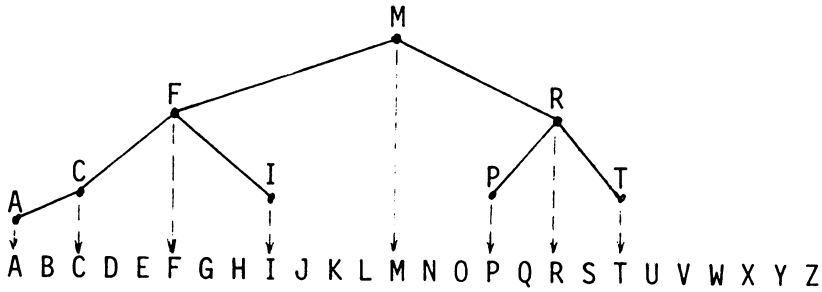
Arbre d'accès de 4 noeuds (A,M,P,R) en prenant les listes de coût minimum, le coût total est de : 8318 + 268 + 15 + 1584 = 10185

Au pas suivant, on prendrait le mot Q et ensuite, on serait bloqué.

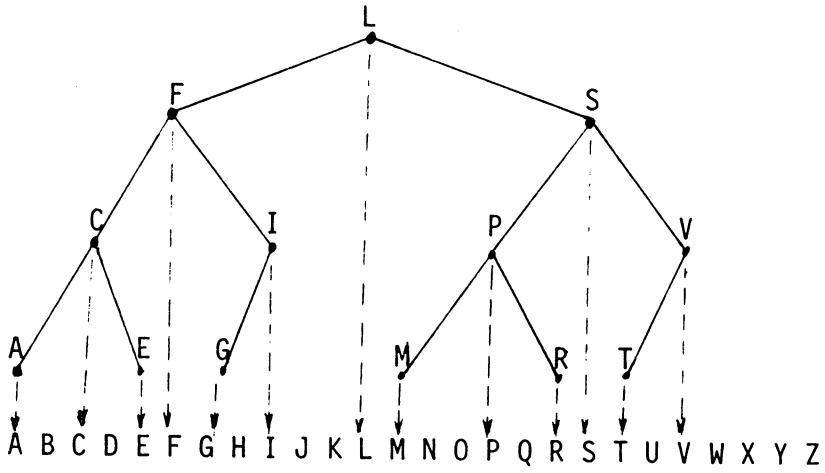
Nous avons obtenu les structures suivantes :



P = 4 noeuds



P = 8 noeuds



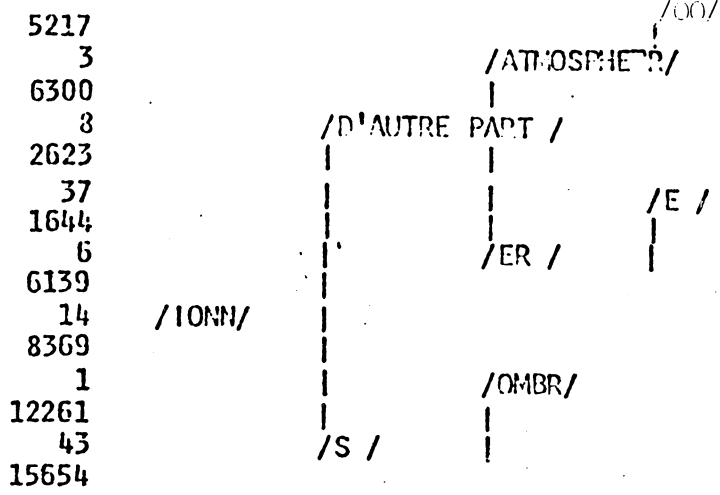
P = 13 noeuds

Exemple 2 :

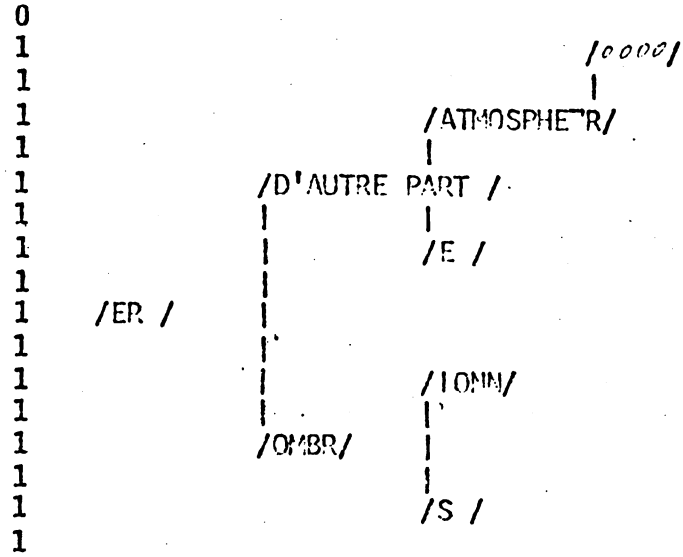
Arbres obtenus pour l'application réelle (courbes 8 et 9) avec un dictionnaire de 2000 éléments dont 1600 éléments "frère".

Dans la colonne de gauche on trouve la fréquence α des éléments "noeuds", suivie de la valeur du coût de l'intervalle avec le noeud suivant.

Arbre binaire de 8 noeuds construit avec l'algorithme de KNUTH

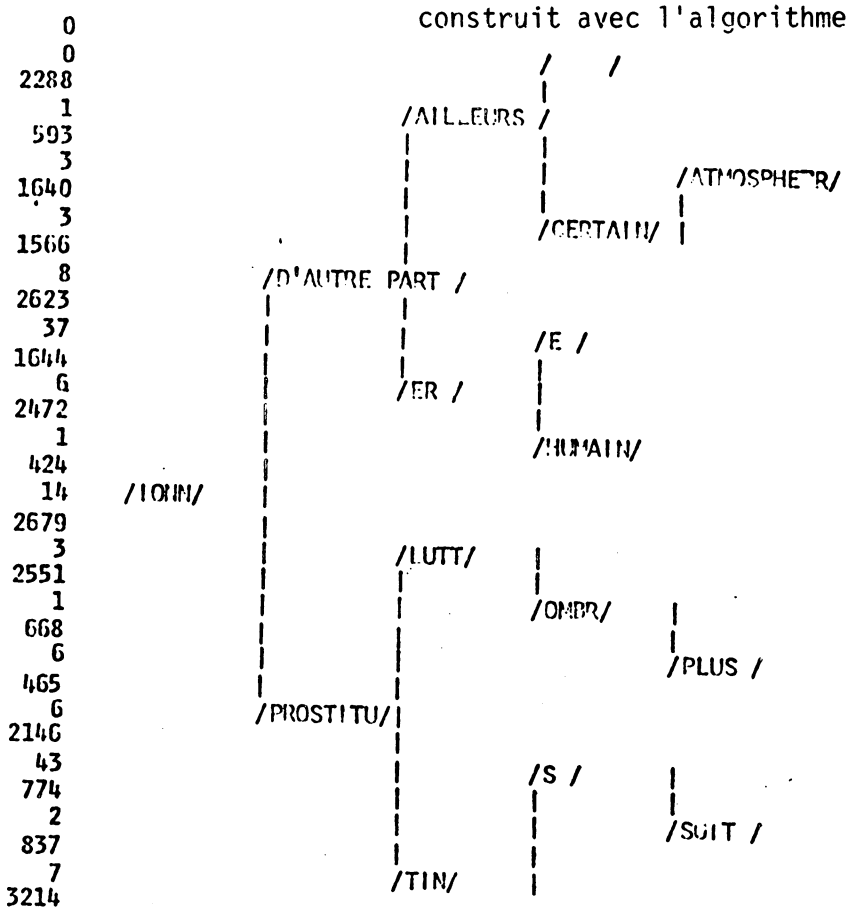


Arbre binaire équilibré de 8 noeuds

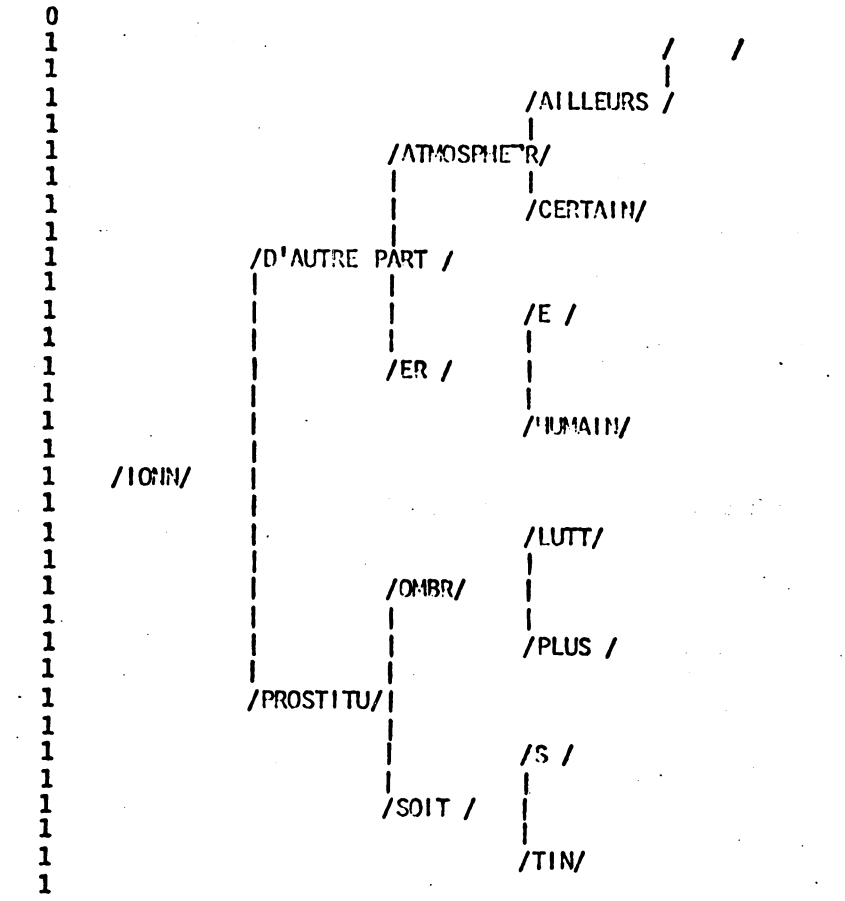


ARBRE BINAIRE DE 16 NOEUDS

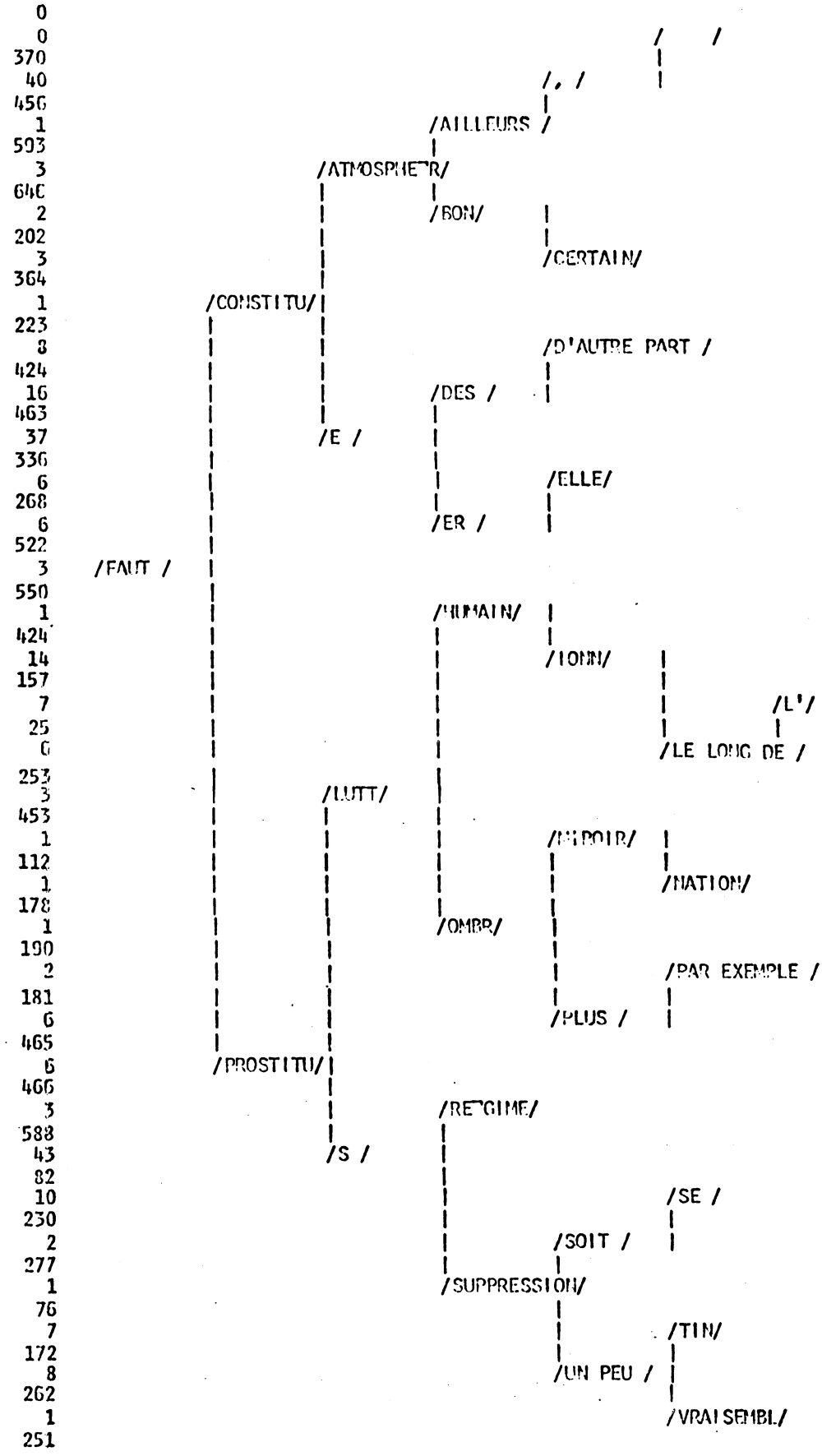
construit avec l'algorithme de KNUTH



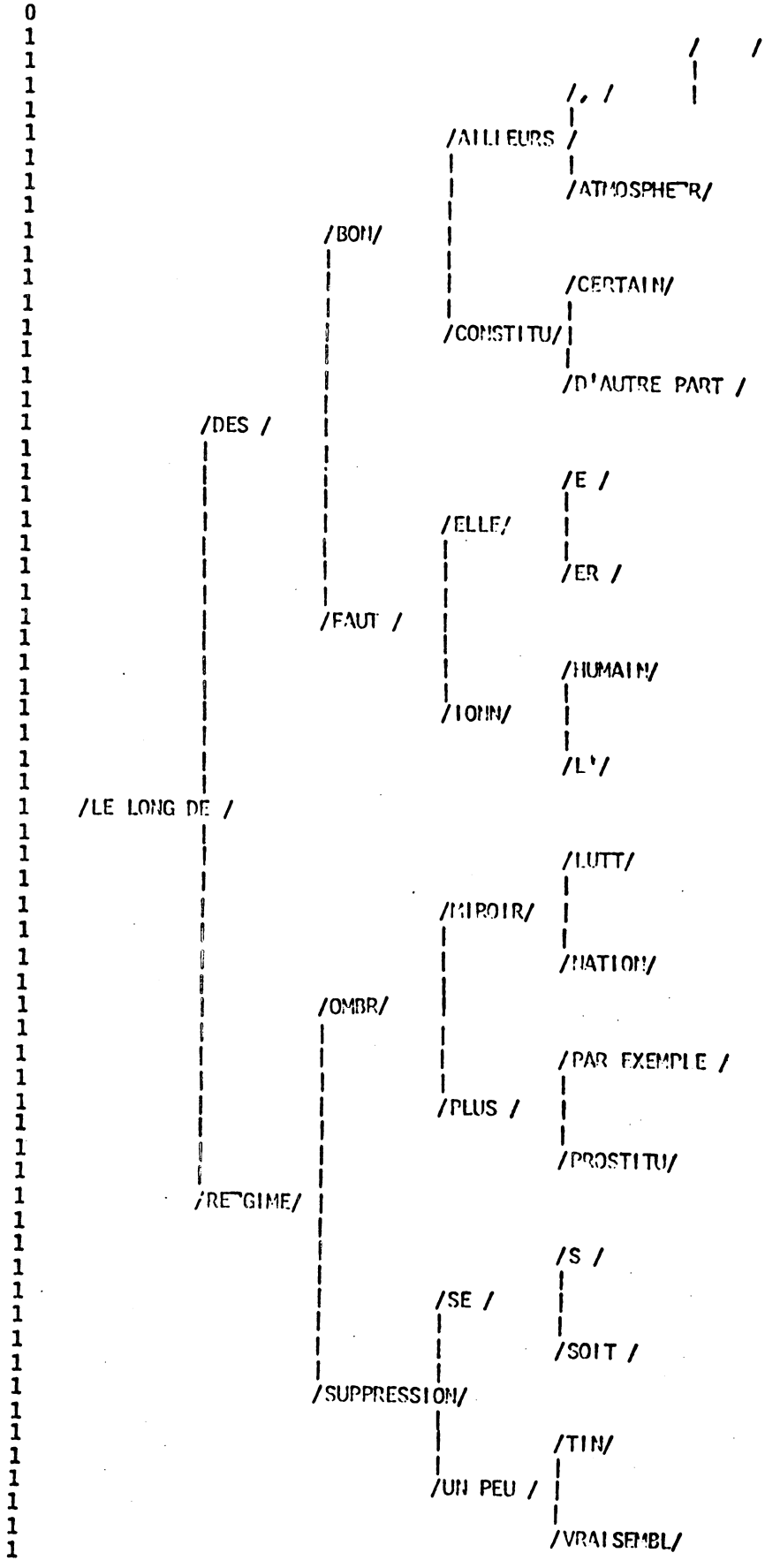
ARBRE BINAIRE EQUILIBRE DE 16 NOEUDS



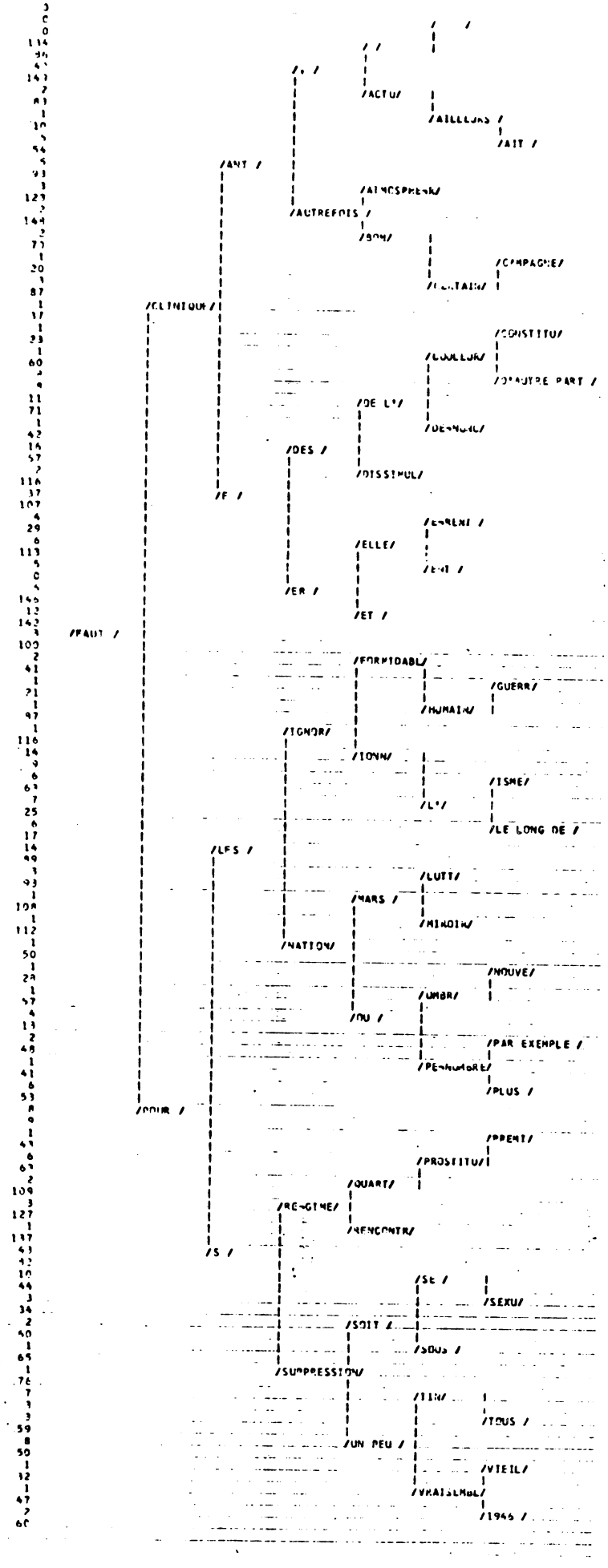
ARBRE BINAIRE DE 32 NOEUDS (KNUTH)



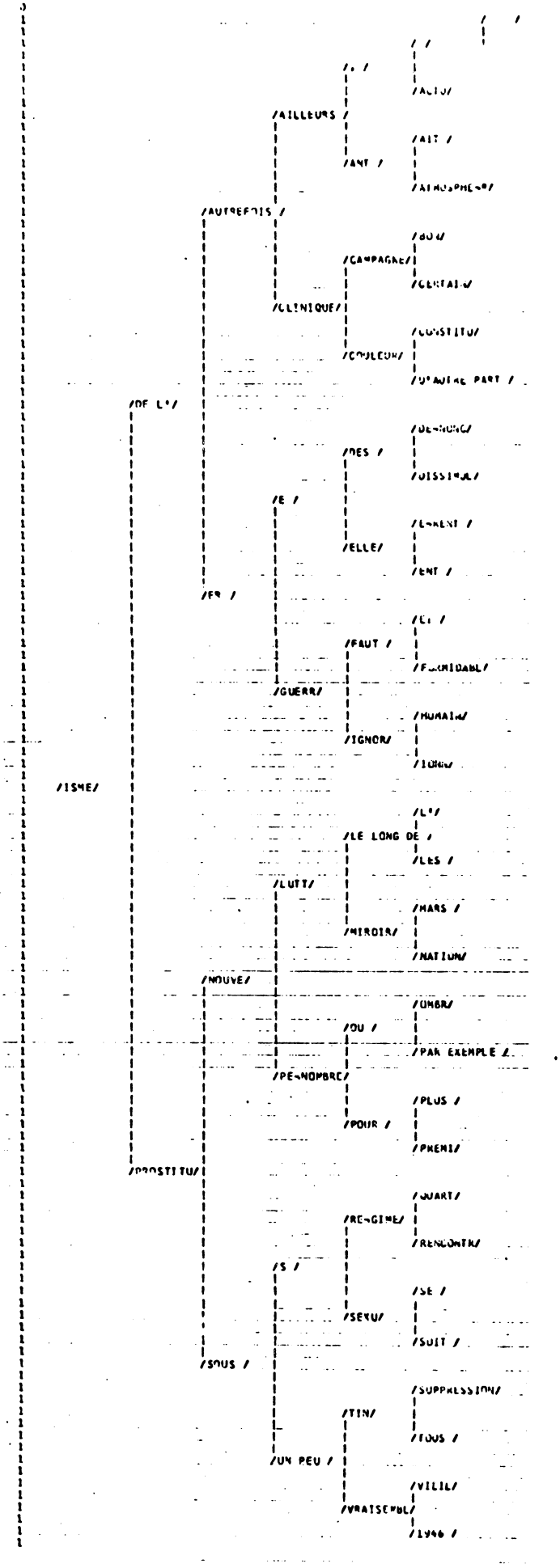
ARBRE BINAIRE EQUILIBRE DE 32 NOEUDS



ARBRE BINAIRE DE 64 NOEUDS (KNUTH)



ARBRE BINAIRE EQUILIBRE DE 64 NOEUDS



III.8. CONCLUSION SUR LE CHAPITRE III

Nous avons résumé dans un tableau les résultats des différentes méthodes déjà illustrées par les courbes précédentes.

Tableau des coûts en secondes

Méthodes	nombre de noeuds / courbes	4	8	16	32	64	120
Référence	0	37,4	20,5	12,1	8,05	6,2	5,4
Dichotomie simple	1	33	22	13	8,5	7,1	5,9
Mots les plus fréquents — KNUTH — α réel $\beta = 1$	2	93	30	20,5	10,5	6,5	4,8
Mots les plus fréquents — KNUTH — α réel β réel	3	90	28	20	10	6,1	4,8
Classes de fréquence égale — $\beta = 1$	4	43,7	22	12,3	8,2	6,2	5
Classes de fréquence égale — β réel	5	44	22	12,3	7,9	6	4,8
Classes de coût égal — KNUTH	6	32,3	19,7	11,1	7,1	5,65	4,9
Classes de coût égal — arbre équilibré	7	32,7	20	11,3	7,2	5,7	5
Classes de coût minimal — KNUTH	8	30	16,3	9,4	6,3	5,2	4,7
Classes de coût minimal — arbre équilibre	9	30	16,5	9,7	6,4	5,25	4,7

Nous rappelons que ces mesures concernent un corpus de 6520 mots découpés en 10200 éléments identifiés dans un dictionnaire de 2000 éléments dont 1600 éléments "Frères".

L'algorithme d'identification est resté identique, seul l'arbre d'accès et en conséquence, les classes varient entre les différentes méthodes.

1) Nous pouvons constater la validité de la méthode théorique de référence dont les valeurs sont "encadrées" par celles des autres méthodes.

Cette méthode peut permettre de donner une estimation d'un accès par une méthode dichotomique simple avec un arbre équilibré.

2) Nous constatons un gain sensible entre la méthode des classes de coût minimal et la méthode dichotomique simple (18 % en moyenne). Toutefois, ce gain n'est pas négligeable si l'on considère une application demandant de nombreux accès au dictionnaire.

3) Enfin, nous voyons que l'algorithme de Knuth est cher et inutile pour l'application considérée. La construction de l'arbre est onéreuse en volume mémoire (limité à 120 noeuds) et en temps puisque l'algorithme est de $O(n^2)$ et de plus il impose un choix manuel des noeuds. Pour effectuer les mesures, nous avions choisi les mots les plus fréquents (écrétage).

4) Nous avons défini un algorithme efficace qui permet de sélecter automatiquement les noeuds à placer dans l'arbre d'accès de telle sorte que le coût total soit proche du coût minimal. On peut alors construire simplement un arbre binaire équilibré.

Si l'on veut vraiment tenir compte des coûts, on peut construire l'arbre binaire au moyen de l'algorithme de KNUTH qui donne le parcours minimal dans l'arbre.

C H A P I T R E IV

SYSTEME DE GESTION DU DICTIONNAIRE

Le système est transparent car il doit être accessible par des utilisateurs non informaticiens (documentalistes, enseignants, linguistes, ...).

Nous avons constaté qu'au cours du fonctionnement normal, il n'y a pas de difficulté particulière puisque les ajouts ou suppressions sont réalisés par une commande automatique après une équivalence. Il nous a fallu néanmoins prévoir un système de gestion indépendant de l'éditeur morphologique et syntaxique pour réaliser certaines opérations particulières dans l'ordre et entre plusieurs sessions.

1) Création du dictionnaire initial (départ à froid)
2) Extension et mise à jour de l'arbre
3) Réorganisation de l'arbre en fonction des modifications précédentes et des nouvelles fréquences.
4) Extension et mise à jour en mode manuel des classes
5) Réorganisation des classes.

IV.1. ENVIRONNEMENT DICTIONNAIRE

Dans l'environnement dictionnaire, le programme de contrôle demande à l'utilisateur les paramètres et les données et prévient celui-ci en cas de traitement irréversible, de traitement impossible ou de paramètre erroné.

La demande de paramètres peut être explicitée sous la forme d'un schéma de programme :

```
SI Fichier dictionnaire existe
ALORS SI Dictionnaire réorganisé ALORS EXTENSION DES CLASSES
  |     SINON   Question : réorganisation ?
  |       |     SI OUI ALORS Réorganisation
  |       |                    |   Extension des classes
  |       |     SINON Extension de l'arbre
  |       |     FINSI
  |     FINSI
  |
SINON création
  |     indicateur ¬ réorganisation
  |     extension
FINSI
```

ACTION EXTENSION DES CLASSES

```
    Question : Extension des classes ?
    SI OUI ALORS extension
           SINON extension de l'arbre
```

FINACTION

ACTION EXTENSION DE L'ARBRE

```
Question : EXTENSION DE L'ARBRE ?
SI OUI
ALORS INDICATEUR ¬REORG
  |     EXTENSION
SINON Question : effacement ?
  |     SI EFFAC
  |     ALORS CREATION
  |       |     INDICATEUR ¬REORG
  |     FINSI
FINSI
```

FINACTION

La demande de données.

Elle est caractérisée par l'état ">".

Un petit langage permet alors les extensions, suppressions, remplacements et interrogations du dictionnaire.

La frappe du signe ? provoque l'apparition au terminal de ce langage.

La règle générale est de toujours encadrer une chaîne de caractères par le signe "/".

IV.2. LANGAGE ET EXEMPLES

Langage :

a) Extension c'est-à-dire : Ajout d'une nouvelle chaîne (base)
avec ses attributs
ou Remplacement des attributs d'une chaîne

dans l'ARBRE =>/BASE{(*)}/MODELE/{chaînage ou vide/n m}
dans les CLASSES =>/BASE{(*)}/MODELE/{CHAINAGE/}

Les éléments entre crochets sont facultatifs.

b) Suppression d'une chaîne (base) et de ses attributs.

- /BASE/

c) **Interr**ogation de l'environnement d'une chaîne (base)

Impression au terminal de la chaîne demandée ou celle de longueur > et de toutes les chaînes de longueur < , ainsi que de la chaîne suivante.

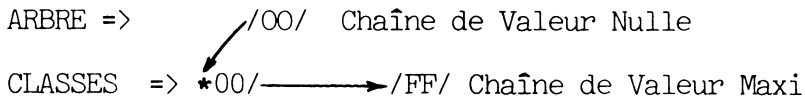
?/BASE/

Par convention, nous appelons BASE la chaîne de caractères contenue dans le dictionnaire. Ses attributs sont :
- le MODELE morphologique auquel la BASE fait obligatoirement référence
- l'indicateur de fin de solutions (*). Forme indécoupable
- le ou les CHAINAGES circulaires avec d'autres bases (génération).
- les nombres n et m représentent respectivement la fréquence de la base et celle des basesde l'intervalle suivant. Ces valeurs sont permises dans l'environnement "extension de l'arbre" et seront prises en compte par le programme de réorganisation de l'arbre.

Opérations sur le dictionnaire : Etat ">"

1) Départ à froid ou effacement du dictionnaire courant

Initialisation automatique de l'arbre et des classes avec des chaînes de valeurs limites mini et maxi et inaccessibles depuis le terminal.

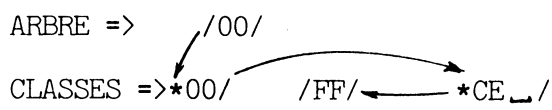


* signifie que la chaîne /00/ est pointée par l'arbre.

2) Extension de l'arbre : Introduction de données au terminal
Construction virtuelle de l'arbre

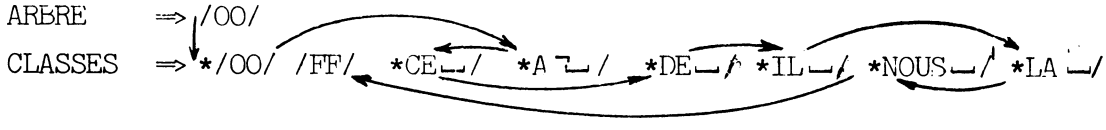
Soit a) /CE␣/CE␣// 7 0

On obtient :



Soit b) /A¬␣/DE␣// 5 0
c) /DE␣/DE␣// 10 0
d) /IL␣/IL␣// 7 0
e) /NOUS␣/NOUS // 8 0
f) /LA␣/LA␣// 7 0

On obtient :

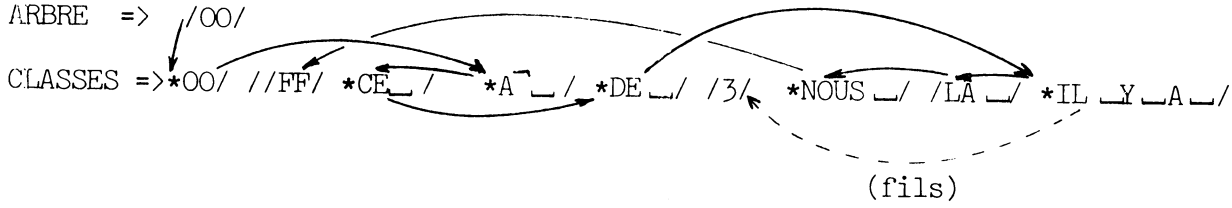


3) Extension de l'arbre au cours d'une deuxième session
Modification du contenu de l'arbre virtuel

a) Supprimer : ¬/LA ␣/

b) Ajouter : / IL ␣Y␣ A␣ /IL Y A// 7 0

on obtient :



4) Cas particuliers sous l'état : extension de l'arbre

	Questions	Réponses	
a)	¬/ZUT/	IMPOSSIBLE	N'existe pas
b)	¬/LA ␣/	IMPOSSIBLE	N'est pas dans l'arbre
c)	/CE ␣/CECI ␣/	1 *CE ␣/CE ␣/	

PRECISEZ LE NUMERO DE BASE

Si 0 → Annulation de la donnée /CE ␣/CECI ␣/

Si 1 → Remplacement des attributs (modèle CECI au lieu de CE)

Si 2 → IMPOSSIBLE Pas d'homographe dans l'arbre

d) ?/IL ␣/ → *IL ␣Y␣A ␣/IL Y A/ /LA ␣/LA ␣./

/IL ␣/

e) /LA ␣/LA ␣// 4 0 La base /LA ␣./ est remise dans l'arbre avec la fréquence 4.

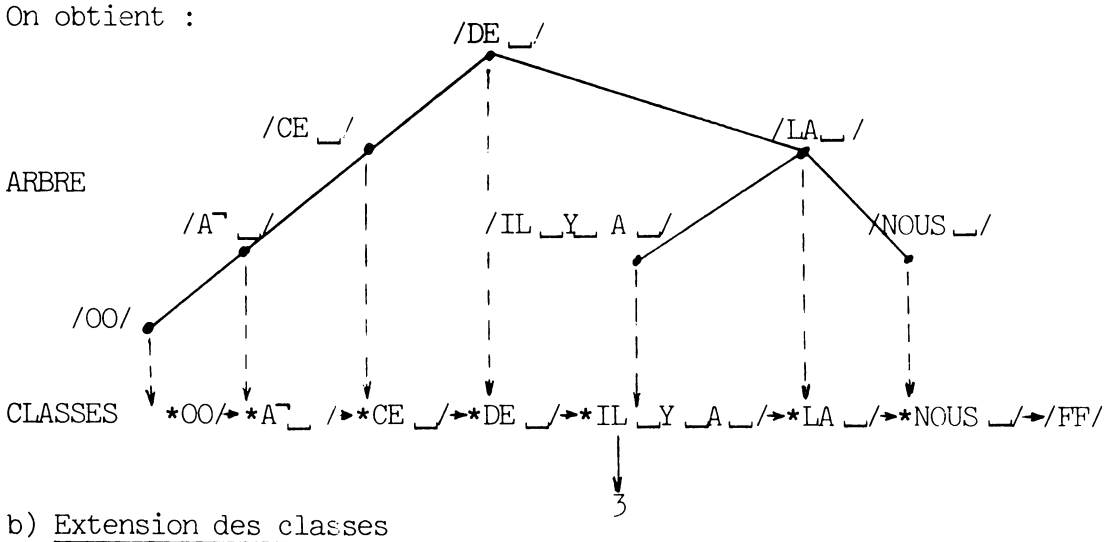
Quand on crée l'arbre avec des chaînes dont la fréquence est inconnue :

- si on indique la valeur 0 pour n et m, l'algorithme construira un arbre où tous les éléments seront en liste (chaîne)
- si on omet n et m, la valeur sera prise égale à 1 et l'algorithme construira un arbre binaire équilibré (dichotomie).

5) Extension des classes

a) Réorganisation du dictionnaire : Construction réelle de l'arbre.

On obtient :

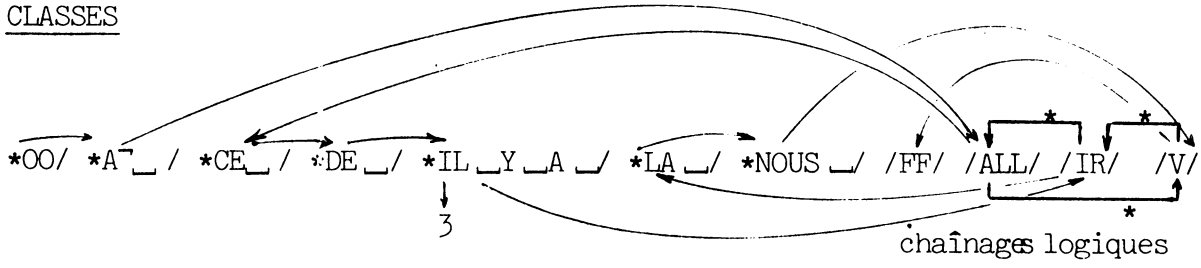


b) Extension des classes

Les manipulations de chaînes portent uniquement sur le contenu des classes et il est impossible de modifier les chaînes pointées par l'arbre.

Questions	Réponses	
¬/CE _/	IMPOSSIBLE	Chaîne pointée par l'arbre
¬/ILS/	IMPOSSIBLE	N'existe pas
/ALL/ALL/		1ère base du verbe ALLER
/IR/IR/ALL/		2ème base du verbe ALLER au futur chaînée à ALL
/V/V/IR/		3ème base du verbe ALLER au présent chaînée à IR

CLASSES



c) Test automatique du % de désordre

Quand le pourcentage dépasse le seuil fixé (paramètre), il y a réorganisation automatique des CLASSES, l'arbre est inchangé..

*)O/→*A␣ /→/ALL/→*CE␣/→*DE␣/→*IL ␣Y ␣A ␣/→/IR/→*LA ␣/→*NOUS␣/→/V/→/FF/

3

d) Cas particuliers sous l'état "extension des classes"

Questions		Réponses
/DE␣TERRE␣/DE␣/		IMPOSSIBLE — Chaîne plus longue que /DE / qui est dans l'arbre. Il faut se placer dans l'état "extension de l'arbre"
/ZUT/ZUT/		LE MODELE N'EXISTE PAS
/CE␣/CECI␣/		1 *CE␣/CE␣/ PRECISEZ LE NUMERO DE BASE >
Si 0	→	Annulation de la donnée /CE␣/CECI␣/
Si 1	→	Remplacement des attributs (modèle CECI au lieu de CE)
Si 2	→	Ajout d'une 2ème base homographe pointant sur le modèle /CECI␣/
/IL␣Y␣A␣(*)/IL Y A/		1 *IL␣Y␣ A ␣/IL Y A/ PRECISEZ LE NUMERO DE BASE 1 Attribut (*)
?/IL␣Y␣A/		*IL␣Y␣A␣(*)/IL Y A/ /IR/IR/ALL/ /IL␣ /IL␣␣/
?/ALL/		/ALL/ALL/V/ *CE␣/CE␣/

Paramètres des programmes

1) Pourcentage de désordre : le programme "DICTIO" contient la valeur de ce seuil et il active en conséquence la réorganisation des classes.

Ecriture : EQUATE POURCENTOLERE SYN 20 ;

2) Nombre de chaînages logiques

Il est contenu dans les programmes "DICTIO", "LISTDICT" et "REORDICT" :

EQUATE NOMBREDEPOINTEURS SYN 1 ;

et dans le programme "EXTENS" :

EQUATE NCHAIN SYN 1 ;

Ce nombre de chaînage est toutefois limité par le nombre de caractères de la ligne d'entrée.

IV.3. PROGRAMMES DE SERVICE

Programme de listage

Un premier programme permet d'obtenir le dessin de l'arbre binaire où chaque noeud est un élément du dictionnaire.

Les fréquences α et β de chaque noeud et de leur intervalle apparaissent en colonne à gauche du dessin.

Un second programme fournit la liste des éléments "frère " et "fils" dans l'ordre alphabétique habituel.

Chaque élément apparaît suivant le format d'entrée; c'est-à-dire suivit de {*}, modèle et indication d'appartenance à l'arbre.

Programme de regénération

Un ensemble de commandes, contenues dans un fichier "EXEC" permettent la regénération d'un dictionnaire à partir d'une file séquentielle d'enregistrements.

Ces enregistrements contiennent des informations dans le format d'entrée :

/BASE{(*)}/MODELE/{chaînage/ n ri}

ou /BASE{(*)}/MODELE/{chaînage/}

Cette file séquentielle peut être obtenue par le second programme de listage ou créée à partir de cartes ou d'un terminal (mode Batch) ou encore par fusion de plusieurs fichiers "dictionnaire".

Ce processus assure la "portabilité" d'un dictionnaire entre 2 machines via une bande magnétique ou même des cartes (support extérieur).

De plus, nous pouvons éventuellement protéger le dictionnaire contre des pannes systèmes ou contre des pannes de nos programmes.

Programme de réorganisation

L'utilisation des pointeurs nous a permis de ne réserver qu'une seule zone de débordement contiguë tout en assurant l'ordre alphabétique des éléments.

Au moment du chargement du fichier dictionnaire en mémoire, nous réservons un espace mémoire égal à la taille du dictionnaire augmenté de la taille de la zone de débordement.

A la fin d'une session ou quand la zone de débordement est remplie, (nous avons volontairement limité sa taille à 2000 octets par mesure de sécurité en cas de pannes du système) le nouveau dictionnaire (ancien + ajouts) est réécrit sur un fichier (disques).

La procédure de réécriture n'est activée que si des modifications ont été réalisées au cours de la session. Elle contient une séquence "d'évaluation du désordre" c'est-à-dire le rapport entre le nombre de déplacements vers des adresses absolues supérieures et le nombre de déplacements vers des adresses absolues inférieures.

Quand ce pourcentage de désordre atteint un certain seuil, (paramètres) la procédure de réorganisation des classes est activée.

Le parcours du dictionnaire sera alors réalisé par des déplacements vers des adresses absolues croissantes, la pagination sera minimale et de plus, la place laissée par les suppressions sera récupérée ("garbage collector").

Etant donnée la taille de notre dictionnaire actuel (14 pages) et le fait que nous disposons d'une très grosse machine et que le nombre de pages allouées varie dynamiquement pour un utilisateur, les mesures de temps de pagination que nous avons effectuées ne sont pas significatives.

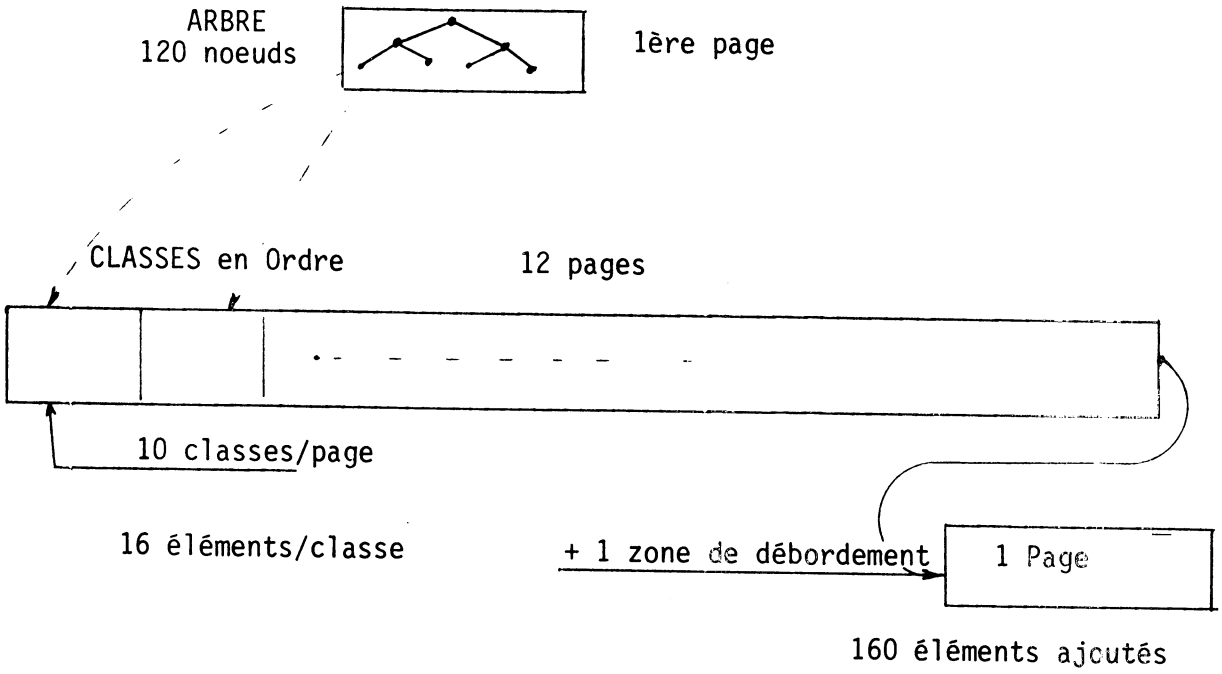
IV.4. ESTIMATION DE LA PAGINATION

On peut toutefois donner quelques estimations de la pagination pour un dictionnaire de 2000 éléments, contenu en mémoire dans 52000 octets. Pour cela, nous avons fait les deux hypothèses suivantes :

1) On a supposé une répartition homogène des fréquences, de manière à obtenir des classes de même taille,

2) On peut envisager une répartition homogène des éléments ajoutés dans chaque classe. On peut alors dire que le pourcentage de désordre est proportionnel au volume des ajouts, c'est-à-dire à la taille de la zone de débordement.

Nous avons l'implantation suivante : avec des pages de 4000 octets.



On voit qu'il sera intéressant de charger l'arbre une seule fois en mémoire. Le nombre de noeuds étant fonction de la taille de la page.

Il faut une page pour la classe et une page pour la zone de débordement. Il sera donc raisonnable d'avoir au minimum 3 pages allouées

Pages de 4000 octets - 120 noeuds dans l'arbre
10 classes/page 16 éléments/classe

% de désordre	8 %	16 %	32 %	50 %	80 %	100 %
Nombre de pages en ordre	12	11	9	6	3	1
Nombre de pages de débordement	1	2	4	7	10	12
Appel pages ordonnées	1	1	1	1	1	1
Appels de pages de débordement	1	2	4	7(8)	10(13)	12(16)

Pages de 2000 octets - 64 Noeuds dans l'arbre 1 page
2,5 classe par page - classes 25 pages - 31 éléments/classe

% de désordre	4 %	8 %	16 %	50 %	80 %	100 %
Nombre de pages en ordre	25	24	22	13	5	1
Nombre de pages de débordement	1	2	4	13	21	25
Appel pages en ordre	1	1	1	1	1	1
Appels de pages de débordement	1	2	4	13(15)	21(25)	25(31)

Pages de 1000 octets - 32 Noeuds dans l'arbre 1 page
0,65 classe/page - classes 51 pages - 62 éléments/classe

% de désordre	2 %	4 %	8 %	50 %	80 %	100 %
Nombre de pages en ordre	51	50	48	26	10	1
Nombre de pages de débordement	1	2	4	26	42	51
Appel pages en ordre	2	2	2	2	2	1
Appels de pages de débordement	1	2	4	26(31)	42(50)	51(62)

IV.5. DICTIONNAIRE PERMANENT et DICTIONNAIRES SATELLITES

Nous envisageons d'améliorer les possibilités des programmes en autorisant la concaténation d'un dictionnaire permanent ou principal, contenant le vocabulaire courant de la langue, avec un nombre variable de dictionnaires satellites ou secondaires, contenant les vocabulaires spécialisés.

Le programme de regénération aurait la charge d'effectuer la fusion du dictionnaire permanent et des dictionnaires satellites demandés par l'utilisateur en fonction du type de corpus à analyser.

IV.6. CHOIX DES NOEUDS DE L'ARBRE

S'il était possible de définir manuellement les 50 premiers mots invariables d'usage courant à placer dans l'arbre, le problème du choix des éléments suivants devenait très délicat car on devait tenir compte simultanément des fréquences α et β (Algorithme de KNUTH). La complexité de ce problème augmente dans le cas de fusion d'un dictionnaire permanent et de dictionnaires satellites.

C'est une des raisons pour lesquelles nous avons donc été amenés à rechercher un algorithme de choix des noeuds et de construction automatique de l'arbre.

IV.7. PROGRAMMATION

Nous avons cherché un langage de programmation évolué, disponible sous CP/CMS qui permette le traitement des chaînes de caractères, des chaînes de bits, la manipulation des pointeurs et enfin pour être efficace qu'il laisse au programmeur la gestion des registres.

Nous avons choisi le langage PL/360 qui est proche du langage ASSEMBLEUR mais qui possède des primitives évoluées (si, tant que) et une structure de bloc.

Les algorithmes ont été programmés sous la forme de procédures globales. On utilise les conventions de liaison standard du système superviseur OS/360. Ce qui permet une imbrication infinie d'appels de procédures et autorise l'emploi de procédures écrites dans un autre langage telles que les procédures GSP FORTRAN du système graphique (2250) ou les procédures d'Entrée/sortie écrites en Assembleur.

Du point de vue de la portabilité sur un autre matériel nous avons constaté que le langage PL/360 était proche des langages LP70 (CII), PL1600(Téléméca nique),... ce qui facilitera beaucoup les réécritures de programmes s'il y a lieu.

Du point de vue de la taille des programmes :
EXTENSION - 4000 octets
REORGANISATION - 4000 octets (KNUTH+ recopie DICT).

Les autres modules environnement, listage, chargement et stockage sont très petits.

Quant aux temps d'exécution des programmes, nous avons déjà donné l'essentiel en ce qui concerne l'identification dans le dictionnaire (voir le chapitre précédent).

Nous pouvons indiquer les temps du programme de réorganisation de l'arbre, c'est-à-dire l'exécution de l'algorithme de KNUTH avec la construction de l'arbre :

Nombre de noeuds	16	32	64	100	120
Temps en secondes	0,25	0,32	0,6	1,2	1,51

Les temps de l'algorithme de sélection automatique des p noeuds donnant des classes de coût minimal ainsi qu'un arbre binaire équilibré sont :

Nombre de noeuds	16	32	64	100	120
Temps en secondes	0,22	0,26	0,34	0,39	0,43

CONCLUSION

Nous avons construit un dictionnaire interactif qui répond aux critères particuliers imposés par l'analyse des langues naturelles.

La mise à jour en mode interactif est un avantage important que permet cette structure de données.

L'utilisation des pointeurs et d'un arbre d'accès ont permis d'obtenir un coût d'identification satisfaisant tout en assurant une taille raisonnable du dictionnaire.

L'algorithme de KNUTH permet la construction d'un arbre binaire de coût minimal. Le problème du choix des noeuds de cet arbre nous a conduit à proposer un algorithme d'élaboration automatique de ces noeuds en fonction de la fréquence d'appel des éléments du dictionnaire.

Cet algorithme est complémentaire de celui de KNUTH, car il permet de minimiser le coût du balayage séquentiel dans les classes d'équivalence. L'importance de ce coût devient grande par rapport à celle de l'arbre quand le nombre de noeuds diminue ou quand le nombre total d'éléments augmente. Il est alors suffisant de construire un arbre binaire équilibré et on peut éviter d'utiliser l'algorithme de KNUTH.

Nous devons néanmoins admettre une insuffisance des mesures au sujet de la pagination pour laquelle nous avons seulement donné des estimations.

Le problème était double et onéreux : il fallait construire des échantillons de dictionnaire en désordre et réduire la taille de la machine réelle ou écrire un programme de simulation.

En ce qui concerne les mesures de coût total, il aurait peut être été souhaitable de poursuivre les mesures avec d'autres corpus et d'autres dictionnaires plus importants afin de mieux préciser l'influence des algorithmes proposés.

L'utilisation courante de ce dictionnaire est celle de l'analyse des langues naturelles. Nous envisageons une application pour des historiens (étude de la généalogie des noms de famille). Il est raisonnable de penser à des applications dans le domaine de la gestion et des systèmes, chaque fois que se pose les problèmes d'accès à des listes importantes, contenant des éléments ayant une fréquence d'appel variable.

Il est envisagé de transporter ce dictionnaire sur une petite machine en vue de l'enseignement des langues, par exemple. Les problèmes de pagination seront alors à étudier.

Enfin, l'aspect "réseau" apporté par les chaînages logiques laisse entrevoir des applications futures intéressantes comme la validation de données linguistiques (correcteurs), l'établissement de concordances et la génération syntaxique.

B I B L I O G R A P H I E

[1] J.R. ABRIAL-J. BAS-G.BEAUME-G.HENNERON-R.MORIN-G.VIGLIANO -
Projet SOCRATE - Spécifications Générales (1970)

[2] J.COURTIN - Organisation d'un dictionnaire pour l'analyse morphologique
Séminaire de Théorie des Automates et Traitement Automatique des Langues, Février 1973

[3] J.COURTIN-E.GRANDJEAN - Editeur lexicographique pour les langues naturelles.
Séminaire de Programmation, mai 1973

[4] J.COURTIN-P.SGALL-J.C.RIEU - Un métalangage pour l'analyse morphologique
Doc. CETA G.2500, 1970

[5] J.COURTIN-J.VOIRON - Introduction à l'algorithmique et aux structures de données.
Cours, 1974

[6] GOUGENHEIM-MICHEA-RIVENC-SAUVAGEOT - Elaboration du français élémentaire
Etude sur l'établissement d'un vocabulaire et d'une grammaire de base.
Paris, Didier 1956-64

[7] E.GRANDJEAN - Compilateur et analyseur syntaxique
Doc. CETA - GTc17-GTc18, 1967-68

[8] F.R.A. HOPGOOD - Techniques de compilation
Dunod, 1970

[9] D.E. KNUTH - The art of computer programming
Vol.1. - Fundamental Algorithms
Vol.3 - Sorting and Searching

[10] D.E. KNUTH - Optimum Binary Search Trees
Acta Informatica 1, 14-23 (1971)

[11] E.S. PAGE et L.B. WILSON : INFORMATION Representation and manipulation in a Computer
Cambridge, University Press 1973

[12] G. VEILLON - Consultation d'un dictionnaire et analyse morphologique en Traduction Automatique
Thèse de 3ème cycle, 1962

[13] G. VEILLON - Modèles et algorithmes pour la traduction automatique
Thèse d'état, 1970