Mise à contribution de plusieurs classifieurs pour la reconnaissance de textes multifontes

Use of Many Classifiers for Multifont Text Recognition

par A. BELAID [1] **et J.C. ANIGBOGU** [2]

[1] *CRIN-CNRS & INRIA Lorraine, Bâtiment Loria, Campus Scientifique - B. P. 239 54506 Vandœuvre-Lès-Nancy Cedex, Email : abelaid,anigbogu@loria.fr*

[2] *Schlumberger, Austin Systems Center 8311 North FM 620 Road, P.O. Box 200015 Austin, TX 78720-0015 U.S.A. Email : Anigbogu@austin.Wireline.sPb.com*

Résumé

Nous présentons dans cet article un système de reconnaissance de caractères multifontes utilisant plusieurs classifieurs. Chaque classifieur fournit une réponse puis le résultat final est obtenu par vote majoritaire. Les classifieurs sont de deux types : *stochastique* et *plus proche voisin*. Les classifieurs stochastiques sont des modèles de Markov cachés du premier et du second ordre. La reconnaissance des caractères est suivie d'un module de vérification lexicale qui utilise un modèle de Markov caché pour les mots dont les paramètres sont déterminés à partir de statistiques sur la langue et d'un dictionnaire.

Mots clés : Reconnaissance de caractères multifontes, Reconnaissance de la fonte, Modèles de Markov cachés, Algorithme de Viterbi, Vote majoritaire.

Abstract

*We present in this paper a character recognition system using many classifiers. Each classifier gives an answer and the final result is selected by majority-vote. The system uses six classifiers built around first and second order hidden Markov models (*HMM*) as well as nearest neighbor considerations. The majority-vote is chosen so as to give better results than each of the other systems applied individually. The recognition process is followed by a post-processing which employs combinations of stochastic and dictionary verification methods for word recognition and error-correction.*

***Key words :** Multifont Character Recognition, Font Recognition, Hidden Markov Models, Viterbi Algorithm, Majority-Vote.*

1. Introduction

L'analyse de documents a fait son entrée dans les administrations et les bureaux d'étude depuis quelques années. Certains aspects fonctionnent correctement, tandis que d'autres nécessitent des efforts supplémentaires. La reconnaissance de caractères multi ou omnifontes fait partie de ces efforts. Très peu de travaux ont été conduits dans ce sens à cause des difficultés rencontrées : la coexistence de plusieurs fontes crée des ambiguïtés entre les caractères et introduit de nouvelles formes à reconnaître. Il faut donc un système robuste avec des capacités d'apprentissage relativement importantes et des procédures de reconnaissance très rapides. Par ailleurs, il s'est posé un autre problème relatif à la restitution des fontes. Faut-il un système omnifonte capable de reconnaître le caractère sans considération de sa fonte [27, 5], ou un système multifonte ne sachant identifier qu'un nombre restreint de fontes mais qui soit capable de les restituer?

Notre choix s'est porté sur la reconnaissance de fontes à partir d'un lot de fontes apprises [2, 3, 4]. Le système segmente le document en blocs homogènes (typographie uniforme), identifie la fonte dominante dans chaque bloc, puis procède à la reconnaissance des caractères dans cette fonte. Sachant que ce problème est difficile, on a échelonné le système en plusieurs niveaux lui permettant de résoudre les difficultés étape par étape.

D'abord, pour la reconnaissance de la fonte, on utilise une pré-classification des caractères sur la base de primitives très sûres et très distinctives des fontes.

Ensuite, pour la reconnaissance des caractères, on a mis à contribution plusieurs classifieurs : classifieurs markoviens du premier et du second ordre, et classifieurs de type métrique. Les modèles de Markov sont construits à partir d'un nombre important d'échantillons. Plusieurs tests ont été faits pour arrêter un modèle idéal de représentation du caractère (choix du nombre d'états, du nombre d'itérations, etc.). Les méthodes métriques sont d'une part faciles à mettre en œuvre et permettent d'autre part de donner un avis

complémentaire à celui des modèles de Markov. Le résultat final est obtenu en faisant voter toutes ces méthodes et en recueillant la décision majoritaire.

Enfin, afin de valider les résultats de la reconnaissance des caractères, on a utilisé plusieurs correcteurs lexicographiques. Le premier utilise un dictionnaire de la langue et réalise la correction par recherche de proximité. Les autres utilisent les chaînes de Markov d'ordre 1 et 2, ainsi que l'avis de l'OCR.

2. Architecture du système

La figure 1 montre l'architecture multi-niveaux du système réalisé. Au premier niveau, le texte est numérisé puis segmenté en blocs de textes à l'aide de l'algorithme RXYC [32]. Ensuite, chaque bloc est analysé, puis sa fonte dominante est déterminée. Chaque caractère, extrait du bloc, est pré-classé suivant quelques indications sur sa forme.

Au deuxième niveau, chaque caractère est traité par plusieurs classifieurs, puis le résultat de la reconnaissance est donné par vote à la majorité. Les temps de reconnaissance exigés par l'emploi d'un nombre élevé de classifieurs (six) est réduit grâce aux indications sur la classe et la fonte du caractère.

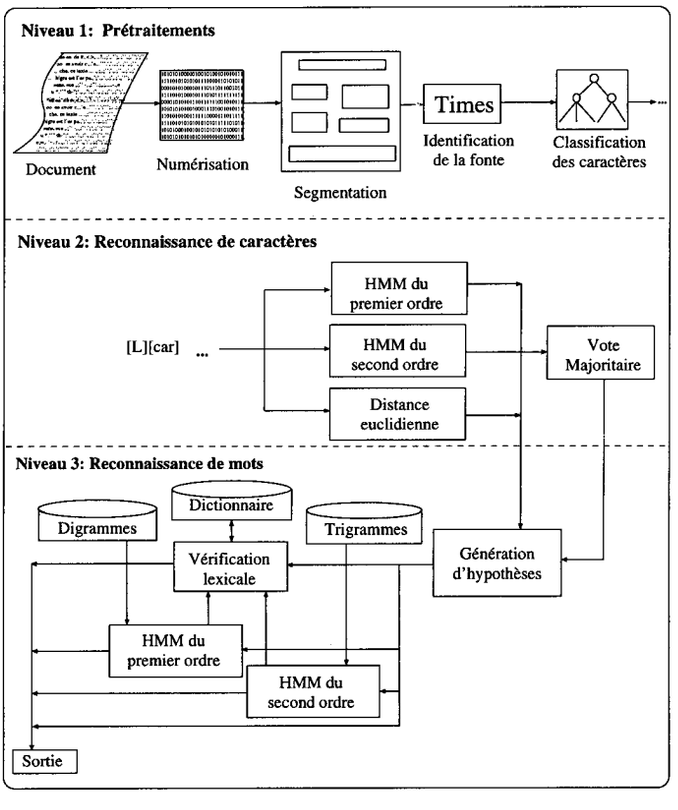


Figure 1. – Architecture multi-niveaux du système.

Le dernier niveau regroupe les caractères en mots, puis procède à leur validation à l'aide d'un lexique et de statistiques sur la langue employée.

3. Identification de la fonte

Nous limitons le champ d'identification de la fonte à chaque bloc, en faisant l'hypothèse qu'il est rare qu'un même bloc puisse être écrit dans plusieurs fontes. Avant d'exposer la méthode d'identification employée, nous allons rappeler quelques notions typographiques relatives aux fontes, utilisées en imprimerie.

3.1. NOTIONS TYPOGRAPHIQUES

Une fonte est un sous-ensemble d'une famille de symboles intervenant dans la composition d'un texte. Ses caractéristiques typographiques ont été normalisées dans l'imprimerie, tant au niveau du symbole qu'à celui de la chaîne (mot) dans chaque ligne du texte.

3.1.1. Au niveau du symbole

La fonte est caractérisée par des dimensions et un dessin. Les dimensions précisent la largeur ou la *chasse* et le *corps*. La largeur est variable d'un symbole à l'autre (ex. : i, l, m, W). Le *corps* désigne la hauteur du caractère comprenant le blanc de séparation horizontale avec la ligne au-dessus. Il varie en fonction de l'usage prévu pour le caractère : texte courant, titrage ou affiche. La *chasse* comprend en plus de la largeur, l'espace entre les caractères. La chasse dépend du dessin, du style et de la grosseur du caractère.

Le dessin représente la forme et l'épaisseur du caractère. A force de corps égale, certains caractères paraissent plus épais (**gras**) que d'autres. On trouvera alors, en ordre croissant de force, du maigre, souvent limité à un trait relativement fin, du demi-gras, du gras, du noir, etc. La *forme* sert à faire ressortir un mot dans le texte. On utilise l'*italique* qui est une forme penchée oblique, les petites capitales qui sont des lettres majuscules de la hauteur des lettres courtes bas de casse. Certaines familles de caractères contiennent des INITIALES qui sont des lettres capitales sans talus.

Plusieurs classifications ont été faites au début du siècle pour répartir les caractères d'imprimerie en familles. Elles sont fondées sur la régularité de l'épaisseur des traits, sur la particularité des empattements et des terminales. Certains caractères ont des empattements triangulaires, d'autres quadrangulaires, etc. La figure 2 donne la classification de Vox [13] en huit familles.

3.1.2. Au niveau de la chaîne

Une chaîne est une suite de symboles appartenant en général à une même fonte ou à des fontes «compatibles» (par rapport à la classification des fontes) sauf peut-être pour l'initiale (lettrine). L'alignement et l'espacement entre les caractères sont aussi normalisés.

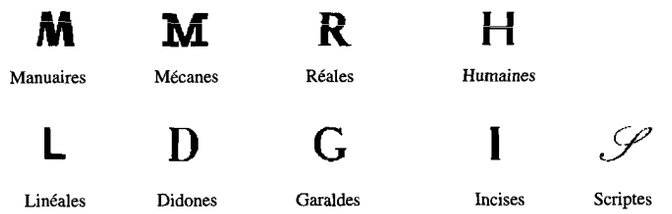


Figure 2. – La classification de VOX.

En effet, tous les caractères d'une même fonte et donc de même *corps* s'alignent en *pied* des lettres courtes : *a*, *e*, *i*, *o*, *u*, ... (voir fig. 3). Cette ligne est souvent appelée *ligne de base* et sert de référence aux calculs d'interligne et d'inclinaison. La *ligne médiane* est la ligne passant par le haut des lettres courtes et constitue avec la ligne de base, une zone de forte densité de points appelée *bande centrale*. De part et d'autre de cette bande, on trouve en haut la *ligne supérieure* joignant le haut des capitales (hauts de casse) et des hampes de bas de casse, et en bas, la *ligne inférieure* joignant les *jambages* des *bas de casse*. Les chiffres sont considérés, depuis le début du siècle, comme des *hauts de casse* et sont alignés en haut et en bas. La plupart des caractères sont placés sur au maximum trois lignes typographiques consécutives (ou deux bandes consécutives), sauf les caractères ()[]{} qui ne sont pas concernés ici.

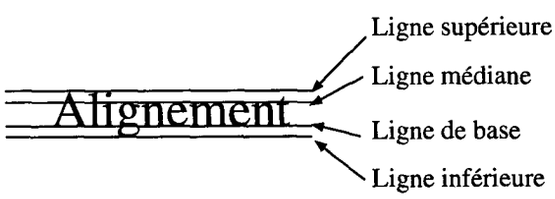


Figure 3. – Alignement des caractères dans un mot.

On désigne par *descendant* (ou *jambage*), la partie d'un caractère de *bas de casse* dépassant par en bas l'*œil* des lettres courtes, comme : g, j, p, q, y.

L'ascendant désigne la partie du caractère dépassant par dessus l'*œil* des lettres courtes (*hampe*), comme : t, h, l.

L'espace[1] correspond au blanc placé entre deux lettres consécutives dans une chaîne. Sa largeur n'est pas toujours la même de manière à assurer une grande lisibilité au texte. En effet, si la largeur est trop serrée, les caractères sont ligaturés, et si elle est trop grande, le mot risque de se fragmenter. Les typographes essaient de trouver là un équilibre entre le style des caractères et leur accolement dans le mot pour moduler la largeur du blanc. L'espace est plus large pour les lettres comportant des verticales comme le N, le H ou le U, et plus étroite pour les lettres à arrondis ou obliques comme le S, le O ou le V.

Par ailleurs, l'espace peut être réduite à cause des *empattements* (*serifs*) qui débordent à gauche et à droite des *jambages* (voir fig. 4). Les empattements dépendent du style du caractère comme cela a été montré dans la section précédente.

[1] Mot féminin en typographie.

Du haut de la colline de Saint-Laure
Du haut de la colline de Saint-Laure
Du haut de la colline de Saint-Laure
Du haut de la colline de Saint-Laure

Figure 4. – Exemples d'espaces différentes entre caractères.

3.2. PROBLÈMES LIÉS À L'IDENTIFICATION DES FONTES

L'identification de la fonte dans une image est sensible à plusieurs facteurs que nous allons d'abord rappeler :

- **Diversité des formes** : les caractères d'une même fonte peuvent changer de morphologie en changeant de style : a, f (*a, f*). Baird [5] proposa de traiter les italiques comme une fonte à part, au même titre que helvetica, Times, etc. D'ailleurs, c'est de cette manière que les firmes comme SONY, SUN, DEC, etc. classent leurs fontes.
- **Irrégularité des alignements** : l'alignement des caractères n'est pas parfaitement rectiligne. En effet, au niveau de la ligne de base, les imprimeurs essaient de rattraper le manque de formes causé par l'arrondi, par un dépassement vers le bas. Dans certaines fontes, les majuscules sont plus courtes que les ascendants afin d'autoriser dans certaines langues, comme le français, l'ajout d'accents, comme pour le È qui se voit abaisser sa hauteur.
- **Irrégularité des espaces** : l'espace entre les caractères peut varier en fonction de la fonte mais aussi en fonction du type de justification réalisée. Dans certaines fontes avec serif et une graisse importante, l'espace peut se réduire considérablement. L'accolement provoqué est un problème difficile en OCR. Par exemple, le cas de «ffi» est délicat et très peu de systèmes sont capables de le résoudre. L'accolement est réalisé par l'imprimeur de manière harmonieuse de sorte qu'il devient difficile d'extraire les caractères. En effet, en regardant le cas du «ffi», on se rend compte que sa largeur n'est pas plus grande que celle du «m» ou du «w» et la partie «fi» est tellement compacte qu'elle ressemble à un «h».
- **Irrégularité de la digitalisation** : deux caractères identiques, issus de la même fonte, n'ont jamais une représentation identique et cela pour plusieurs raisons. D'abord, des différences peuvent apparaître à cause du mécanisme d'impression et de la qualité du papier qui est loin d'être homogène sur toute la surface. A ces différences, s'ajoutent ensuite celle de la qualité de l'appareil de saisie et celle du processus de digitalisation. Ces différences ont parfois des conséquences très graves pour les caractères appartenant à des fontes très minces, provoquant des coupures ou des ligatures de caractères.

3.3. MÉTHODE CONCEPTUELLE

Rubinstein [39] mentionne quelques paramètres liés aux caractéristiques intrinsèques à la conception d'une fonte. Ces paramètres sont :

- la *couleur* qui mesure l'espace entre les caractères d'une même fonte, placés côte à côte suivant l'ordre alphabétique;

- le *poids* qui donne une mesure de la graisse et qui est estimé égal à la largeur des traits verticaux;
- le *contraste typographique* donné par le rapport du poids des traits verticaux sur celui des traits horizontaux.

La table 1 donne les valeurs de ces mesures pour les fontes helvetica et Times.

Tableau 1. – Mesures typographiques des fontes helvetica et Times.

Fontes	Couleur	Poids	Contraste
helvetica	163	0.16	1
Times	156	0.17	2

Ces paramètres sont difficiles à calculer une fois les caractères digitalisés. En effet, le calcul de la couleur, par exemple, est impossible à réaliser sur un texte réel, faute de pouvoir trouver la disposition demandée des caractères.

3.4. MÉTHODE ANALYTIQUE

Sans trop s'éloigner de ces caractéristiques, nous en avons cherché d'autres relevant de la typographie des fontes. Notre choix s'est porté sur des primitives typographiques issues directement des formes des caractères. En effet, compte tenu de ce qui a été dit, les caractéristiques d'une fonte sont «résumées» dans celles de certains de ses caractères.

Notre méthode consiste donc à ramener le problème de l'identification des fontes à celui de la comparaison de leurs lettres. L'idée est de sélectionner les lettres comparables morphologiquement, puis de mesurer leur différence d'une fonte à l'autre. La fonte qui présente l'écart le plus faible avec la fonte en cours sera retenue.

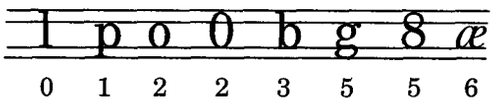
Nous allons d'abord décrire les primitives qui vont permettre de constituer ces classes de caractères dans les fontes, puis nous décrirons les méthodes d'apprentissage et de reconnaissance de fontes.

3.4.1. Extraction de primitives

Nous avons défini quatre types de primitives relatives aux cavités, aux transitions horizontales et verticales, et au *ratio*. Nous allons revenir, dans la suite, sur la définition de chacune d'entre elles.

Cavités

Presque un tiers de l'alphabet latin comprend des cavités dont le nombre et la position varient en fonction du caractère mais aussi de la fonte, comme par exemple **a**, *a*, **g**, g. Pour extraire les cavités, on utilise l'algorithme de May [31] qui consiste à suivre le contour extérieur dans le sens des aiguilles d'une montre, et à suivre le contour intérieur dans le sens contraire. Ces cavités sont caractérisées par rapport au contour extérieur du caractère. La figure 5 explicite les types de cavités issus de ce codage.

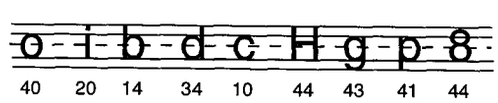


Code	Type	Exemples
0	absence de cavité	I T
1	1 cavité en haut	P q
2	1 cavité au centre	o O D
3	1 cavité en bas	b d 6
4	2 cavités, les deux côte à côte	∞
5	2 cavités, une en haut, l'autre en bas	g 8 B
6	2 cavités, l'une centrée à gauche, l'autre en haut à droite	œ
7	l'inverse de 6	
8	caractère trop bruité	

Figure 5. – Différents types de cavités.

Transitions horizontales

Chaque caractère est placé sur au maximum deux bandes typographiques consécutives. Afin de coder le passage par ces bandes et distinguer les caractères avec ascendant et descendant droit ou gauche, on fait traverser chaque bande par une sonde horizontale, puis on compte le nombre de transitions blanc-noir-blanc dans ces bandes. On obtient alors deux codes, un pour chaque bande. Si le caractère est centré, alors le deuxième code est nul. La figure 6 donne des exemples de codes pour la fonte helvetica et le tableau des codes.



Code	Type	Exemples	Code	Type	Exemples
0	absence de transition		4	2 transitions	o n
1	1 transition à gauche	c	5	3 transitions	m w
2	1 transition au centre	1	6	4 transitions	**w M**
3	1 transition à droite	d			

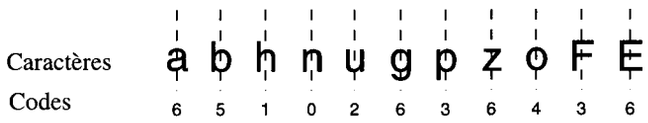
Figure 6. – Transitions horizontales.

Transitions verticales

Le caractère est plus stable dans le sens vertical; on utilise alors une seule sonde verticale passant par le centre de gravité du caractère donnant un seul code, comme cela est indiqué dans la figure 7.

Ratio

Certaines lettres ont un rapport hauteur/largeur très important (l, i), tandis que d'autres ont un rapport plutôt faible (m, w). Le ratio



Code	Type	Exemples
0	1 transition en haut	n
1	1 transition au milieu	h
2	1 transition en bas	u v
3	2 transitions, l'une en haut, l'autre au milieu	p F
4	2 transitions, l'une en haut, l'autre en bas	o
5	2 transitions, l'une au milieu, l'autre en bas	b d
6	3 transitions	a E
7	4 transitions	g

Figure 7. – Transitions verticales.

qui exprime ce rapport, est codé par une valeur comme indiqué dans le tableau suivant :

Code	Type	Exemples
0	formes très larges $\ll 1$	m
1	formes carrées	o n c
2	formes plus hautes que larges	k b d
3	formes très hautes	l i j

3.4.2. Apprentissage des fontes

Construction d'arbres de décision

La base d'apprentissage est composée des dix fontes les plus courantes du Macintosh. Chaque fonte comprend trois alphabets : les bas de casse (26), les hauts de casse (26) et les chiffres (10). Chaque symbole est donné par 240 échantillons, d'où une base d'apprentissage de 14 880 échantillons. Les trois alphabets sont appris séparément de manière à réduire l'espace de recherche lors de la reconnaissance. Chaque alphabet est décrit par un arbre de décision répartissant les caractères dans des classes typographiques relativement homogènes. Les tests sont déduits des primitives extraites précédemment. Pour les bas de casse, on répartit d'abord les caractères dans trois classes correspondant l'une aux ascendants, l'autre aux descendants et la troisième aux caractères centrés. Ensuite, on examine dans chaque classe le nombre et la position des cavités, etc. Les figures 8 et 9 donnent les arbres de décision correspondant respectivement aux fontes Times et Courier.

Nous avions indiqué précédemment que la nature des fontes était résumée dans quelques-uns de ses caractères. L'intérêt de ces arbres est donc de dégager les différents points de comparaison entre les fontes en distinguant par les tests les caractères identiques et ceux différents. Comme on peut le voir dans ces figures, les feuilles peuvent contenir plusieurs caractères différents et un

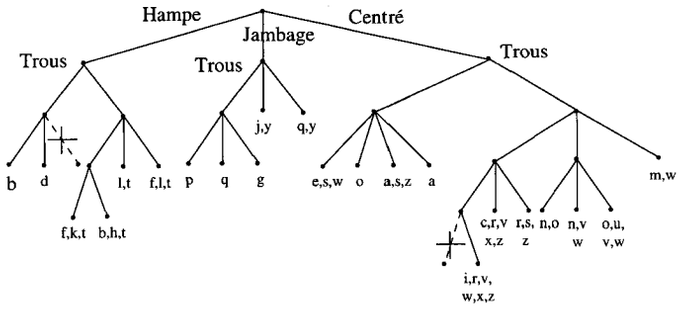


Figure 8. – Arbre de décision des bas de casse de la fonte Times.

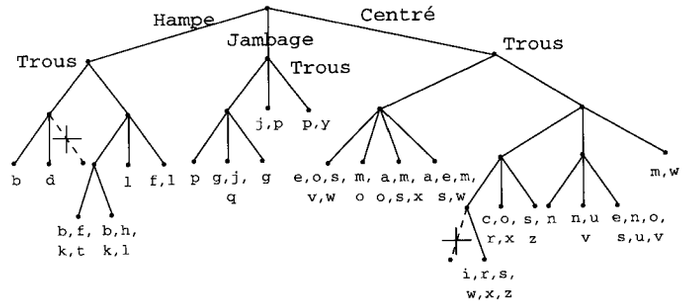


Figure 9. – Arbre de décision des bas de casse de la fonte Courier.

même caractère peut apparaître dans plusieurs feuilles. De plus, un même caractère peut être représenté par plusieurs échantillons dans la même feuille.

Recherche de représentants

Afin de réduire l'ensemble des échantillons d'un caractère dans une feuille à un seul représentant, on utilise l'algorithme de classification par nuées dynamiques : K-means [43]. Pour chaque feuille, on initialise les centres des classes avec le premier échantillon de chaque caractère. Ensuite, à chaque itération k, on distribue les suites de primitives $\{x\}$ dans ces classes (au nombre de K) selon la règle suivante :

$$x \in E_j(k) \text{ si } \|x - c_j(k)\| < \|x - c_i(k)\| \; \forall i = 1, 2, \ldots, k, \; i \neq j$$

où $E_j(k)$ est l'ensemble d'échantillons qui ont $c_j(k)$ comme centre à l'itération k. A partir de ces distributions d'échantillons dans les K classes, on calcule les nouveaux centres $c_j(k+1)$, pour $j = 1, 2, \ldots, K$, de telle sorte que la somme des distances (D_j) de tous les échantillons du nouveau centre soit minimale. On minimise donc la quantité :

$$D_j = \sum_{x \in E_j(k)} \|x - c_j(k+1)\|^2, \; j = 1, 2, \ldots, K$$

où $c_j(k+1)$ est simplement la moyenne de la classe $E_j(k)$:

$$c_j(k+1) = \frac{1}{N} \sum_{x \in E_j(k)} x, \quad j = 1, 2, \ldots, K,$$

où N_j est le nombre d'échantillons dans la classe $E_j(k)$. L'algorithme se termine quand la distance entre les centres construits

aux itérations k et $k+1$ est nulle pour tout j. Ce critère d'arrêt se résume par :

$$D = \|c_j(k+1) - c_j(k)\| = 0 \quad \forall j = 1, 2, \ldots, K$$

3.4.3. *Identification de la fonte*

L'identification de la fonte dans un paragraphe ne nécessite pas l'emploi de tous les caractères présents. Seuls quelques représentants de l'alphabet suffisent pour faire ressortir les caractéristiques typographiques de la fonte.

Recherche de représentants

Le texte est examiné caractère par caractère, puis les caractères différents par leur image sont repérés. La ressemblance (S) entre deux caractères A et B est estimée par calcul de corrélation entre leurs images, comme suit :

$$S = 1 - \frac{XOR(aligne(I_A, I_B))}{Max(I_A, I_B)}$$

où I_A et I_B sont les images de dimensions respectives $H_A \times L_A$ et $H_B \times L_B$, et «*aligne*» une fonction de réajustement des images des caractères.

L'opérateur XOR conduit à supprimer chaque point commun aux deux images. Donc, si les deux images sont identiques, alors leur XOR est nul et le coefficient de corrélation est maximal ($S = 1$). En réalité, à cause du bruit dans les images, on n'atteint jamais ce maximum, et il faut utiliser un seuil proche de 1 (voir fig. 10). Par contre, si les images sont complètement différentes, le XOR garde plus de points qu'il n'en existe dans chaque image et le rapport $\frac{XOR(I_A, I_B)}{\max(I_A, I_B)} > 1$. Avec cette quantité, on obtient une corrélation négative (voir fig. 11).

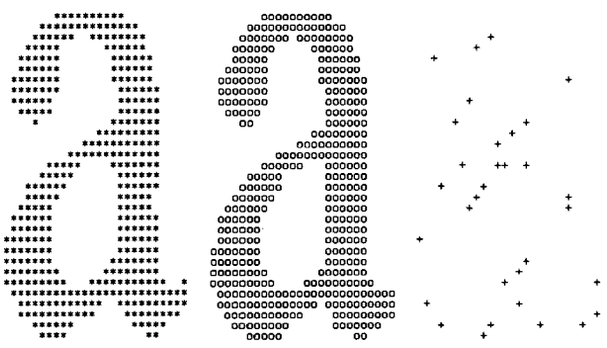


Figure 10. – Corrélation de deux «a».

L'application de cette fonction de corrélation est précédée de trois prétraitements.

1. Le premier conduit à s'assurer que les dimensions des deux images ne sont pas trop inégales, ce qui conduirait sinon à un calcul inutile. Dans ce cas, les dimensions doivent satisfaire les contraintes de taille suivante :

$$H_B = \alpha H_A, \qquad L_B = \beta L_A$$

2. Le second consiste à aligner les deux images en les cadrant au mieux (par la fonction $aligne$) avant de faire le calcul. «*aligne*» cherche d'abord à faire coïncider les centres de gravité des deux caractères, puis effectue des translations verticales et horizontales de manière à faire correspondre au mieux les profils.
3. Enfin, pour limiter la corrélation aux caractères les plus proches morphologiquement, on répartit les caractères dans des classes suivant leur emplacement par rapport aux lignes typographiques et suivant le nombre et la position de leurs cavités [1].

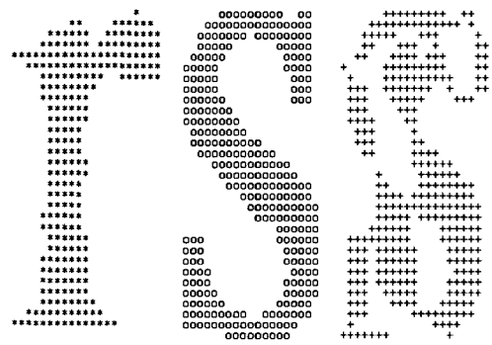


Figure 11. – Corrélation d'un «r» et d'un «s».

Recherche de la fonte dominante

Ne connaissant pas le mode des caractères utilisés (minuscule, majuscule ou chiffre) dans un paragraphe, leurs représentants sont introduits dans les trois arbres génériques. Ensuite, on compare l'arbre obtenu avec ceux des fontes apprises. La fonte retenue sera celle dont l'arbre est le plus proche de celui du paragraphe. La proximité des arbres est recherchée par calcul et cumul des distances entre les feuilles homologues dans les arbres à l'aide de la formule suivante :

$$CorrectFont = Arg \min_{1 \le j \le F} [\sum_{i=1}^{N} (\min_{\alpha} \|V_i - P_{\alpha j}\|)]$$

où : N est le nombre de caractères dans le paragraphe, F est le nombre de fontes pour lesquelles il existe un arbre, V_i est le vecteur paramètre pour le $i^{ième}$ caractère dans le paragraphe, et $P_{\alpha j}$ est le $\alpha^{ième}$ prototype de la fonte j. Il est évident que chaque V_i est comparé aux prototypes dont les α's sont dans des nœuds correspondants.

La figure 22.b montre les résultats obtenus pour le texte de la figure 22.a. Parmi les 712 caractères du paragraphe, seuls 41 sont considérés comme représentants, d'où un facteur de compression élevé de 94.24%. Le système propose les trois fontes les plus proches possibles avec dans l'ordre : Chicago (d=7),

Helvetica (d=15) et Athens (d=16). Le temps de reconnaissance est relativement réduit, de l'ordre de 0.2 seconde.

4. Reconnaissance de caractères

Il existe dans la littérature plusieurs méthodes de reconnaissance de caractères [29, 17, 40, 6]. Chacune a ses points forts et ses faiblesses, mais aucune n'est capable de fournir à elle seule une très bonne réponse pour chaque caractère. D'où l'idée de mettre à contribution plusieurs méthodes et de décider à la majorité. Pour cela, nous avons choisi des méthodes de type différent : *stochastique* et *métrique*, que nous allons décrire dans la suite.

4.1. MODÈLES DE MARKOV CACHÉS

4.1.1. Introduction

Un processus stochastique met en œuvre des modèles probabilistes spécifiques dans le but de gérer l'incertitude et le manque d'information qui entachent les formes à reconnaître. Dans notre cas (la reconnaissance de caractères), ce phénomène de manque d'information ou d'incertitude est assez courant; l'utilisation des processus stochastiques permet de rassembler au sein d'un même modèle les alternatives de représentation de chaque caractère, en faisant ressortir les primitives caractéristiques, et en amoindrissant l'effet des cas rares et du bruit. De plus, ces alternatives étant réparties de manière optimale et valuée, il est possible d'effectuer une reconnaissance rapide et d'accorder un score de confiance à la décision.

Les modèles de Markov cachés (Hidden Markov Models : HMM) sont des processus stochastiques particuliers. Leur utilisation en reconnaissance des formes n'est pas nouvelle. Leur principe de base a été publié par Baum et al. [10, 11, 9] dans les années 70. Ils ont d'abord été appliqués sur des signaux de parole par [7, 8, 36, 26, 30, 22, 25, 14, 21, 24], puis devant le succès obtenu, ils ont été utilisés sur les caractères manuscrits par Neuhoff [33], Farag [15] et Kundu [25]. A notre connaissance, c'est Kordi [23] qui a été le premier à les utiliser sur les caractères imprimés.

Comme on le voit, les HMM peuvent être utilisés sur n'importe quel type de signal. La seule contrainte imposée est d'avoir des primitives stables et de se munir d'une règle d'observation de primitives; mais ceci n'est-il pas la règle commune à tous les classifieurs de type structurel? Cela étant, il existe des critères d'optimisation qui rendent l'utilisation des modèles de Markov plus adéquate à un problème donné, comme le type du modèle, son ordre, son nombre d'états, etc. On peut trouver dans Rabiner [34, 35] quelques recommandations sur l'utilisation et l'implication de certains de ces modèles.

4.1.2. Définition générale

Un HMM représente une forme par deux suites de variables aléatoires; l'une observable (O_T) et l'autre cachée (Q_T).

Un HMM discret est donné par :

- N, le nombre d'états notés : $S = s_1, s_2, \ldots, s_N$. On désignera par q_t l'état à l'instant t.
- M le nombre distinct de symboles observables par état, noté $V = \{v_1, v_2, \ldots, v_M\}$. Un symbole observé à l'instant t sera désigné par o_t.
- $A = \{a_{ij}\}$, où a_{ij} est la probabilité de transition de l'état i à l'état j.

  $a_{ij} = P(q_{t+1} = s_j / q_t = s_i), 1 \leqslant i, j \leqslant N$

  La définition de A révèle deux propriétés des modèles étudiés. La première indique qu'il s'agit d'un modèle *stationnaire*, car pour figer ainsi la valeur des a_{ij}, il faut qu'elles vérifient la propriété suivante :

  $$a_{ij} = P(q_{t+1} = s_j / q_t = s_i) = P(q_{t+k+1} = s_j / q_{t+k} = s_i) \forall k$$

  La seconde indique qu'il s'agit d'un HMM du premier ordre puisque la probabilité d'observation à un état ne prend en compte que l'état courant et l'état précédent.
- $B = \{b_j(k)\}$, où $b_j(k)$ est la probabilité d'observer le symbole v_k, sachant que le modèle est dans l'état j.

  $b_j(k) = P(o_t = v_k / q_t = s_j), 1 \leqslant j \leqslant N, \ 1 \leqslant k \leqslant M$

  Les deux matrices A et B ne dépendent pas de t pour un HMM stationnaire.
- $\pi = \{\pi_i\}$, où π_i est la probabilité de commencer à l'état i.

  $\pi_i = P(q_1 = s_i), 1 \leqslant i \leqslant N$

Un HMM sera donc totalement déterminé par la connaissance de N, M et λ avec $\lambda = (\pi, A, B)$

4.1.3. Types de HMM

Il existe deux types de HMM, le modèle *ergodique* et le modèle *gauche-droite*. Le modèle *ergodique* est un modèle sans contrainte où toutes les transitions sont permises. Dans ce modèle, tout nœud peut être un nœud de départ et également un nœud d'arrivée (les π_i peuvent être non nuls). Le modèle *gauche-droite*, dit modèle de Barkis, est un modèle contenant des contraintes entre états (interdictions de certaines transitions). Ce type de modèle donne une structure particulière à la matrice A (matrice triangulaire supérieure inversible) : on part de l'état le plus à gauche vers l'état le plus à droite sans retour en arrière possible. Ce modèle présente deux variantes : un modèle sériel et un modèle à branches parallèles. Ces deux modèles imposent un seul état de départ (1) et un seul état d'arrivée (N), et une traversée séquentielle des états suivant les transitions permises ($a_{ij} = 0$ si $i > j$)

4.1.4. Utilisation des HMM

L'utilisation efficace des HMM sur des cas réels conduit à résoudre trois types de problèmes :

Evaluation : *soient une suite d'observations O et un modèle λ, comment évaluer la probabilité d'observation $P(O/\lambda)$?*

Une évaluation directe affecte à $P(O/\lambda)$ la somme sur tous les chemins d'états possibles Q des probabilités conjointes de O et de Q, soit :

$$P(O/\lambda) = \sum_Q P(O,Q/\lambda) = \sum_Q P(O/Q,\lambda)P(Q/\lambda)$$
$$= \sum_Q \pi_{q_1} b_{q_1}(O_1) a_{q_1} a_{q_2} b_{q_2}(O_2) \ldots a_{q_{T-1}q_T} b_{q_T}(O_T)$$

Cette formule directe nécessite $2TN^T$ opérations, ce qui est très lourd; d'où une autre variante, appelée *Forward-Backward* [10, 11]. Dans cette approche, on considère que l'observation peut se faire en deux temps : d'abord, émission du début de l'observation $O(1:t)$ en aboutissant à l'état q_i au temps t, puis émission de la fin de l'observation $O(t+1:T)$ sachant que l'on part de q_i au temps t. Dans ce cas, l'évaluation de l'observation est égale à :

$$P(O/\lambda) = \sum_{q_i} \alpha(t,q_i)\beta(t,q_i)$$

où α correspond à l'émission du début et β à l'émission de la fin. Le calcul de ces deux quantités se fait de manière inductive, conduisant à une réduction sensible de la complexité de l'algorithme d'évaluation (N^2T opérations).

Calcul du chemin optimal, ou estimation de la partie cachée, ou encore reconnaissance. *Soient la suite d'observations O et le modèle λ, comment trouver une suite d'états $Q = q_1 q_2 \ldots q_T$ qui soit optimale au sens d'un certain critère*?

Le critère le plus utilisé est celui de trouver la meilleure suite d'états maximisant $P(Q/O,\lambda)$. La solution pour ce critère est appelée suite d'états de Viterbi car elle est issue de l'algorithme de Viterbi [16], fondé sur la technique de programmation dynamique.

Estimation du modèle : *comment ajuster les paramètres du modèle pour maximiser $P(O/\lambda)$?*

Ceci relève de l'apprentissage qui pose un problème relatif à l'absence de critère d'optimisation globale et à l'absence de méthode directe. Les solutions utilisées ne présentent que des optimisations locales telles que les techniques du *gradient* et les procédures de ré-estimation. L'idée de base des procédures de ré-estimation est d'affiner le modèle petit à petit en démarrant à partir d'un ensemble initial de paramètres λ_0 , de calculer λ_1 à partir de λ_0 et de répéter ce processus jusqu'à un critère de fin. L'algorithme de Baum et Welch présente une procédure de ré-estimation.

4.1.5. Application à la reconnaissance de caractères

Les chaînes de primitives

Le choix de la représentation du caractère par une chaîne de primitives nous a conduits naturellement à utiliser un modèle de type *gauche-droite*. Les primitives utilisées sont des codes représentant l'allure du profil centroïdal du caractère, suivant un sens fixé le rapprochant ainsi de la représentation d'un signal continu. On fait évidemment l'hypothèse que ces primitives sont régies par les propriétés des modèles d'ordre 1.

La figure 12 donne un exemple de codage de ce profil pour la lettre «n». Pour déterminer le profil centroïdal, on subdivise le cadre du caractère en octants, puis on calcule dans chaque octant deux quantités m_1 et m_2. m_1 est la moyenne des distances du centre de gravité du caractère aux points extrêmes du contour extérieur dans l'octant. On calcule de la même manière m_2, mais en considérant les points du contour placés sur le cadre. Un code entre 0 et 3 est attribué selon la valeur du rapport entre m_1 et m_2. Ce rapport assure une invariance par rapport à la taille et à la densité d'impression.

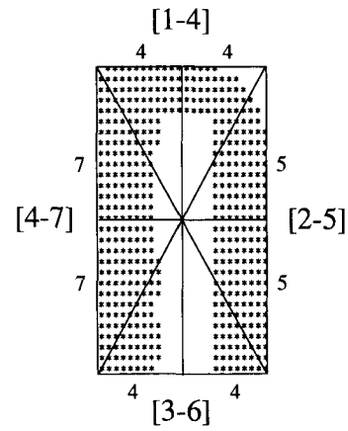


Figure 12. – Codage du profil centroïdal de la lettre «n» par la chaîne «44554477».

Avec ce type de codage, «l» aura le même code dans chaque octant (33333333), puisque le contour extérieur coïncide avec le cadre englobant. Afin d'avoir une fourchette de codes plus grande et prendre plus en compte la zone géographique où s'effectuent ces mesures, on a ajouté 1,2,3 et 4 à chaque code dans les directions N,E,S et O respectivement. On obtient donc les fourchettes suivantes dans les quatre zones : [1-4], [2-5], [3-6] et [4-7]. Ceci transforme le code du «l» en 44556677. On a choisi l'ordre d'extraction N, E, S, O de manière arbitraire car des analyses de données, notamment en composantes principales, ne nous ont révélé aucun ordre pertinent.

Apprentissage

Etant donné la pré-classification réalisée au premier niveau du système, les modèles des caractères ont été répartis par fonte et par classe (les classes correspondent aux feuilles des arbres de décision). La base d'apprentissage comporte 240 échantillons par caractère pour les fontes principales comme helvetica et Times et 60 échantillons pour les fontes italiques. Ces échantillons sont de corps 10 à 18 points et numérisés à 300 points par pouce. A l'exception des fontes italiques qui sont en mono-style, les échantillons comportent les styles romain et gras car, dans ces deux styles, la morphologie reste inchangée.

N'ayant fait aucune hypothèse sur la distribution des primitives dans les états du modèle, nous avons choisi d'attribuer le même poids à chaque primitive dans chaque état, ce qui donne $b_j(k) =$

Tableau 2. –Distribution initiale des symboles.

$$B = \begin{pmatrix} 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 \\ 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 \\ 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 \\ 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 \\ 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 & 0.111 \end{pmatrix}$$

$\frac{1}{M} \quad \forall j,k$ satisfaisant à la contrainte $\sum_{k=1}^{M} b_j(k) = 1$. Pour l'exemple précédent, la suite d'observations O_T sera égale à $O_8 = 44554477$.

Tous les modèles sont identiques au départ. La figure 13 et le tableau 2 montrent les valeurs initiales du modèle choisi à 5 états ($N = 5$) avant l'apprentissage. Un état fictif (0) a été ajouté pour faciliter la programmation; il sera sans effet sur les modèles comme le montrent les valeurs de A et B. En effet, on peut remarquer que les premières lignes sont nulles (correspondant à l'état 0 avant et après l'apprentissage). Ceci a été choisi ainsi car l'une des caractéristiques de l'algorithme d'apprentissage est que les valeurs qui sont nulles au départ le resteront à la fin.

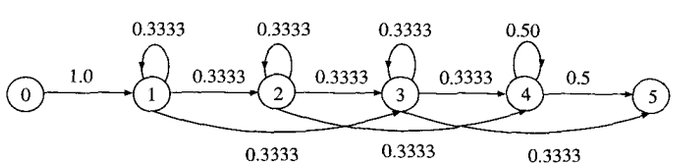


Figure 13. – Transitions initiales.

La figure 14 donne les modèles de transition obtenus après 15 itérations pour les formes ***a*** et **a**. Bien qu'issues d'un même modèle initial, les deux formes de la même lettre ont des modèles différents.

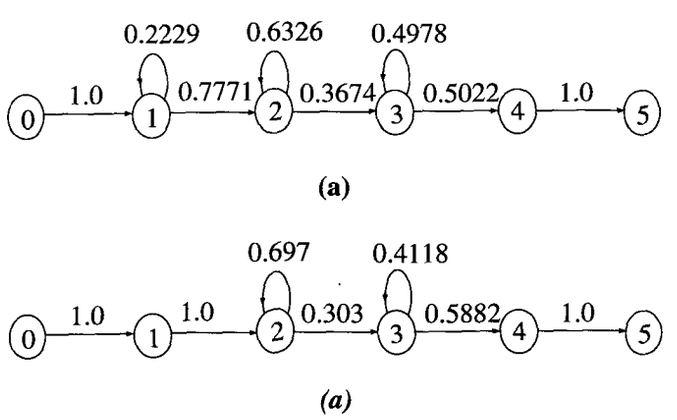


Figure 14. – L'état des modèles de la lettre a au cours de l'apprentissage.

Reconnaissance

Connaissant la classe à laquelle appartient le caractère, celui-ci est comparé aux modèles λ_k, $k = 1, \ldots, L$ de sa classe. Le modèle retenu sera celui qui fournira la meilleure probabilité correspondant à l'évaluation de sa suite de primitives, c'est-à-dire :

$$car = arg \max_{1 \leqslant k \leqslant L} [P(O/\lambda_k)]$$

Grâce aux informations dont on dispose, l'algorithme de *Viterbi* [16, 36] utilisé pour la reconnaissance se trouve allégé. En effet, l'étape de cheminement arrière n'est plus nécessaire. Il n'y a aucun phénomène physique lié aux états et par conséquent il n'y en a pas non plus aux suites d'états empruntés pour arriver à la probabilité maximale. La découverte du modèle donnant la plus forte probabilité $P(O/\lambda)$ est suffisante pour identifier le caractère. De plus, sachant que les valeurs de A et B sont connues à l'avance, on peut optimiser l'algorithme de reconnaissance en les transformant en valeurs logarithmiques. Ainsi, on n'a plus que des additions à faire.

L'algorithme de reconnaissance devient :

- *Initialisation*
  $\delta_1(i) = \log(\pi_i) + \log(b_i(O_1)) \quad 1 \leq i \leq N,$
- *Induction*
  $\delta_t(j) = \max_{1 \leqslant i \leqslant N} [\delta_{t-1}(i) + \log(a_{ij})] + \log(b_j(O_t)) \quad 1 \leq j \leq N, \quad 2 \leqslant t \leqslant T$
- *Terminaison*
  $P^* = \max_{1 \leqslant i \leqslant N} [\delta_T(i)]$

Résultats et discussion

Influence du nombre d'états

Le tableau 3 donne les résultats de reconnaissance pour les différentes fontes utilisées, répartis suivant le nombre d'états dans le modèle. On remarque que le nombre d'états influe sur les résultats d'un modèle, ce qui est en tout cas logique, car celui-ci intervient beaucoup dans le calcul des paramètres du modèle. Globalement, ce tableau montre que l'évolution du score de reconnaissance suit celle du nombre d'états. Cependant, si l'on examine ces scores fonte par fonte, on s'aperçoit que cette évolution est irrégulière, voire dégressive par moment. Ceci correspond aux constatations de Rabiner [37] qui avait fait remarquer que l'évolution du taux de reconnaissance ne suit pas toujours l'augmentation du nombre d'états et qu'il n'y a pas de moyen théorique pour

déterminer *a priori* le nombre d'états optimal car ces derniers n'ont pas de relation avec des phénomènes physiques.

Tableau 3. – Taux de reconnaissance (SV).

Fonte	Nombre d'états dans chaque modèle				
	4	5	6	7	8
Chicago	98.01	97.94	98.50	97.57	97.77
Geneva	98.35	99.12	99.52	99.83	99.83
New York	97.03	97.90	97.83	98.62	98.77
Princeton	99.11	99.50	99.61	99.62	99.62
Toronto	98.86	99.10	99.49	99.53	99.54
Monaco	94.76	97.22	97.48	97.65	97.57
Athens	96.30	96.01	96.14	96.47	96.85
Courier	93.49	95.13	96.12	96.44	96.92
Helvetica	97.58	98.18	99.59	99.57	99.50
Times	95.84	97.01	97.41	97.94	98.27
Moyenne	96.93	97.71	98.17	98.32	98.46

Cependant, quelques critères peuvent être donnés pour aider à trouver le nombre d'états. En effet, on peut limiter le nombre d'états par le haut en le fixant inférieur au nombre de symboles ($N \leqslant M$), car sinon certains états ne seront pas utilisés et seront considérés comme redondants ou superflus. En reconnaissance de la parole, de tels états sont utilisés pour simuler le silence. Ces états ne servent à rien dans notre cas, puisque l'on modélise des formes fermées. On peut aussi limiter le nombre d'états par le bas, en évitant de le réduire à trop peu d'états, ce qui risquerait de concentrer les observations dans les mêmes états. Les expériences que l'on a réalisées ont montré qu'il ne faut pas descendre au-dessous de 4 états.

Influence de l'initialisation

L'algorithme d'apprentissage garantit l'obtention de valeurs optimales de A et B de telle sorte que $P(O/\lambda)$ soit un point critique. Expérimentalement, ce point critique correspond en fait à un maximum local, et ainsi différentes valeurs initiales de A et B peuvent conduire à des valeurs différentes de $P(O/\lambda)$. Comme le montre la figure 15, λ correspond au modèle de départ, $\overline{\lambda}$ correspond au modèle réestimé et λ^* désigne le modèle optimal recherché. Il s'agit donc de trouver des valeurs de A et B idéales conduisant à une valeur de P globalement maximum (et donc constituer le meilleur modèle possible). Malheureusement, ce problème ne connaît pas de solution simple et directe. En effet, Rabiner [36] a constaté par expérience que des valeurs initiales aléatoires ou uniformes des matrices A et B donnent de bons résultats, c'est-à-dire que $a_{ij} = \dfrac{1}{N} + \varepsilon_1$ et $b_j(k) = \dfrac{1}{M} + \varepsilon_2$, à condition bien sûr de respecter :

$$\sum_{j=1}^{N} a_{ij} = 1 \; i = 1, 2, \ldots, N \text{ et } \sum_{k=1}^{M} b_j(k) = 1 \; j = 1, 2, \ldots, N$$

Dans ce système, on a choisi $\varepsilon_2 = 0$ pour la matrice B (c'est-à-dire équiprobabilité des primitives), et pour A, $\varepsilon_1 = 0$, mais $a_{ij} = \dfrac{1}{L}$ où L est le nombre de transitions non nulles autorisées par le modèle à partir de l'état i. Ceci implique que les valeurs ne sont pas identiques suivant l'état où on se trouve. La contrainte $\sum a_{ij} = 1$ est toujours respectée à partir de n'importe quel état i (voir le modèle de la figure 15).

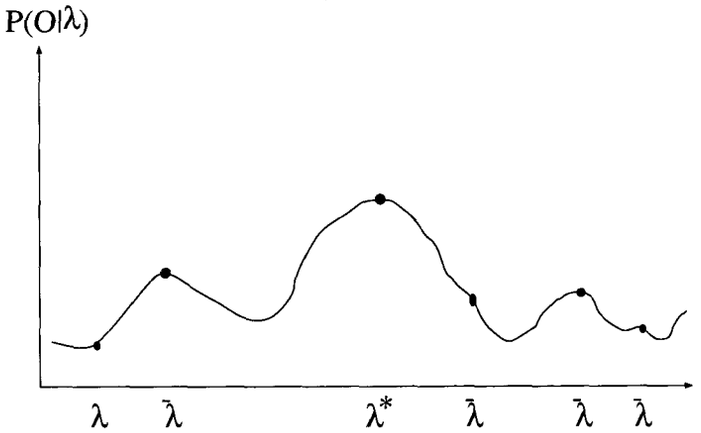


Figure 15. – Points critiques.

Influence du nombre d'itérations

Comme l'algorithme d'apprentissage ne garantit pas un maximum global, et comme il n'y a aucun moyen théorique de trouver le pic de $P(O/\lambda)$ globalement maximum, on a choisi de multiplier l'apprentissage des échantillons en l'itérant plusieurs fois. Le résultat obtenu (voir fig. 16) montre que l'influence de ce nombre d'itérations est variable en fonction du nombre d'états. Pour des modèles de 7 à 8 états, la variation est très minime. Par contre, pour des modèles de 4 à 6 états, de fortes variations apparaissent jusqu'à la quinzième itération, après laquelle le processus semble se stabiliser. On peut donc imaginer d'utiliser une fonction de lissage qui a pour effet de stabiliser les modèles.

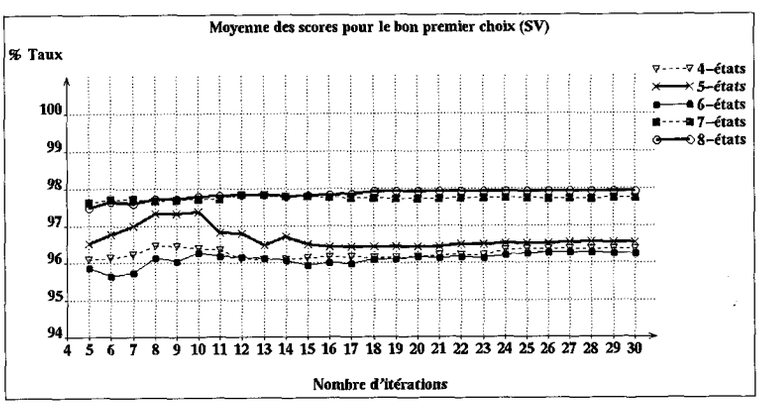


Figure 16. – Taux de reconnaissance en fonction du nombre d'itérations.

On constate en effet que le nombre d'itérations pour les modèles optimaux diffère suivant la fonte et le nombre d'états que comporte le modèle. Le tableau 4 donne le nombre d'itérations (entre parenthèses) conduisant à $P(O/\lambda)$ optimal pour chaque fonte et chaque nombre d'états. Ces nombres ne suivent pas une règle particulière; ils sont purement expérimentaux.

Influence du filtrage

Afin de compenser le manque d'optimalité de l'algorithme d'apprentissage, on a apporté deux solutions. La première consiste à orienter le choix du modèle à partir d'une connaissance sur la classe, c'est ce qui a été fait. La seconde consiste à tenir compte du nombre de caractères des classes d'apprentissage dans le calcul de la réponse de Viterbi ($\delta_T(N)$). Cette dernière est pondérée par la quantité $pd[L][C]$, définie comme étant la probabilité d'appartenance du caractère C à la classe L (famille de caractères jugés ressemblants lors de la pré-classification. Le tableau 5 montre ces résultats après 10 itérations. La figure 17 confirme ces améliorations et montre que l'augmentation du taux correspond ici aussi à une augmentation du nombre d'itérations. La courbe des modèles à quatre états laisse supposer qu'à un certain moment, ce type de modèle peut atteindre la performance d'un modèle à 8 états.

Tableau 4. – Nombre d'itérations optimal pour un HMM d'ordre 1.

Fonte	Nombre d'itérations nécessaire pour λ^*				
	4	5	6	7	8
Chicago	98.04(24)	98.00(20)	98.51(8)	98.05(5)	97.84(7)
Geneva	98.36(8)	99.54(23)	99.83(12)	99.84(6)	99.83(5)
New York	97.03(10)	97.90(10)	98.05(12)	98.82(14)	98.86(7)
Princeton	99.51(12)	99.61(23)	99.62(6)	99.62(10)	99.62(5)
Toronto	99.02(8)	99.13(22)	99.51(14)	99.73(5)	99.54(7)
Monaco	94.96(9)	97.33(30)	97.67(27)	97.99(6)	97.86(7)
Athens	96.30(10)	96.27(15)	96.25(22)	96.47(10)	96.86(9)
Courier	93.68(25)	95.94(23)	96.87(18)	96.73(29)	96.92(10)
Helvetica	97.67(5)	98.56(27)	99.59(10)	99.61(7)	99.54(8)
Times	96.16(14)	97.43(28)	97.78(12)	97.96(9)	98.28(12)
Moyenne	97.07	97.97	98.37	98.48	98.51

4.2. MODÈLES DE MARKOV DU SECOND ORDRE

Il est naturel de penser que plus on augmente l'ordre des HMM, plus on doit avoir de l'information et plus ces derniers seront performants [25, 24, 28]. Il est vrai que pour la reconnaissance de mots, la connaissance de plusieurs caractères à l'avance peut être avantageuse pour la reconnaissance du caractère en cours. Pour les primitives dans un caractère, ceci n'était qu'une hypothèse que l'on voulait vérifier. Mais *a priori* rien ne permet de l'appuyer car les primitives ne sont pas liées entre elles et ne suivent aucune distribution statistique compatible avec un modèle du deuxième ordre.

Tableau 5. – Taux de reconnaissance (SVP).

Fonte	Nombre d'états dans chaque modèle				
	4	5	6	7	8
Chicago	98.15	97.97	98.49	97.57	97.77
Geneva	98.04	99.12	99.64	99.96	99.96
New York	98.10	98.17	98.70	99.31	99.24
Princeton	99.60	99.62	99.69	99.62	99.65
Toronto	98.93	99.48	99.52	99.52	99.54
Monaco	95.54	97.35	97.53	97.83	97.98
Athens	95.79	95.90	96.43	96.69	97.02
Courier	93.54	96.19	96.67	96.66	97.18
Helvetica	97.90	98.37	99.60	99.60	99.57
Times	97.53	97.84	97.95	98.78	98.74
Moyenne	97.31	98.00	98.42	98.55	98.67

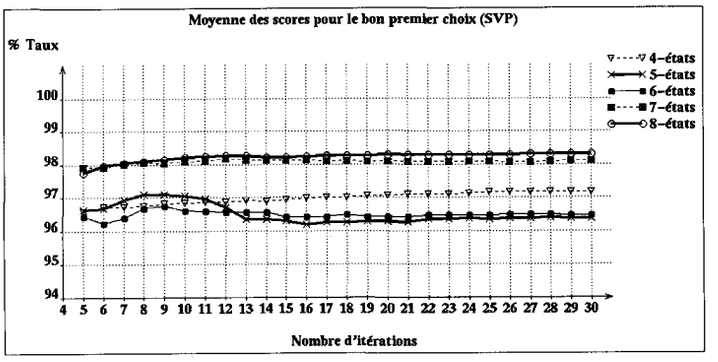


Figure 17. – Taux de reconnaissance en fonction du nombre d'itérations.

4.2.1. Apprentissage

Nous avons utilisé une version étendue de l'algorithme *Forward-Backward* de Baum, formalisée par Kriouille [24]. La variable $\alpha_t(i)$ du premier ordre devient $\alpha_t(i,j)$. Elle est définie comme la probabilité d'avoir la suite d'observations partielle jusqu'au temps t et d'être à l'état s_j à l'instant t en étant dans l'état s_i à l'instant $t-1$:

$$\alpha_t(i,j) = P[O_1 O_2 \ldots O_t, q_{t-1} = s_i, q_t = s_j/\lambda)$$

Par conséquent, nous avons à l'instant T

$$\alpha_T(i,j) = P[O_1 O_2 \ldots O_t, q_{T-1} = s_i, q_T = s_j/\lambda)$$

On calcule α_t de la même manière inductive :

1. $\alpha_2(i,j) = \pi_i b_i(o_1) a_{ijk} b_j(o_2),\ 1 \leqslant i,j \leqslant N$
2. Induction pour : $2 \leqslant t \leqslant T-1$

$$\alpha_{t+1}(j,k) = [\sum_{i=1}^{N} \alpha_t(i,j) a_{ijk}] b_k(o_{t+1}),\ 1 \leqslant j,k \leqslant N$$

3. Terminaison

$$P(O/\lambda) = \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_T(i,j)$$

Comme pour les modèles du premier ordre, dès qu'un choix est fait, il faut réestimer les paramètres du modèle. La réestimation de a_{ijk} se déduit directement de celui de a_{ij}. La réestimation des paramètres tient compte des doubles transitions. Par exemple, le paramètre

$$\bar{a}_{ijk} = \frac{\textit{nombre de doubles transitions de l'état } s_i \textit{ à l'état } s_j \textit{ et de } s_j \textit{ à } s_k}{\textit{nombre de transitions simultanées provenant des états } s_i \textit{ et } s_j}$$

$$= \frac{\sum_{t=1}^{T-2} \eta_{t+1}(i,j,k)}{\sum_{t=1}^{t-2} \xi_t(i,j)}$$

où $\eta_{t+1}(i,j,k) = \dfrac{\alpha_{t+1}(i,j)a_{ijk}b_k(o_{t+1})\beta_{t+2}(j,k)}{P(O|\lambda)}$

4.2.2. Reconnaissance

On utilise l'extension de l'algorithme de Viterbi comme mentionné dans He [18]. Comme pour le premier ordre, on simplifie l'algorithme initial en ne retenant pas de chemin.

1. Initialisation
   $\delta_1(i) = \log(\pi_i) + \log(b_i(O_1)), \quad 1 \leqslant i \leqslant N$
   $\delta_2(i,j) = \delta_1(i) + \log(a_{ij}) + \log(b_j(O_2)), \quad 1 \leqslant i,j \leqslant N$
2. Récursion
   $\delta_t(j,k) = \max_{1\leqslant i\leqslant N}[\delta_{t-1}(i,j) + log[a_{ijk}]] + log[b_k(O_t)], \quad 3 \leqslant t \leqslant T$
   $1 \leqslant j,k \leqslant N$
3. Fin de procédure
   $P^* = \max_{1\leqslant i,j\leqslant N}[\delta_T(i,j)]$

4.2.3. Résultats et discussions

Comme pour le premier ordre, on s'est livré à l'étude de l'influence de l'apprentissage sur le taux de reconnaissance. On a pour cela réalisé les mêmes tests en variant le nombre d'états et le nombre d'itérations. Les tableaux 6 et 7 montrent les résultats obtenus respectivement sans pondération et avec pondération.

Ces résultats ne permettent pas de constater des améliorations substantielles des taux de reconnaissance justifiant l'emploi des modèles du deuxième ordre. On peut remarquer que pour des modèles où le nombre d'états est faible (4 à 5), les résultats sont meilleurs que ceux correspondants dans les modèles du premier ordre. Mais ces mêmes résultats peuvent être obtenus par les modèles du premier ordre à plus fort nombre d'états

Tableau 6. – Nombre d'itérations optimal pour un HMM d'ordre 2.

	Nombre d'itérations nécessaire pour λ^*				
Fonte	4	5	6	7	8
Chicago	98.16(7)	98.12(5)	97.57(5)	98.18(15)	98.23(9)
Geneva	99.74(11)	99.82(8)	99.75(5)	99.83(14)	99.83(10)
New York	98.03(8)	98.13(6)	98.37(14)	98.91(11)	98.75(20)
Princeton	99.58(9)	99.62(15)	99.62(5)	99.62(7)	99.65(21)
Toronto	99.31(12)	99.50(9)	99.48(10)	99.52(7)	99.53(25)
Monaco	95.85(11)	97.20(11)	97.59(13)	97.69(15)	97.96(12)
Athens	95.81(17)	97.10(5)	96.34(17)	96.36(12)	96.44(17)
Courier	94.52(12)	95.79(14)	96.44(11)	96.66(18)	96.62(27)
Helvetica	98.90(7)	99.43(6)	99.13(6)	99.45(11)	99.50(28)
Times	96.88(21)	97.86(17)	97.84(21)	98.18(22)	98.17(21)
Moyenne	97.68	98.26	98.21	98.44	98.47

Tableau 7. – Nombre d'itérations optimal pour un HMM d'ordre 2 pondéré.

	Nombre d'itérations nécessaire pour λ^*				
Fonte	4	5	6	7	8
Chicago	98.16(7)	98.22(5)	97.67(5)	98.18(15)	98.23(9)
Geneva	99.65(11)	99.95(8)	99.83(7)	99.96(14)	99.95(10)
New York	98.44(22)	98.58(6)	98.96(14)	99.15(14)	99.13(14)
Princeton	99.61(12)	99.65(15)	99.64(5)	99.64(6)	99.65(9)
Toronto	99.35(15)	99.50(9)	99.49(8)	99.52(9)	99.53(25)
Monaco	96.98(20)	97.68(12)	97.93(27)	98.03(16)	98.23(12)
Athens	95.84(22)	96.93(6)	96.72(16)	96.30(12)	96.83(11)
Courier	95.15(13)	96.17(7)	97.00(12)	96.92(26)	97.12(25)
Helvetica	99.11(9)	99.63(8)	99.54(5)	99.54(21)	99.62(18)
Times	97.99(25)	98.42(17)	98.57(17)	98.73(18)	98.91(25)
Moyenne	98.03	98.47	98.53	98.60	98.72

(7 à 8). En conclusion, on peut dire que malgré l'amélioration apportée par le filtrage et compte tenu des calculs supplémentaires nécessaires aux modèles du second ordre, ces derniers sont moins performants que ceux du premier ordre. La cause de cette mauvaise performance proviendrait de l'insuffisance de l'apprentissage. En effet, ce type de modèle comporte un tiers de plus d'états que son homologue du premier ordre ($O(N^3T)$) contre ($O(N^2T)$). La donnée (liste de primitives de chaque échantillon) est donc distribuée sur N fois plus d'états que pour le premier ordre. L'importance de ce nombre d'états fera que la distribution des échantillons ne sera pas uniforme et des chaînes seront plus pénalisées que d'autres créant une baisse de probabilité (dilution). La théorie indique que pour retrouver un ordre de grandeur de probabilité égal à celui du premier ordre et équivalent

Tableau 8. – Résultats comparatifs des différentes méthodes.

Fonte	Résultats des différentes méthodes						
	HMM1		HMM2		DE		MAJ
	SV	SVP	SVE	SVEP	DE	DEP	MAJ
Chicago	98.51(6)	98.49(6)	98.23(8)	98.23(8)	96.38	97.54	99.18(8)
Geneva	99.84(7)	99.97(7)	99.83(8)	99.96(7)	98.99	99.25	99.83(7)
New York	98.86(8)	99.33(8)	98.91(7)	99.15(7)	97.67	98.72	98.62(8)
Princeton	99.62(8)	99.70(6)	99.65(8)	99.65(8)	99.20	99.52	99.62(8)
Toronto	99.73(7)	99.54(8)	99.53(8)	99.53(8)	96.10	99.72	99.52(8)
Monaco	97.99(7)	98.14(7)	97.96(8)	98.23(8)	93.62	97.50	98.76(8)
Athens	96.86(8)	97.02(8)	97.10(5)	96.93(5)	91.55	96.11	98.67(8)
Courier	96.92(8)	97.18(8)	96.66(7)	97.12(8)	91.73	95.91	98.00(8)
Helvetica	99.61(7)	99.67(7)	99.50(8)	99.63(5)	97.68	98.31	99.50(7)
Times	98.28(8)	98.87(8)	98.18(7)	98.91(8)	97.81	98.35	98.80(8)
Moyenne	98.62	98.79	98.55	98.73	96.07	98.09	99.05

au nombre d'états, il faut $R^{3/2}$ échantillons où R est le nombre d'échantillons utilisés pour les modèles du premier ordre.

4.3. CLASSIFIEURS DE TYPE PLUS PROCHE VOISIN

Au lieu de comparer le caractère entré aux modèles de Markov des prototypes de la classe dans laquelle il se trouve dans l'arbre de décision, on a pensé utiliser tout simplement l'idée du plus proche voisin ou k_NN (k nearest neighbor) [43, 12]. L'idée est de chercher parmi les prototypes des caractères représentés celui qui est le plus proche et d'affecter sa classe au caractère entré.

La similarité entre deux formes x et y est donnée par :

$$S_{xy} = \sum_{i=1}^{N}(x_i - y_i)^2, (x_i, y_i, i = 1, \ldots, N)$$

où x_i est la $i^{ème}$ primitive pour la forme x et N, le nombre de primitives. Nous pouvons transformer cette formule en une mesure de probabilité de la forme suivante :

$$P(x = p) = (1 - \frac{\sqrt{S_{xp}}}{\sqrt{S_{po}}})$$

où $\sqrt{S_{xp}}$ est la distance euclidienne entre le caractère x et le prototype p, et $\sqrt{S_{po}}$ est la distance entre p et une origine fixe o, dans l'espace euclidien. Par conséquent,

$$S_{xp} = \sum_{i=1}^{N}(x_i - p_i)^2 \text{ et } S_{po} = \sum_{i=1}^{N}(p_i)^2$$

Si les caractères x et p sont identiques, $\sum_{i=1}^{N}(x_i - p_i)^2 = 0$ et nous obtenons P(x=p)=1. Pour être considéré comme une mesure de probabilité, toute valeur inférieure à zéro sera considérée comme nulle.

Nous avons bâti le cinquième votant autour de cette mesure. Les résultats dans le tableau 8, sous le label de, montrent des taux de reconnaissance allant de 91.55% pour la fonte Athens à 99.20% pour la fonte Princeton.

Le défaut de cette méthode est qu'elle ne tient pas compte de la sûreté des primitives. Pour cela, nous avons développé un sixième votant autour de la formule suivante :

$$P(x = p) = \frac{\sum_{\forall x_i = p_i = \sigma_i, i=1}^{N} poids_p(\sigma_i)}{N}$$

où $poids_p(\sigma_i)$ est la probabilité que la $i^{ème}$ primitive pour le prototype p, ait la valeur σ. Autrement dit, c'est la proportion d'échantillons de caractère représenté par p qui ont σ comme valeur de la $i^{ème}$ primitive. On peut remarquer que le numérateur est égal à N si $x = p$. Pour tout autre cas, $0 \leqslant P(x = p) < 1$.

Comme le montre le tableau 8 (colonnes de et dep), cette méthode donne, pour la fonte Toronto, de meilleurs résulats que les autres. Les taux vont de 95.91% pour la fonte Courier à 99.72% pour la fonte Toronto.

4.4. RECONNAISSANCE PAR VOTE MAJORITAIRE

Le vote majoritaire consiste à faire voter de manière indépendante plusieurs classifieurs et à retenir la réponse majoritaire, correspondant normalement à la meilleure solution. Ceci est issu du fait

que les classifieurs pris individuellement sont incapables d'assurer une bonne reconnaissance pour tous les caractères. Suivant la méthode qui les anime, ils sont très performants pour certains et moins pour d'autres. Par exemple, un classifieur peut facilement faire la différence entre un "S" et un "5" et être en même temps incapable de distinguer entre un "2" et un "Z". L'idée donc est de combler ces déficiences en demandant l'avis de plusieurs classifieurs. Le vote majoritaire peut conduire, malgré les erreurs individuelles, à la bonne réponse. Il va de soi que cette méthode n'aboutira pas si le caractère a une forme pathologique ou s'il n'y a pas de consensus qui se dégage. On pense seulement qu'une telle méthode fera moins d'erreurs globalement que les méthodes prises individuellement.

Le tableau 8 montre les résultats des six votants utilisés et la réponse majoritaire sélectionnée (colonne maj) pour chaque fonte. Les six votants sont : deux votants de type hmm du premier ordre (méthode de Viterbi (SV) et sa version pondérée (SVP)), deux votants d'ordre 2 (Viterbi étendue (SVE) et sa version pondérée (SVEP)), et deux votants de type plus proche voisin (distance euclidienne (DE) et sa version pondérée (DEP)). Ces résultats sont obtenus après 30 itérations. Les valeurs entre parenthèses correspondent aux nombres d'états. On voit que globalement, on obtient de bons scores, mais la figure 18 montre que la vitesse se réduit considérablement et qu'un tel système n'est raisonnable que dans le cas de caractères dégradés, comme l'a déjà prouvé Ho [19].

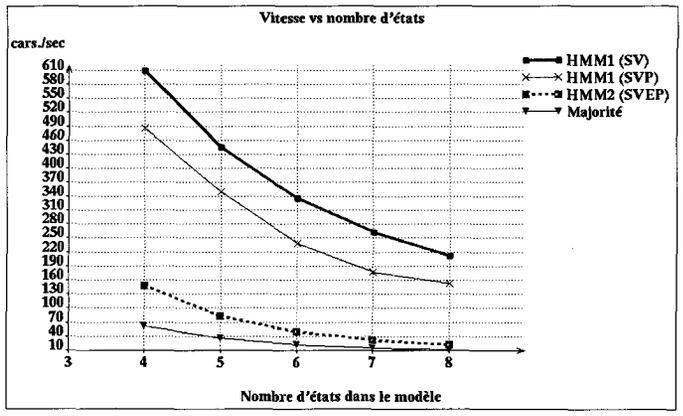


Figure 18. – Vitesse en fonction du nombre d'états.

5. Correction linguistique

La dernière étape du système consiste à regrouper les caractères en mots et à vérifier leur cohérence linguistique en fonction des données dont on dispose. Cette cohérence linguistique a été faite à l'aide de deux méthodes :

- vérification lexicale par comparaison par rapport à un dictionnaire de la langue employée;
- distribution de probabilités à l'aide de méthodes stochastiques.

5.1. VÉRIFICATION LEXICALE

Elle consiste à répartir les mots du lexique dans des tables de hash codes et à gérer, lors de la comparaison des mots, le conflit d'accès par une méthode de comparaison de chaînes.

Les mots sont d'abord classés par longueur, puis regroupés en fonction de la position de leurs caractères par rapport à la ligne de base et à la ligne médiane. Les caractères ayant des ascendants sont codés par 1, ceux ayant des descendants sont codés par 3, et les autres sont codés par 2. Une succession de codes identiques est remplacée par un code unique. Ainsi, on préserve par ce type de codage, l'aspect topologique du mot. On ne retiendra parmi ces codes que les cinq premiers qui sont en général suffisants pour limiter les collisions lors des comparaisons. Cette chaîne est complétée par la longueur initiale du mot, codée sur deux chiffres, ramenant le nombre de codes à 7.

Exemples :

mot	codes	suppression des codes identiques	sélection des 5 premiers	ajout de la longueur
exemple	2222312	2312	23120	2312007
algorithmes	2132221122	213212	21321	2132110

La recherche de mots par mise en correspondance entre le mot entré et les mots du dictionnaire est réalisée à l'aide de la méthode de Ratcliff et Obershelp [38].

5.2. DISTRIBUTION DE PROBABILITéS

L'idée de base de ces méthodes est de considérer que les lettres dans un mot sont reliées entre elles par un processus stochastique. L'observation d'une lettre à un endroit de la chaîne est en relation avec celle faite sur les lettres qui la précèdent dans la chaîne. L'identité z_n de la $n^{ième}$ lettre doit maximiser $P(x_n = z_n/x_0, \ldots, x_{n-1})$. Ceci peut permettre alors de corriger certains caractères, comme par exemple : "qvestion" en "question", sachant qu'après un "q" dans la langue française, il ne peut y avoir qu'un "u".

L'algorithme de Viterbi a été appliqué initialement sur les mots par Neuhoff [33]. Le modèle de Markov associé décrit les successions possibles de lettres dans les mots de la langue. Les arcs expriment la probabilité de succession de couples de lettres, et les nœuds les probabilités d'observation des lettres aux positions qu'elles occupent dans le mot. La reconnaissance d'un mot consiste à trouver un chemin qui maximise la mise en correspondance entre les lettres reconnues par l'OCR et celles observées le long de ce chemin.

En effet, soit la chaîne $x = x_1, x_2, \ldots, x_n$ donnée par l'OCR. La probabilité qu'un autre mot $z = z_1, z_2, \ldots, z_n$ ait donné lieu à x est égale (d'après la formule de Bayes) à :

$$log(P(z|x)) = log(P(x|z)) + log(P(z)) - log(P(x))$$

Comme $P(x)$ est indépendant de z, maximiser $log[P(z|x)]$ revient à maximiser :

$\log(P(x|z)) + log(P(z))$, soit : $\log(P(z|x)) \equiv log(P(x|z)) + log(P(z))$

En supposant que les mots suivent un processus markovien, on doit maximiser pour les modèles d'ordre 1 :

$$G_1(x,z) = \sum_{i=1}^{n} log(P(x_i|z_i)) + \sum_{i=1}^{n+1} log(P(z_i|z_{i-1}))$$

et pour l'ordre 2 :

$$G_2(x,z) = \sum_{i=1}^{n} log(P(x_i|z_i)) + log(P(z_1|z_0)) + \sum_{i=2}^{n+1} log(P(z_i|z_{i-2},z_{i-1}))$$

Ces deux équations supposent qu'il n'y a pas de dépendance entre $x_1,\ldots,x_{n-1},x_n$, ce qui est vrai dans le cas des caractères imprimés où ceux-ci sont reconnus un à un. Le terme $P(x_i|z_i)$ est la probabilité de confusion entres les deux lettres x_i et z_i, c'est-à-dire, la probabilité que l'OCR a proposé la lettre x_i quand la vraie identité de la forme est la lettre z_i.

$G_r(x,z)$ a une forme de treillis, et se prête donc à la programmation dynamique. L'algorithme de Viterbi permet de calculer G_r de manière efficace et optimale. Cet algorithme trouve le mot z sur toutes les chaînes imaginables de longueur n, qui maximise $G_r(x,z)$, sachant que ce mot ne se trouve pas forcément dans un dictionnaire.

5.2.1. Modèles de Markov d'ordre 1

Apprentissage

Les capacités de correction des HMM d'ordre 1 sont limitées comme l'avait indiqué Jouvet [21]. Cependant, ces modèles sont intéressants si les paramètres sont bien choisis (apprentissage complet) [42, 25, 41]. On a utilisé des modèles de type *gauche-droite* à branches parallèles à partir d'un dictionnaire de 190 000 mots. Les matrices A et B sont déduites à partir de ce lexique. La matrice A contient les fréquences de di-grammes des couples de lettres, tandis que la matrice B contient les probabilités d'apparition des lettres dans leur position dans les mots.

On a montré dans la section précédente que le nombre d'états influe sur le taux de reconnaissance et donc sur les paramètres de A et B. Ce phénomène se retrouve ici, car la longueur différente des mots appris influe sur la distribution des *n-grammes*. Pour résoudre ce problème, nous avons choisi de contraindre les calculs par la longueur des mots. En effet, les éléments de B seront donnés par :

$$b_t(k) = \frac{\gamma_t(k)}{\eta_T}$$

où $b_t(k)$ est la probabilité que la lettre ou le symbole v_k apparaisse dans l'état ou la position t (dans le mot), $\gamma_t(k)$ est le nombre de fois que le caractère représenté par v_k est observé dans cet état, et η_T est le nombre de symboles observés à l'état t additionné le long de tous les mots de longueur T, c'est-à-dire :

$$\eta_T \equiv \sum_{\forall V_i} \gamma_t(i)$$

De manière semblable, le calcul de a_{ij} est contraint par T, c'est-à-dire :

$$a_{Tij} = \frac{\xi(i,j)}{\sum_{\forall V_k} \xi(i,k)}$$

où $\xi(i,j)$ est le nombre de transitions de la lettre V_i vers la lettre V_j et $\sum_{\forall V_k} \xi(i,k)$ est le nombre de transitions en provenance de la lettre V_i.

Reconnaissance

Nous avons utilisé une version modifiée de l'algorithme de Viterbi. Supposons que la matrice $p(T,N)$ contienne les trois réponses de l'OCR ($N = 3$), T étant la longueur de la chaîne. Il faut lire ses éléments $p_t(i), i = 1,2,3$ comme étant la $i^{ème}$ alternative de la lettre pour la position t dans le mot. Pour optimiser le calcul, on profite de la structure de treillis de $p(T,N)$ en ne calculant $\delta_t(j)$ qu'avec des $p_t(j)$ non vides ($p_t(j) \neq \phi$)

1. $\delta_1(i) = \log(\pi_i) + \log(B(1,o(1))), \quad 1 \leqslant i \leqslant N$

   où π_i est équivalent à $A(0,p_1(i))$, c'est-à-dire, la probabilité que l'alternative i soit le premier caractère du mot; o(t) est la $t^{ème}$ observation, qui est également équivalente à $p_t(1)$ (en indexant la matrice $p(T,N)$).

   De manière générale, $\delta_t(i)$ est la probabilité maximum qui explique les premiers t caractères qui terminent dans l'état i où $t \leqslant T$ et i est l'une des lettres a à z.

2. Récursion

$$\delta_t(j) = \max_{1\leqslant i\leqslant N}[\delta_{t-1}(i) + \log(A(p_{t-1}(i),p_t(j)))] + log(B(t,o(t))), 2 \leqslant t \leqslant T \quad 1 \leqslant j \leqslant N$$

$$\psi_t(j) = arg \max_{1\leqslant i\leqslant N}[\delta_{t-1}(i) + \log(A(p_{t-1}(i),p_t(j)))]$$

3. Fin de procédure

$$P^* = \max_{1\leqslant i\leqslant N}[\delta_T(i)]$$

$$q_T^* = arg \max_{1\leqslant i\leqslant N}[\delta_T(i)]$$

4. Cheminement arrière

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1,\ldots,1$$

   La suite d'indices récupérés à la sortie de cette étape servent d'indices dans la matrice p pour retrouver la suite la plus probable.

La figure 19 montre un exemple de mot à sept lettres pour les sorties suivantes de l'OCR : **(m,w), a, (l,t,f), a, i, (s,z), e.** Le chemin suivi par l'algorithme de reconnaissance est en gras. Il ne correspond pas au mot donné par l'OCR qui est "malaise', mais au mot "mataise', donné avec une probabilité 1.498e-15. La raison à cette anomalie est que les probabilités de choisir "al", "at" et "af" après avoir choisi "ma" sont telles que : $pr(af)[0.0098] < pr(al)[0.0651] < pr(at)[0.104]$, d'où l'emprunt du chemin "at". On résoud ce problème en incorporant

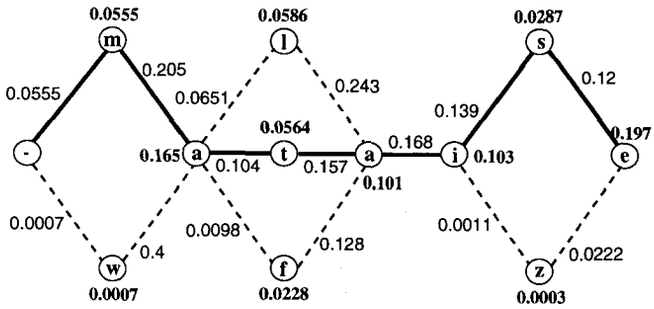


Figure 19. – Réseau de Viterbi modifié pour une chaîne de 7 lettres.

les taux de confiance que l'OCR a obtenu lors de ses choix des possibilités.

La figure 20 montre que le système a cette fois-ci emprunté le bon chemin malgré des probabilités finales nettement plus faibles qu'avant (5.582e-30 contre 1.498e-15), dues aux multiplications des valeurs faibles avec des valeurs encore plus faibles. Les scores de reconnaissance de l'OCR sont montrées en italique. On peut voir que le score de l'OCR pour le choix du troisième caractère l a un ordre de grandeur beaucoup plus important que les deux alternatives t et f (0.0461 contre 2.616e-16 et 1.193e-23 respectivement).

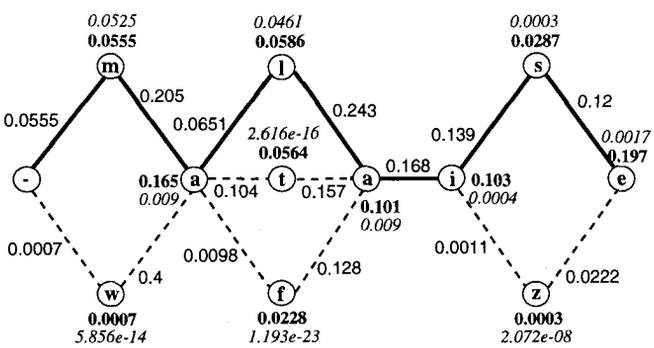


Figure 20. – Réseau de Viterbi pondéré pour une chaîne de 7 lettres.

5.2.2. Modèles de Markov d'ordre 2

Apprentissage

Comme les transitions s'étendent sur des triplets de nœuds, on recalcule A et on conserve B. Pour les triples transitions de lettres, on pose :

$$a_{Tijk} = \frac{\xi(i,j,k)}{\sum_{\forall V_k} \xi(i,j,k)} \quad (1)$$

où $\xi(i,j,k)$ est le nombre de transitions de la lettre V_i à la position $t-2$ vers l'état V_j à l'instant $t-1$ et finalement vers la lettre V_k à l'instant t, $\sum_{\forall V_k} \xi(i,j,k)$ étant le nombre total de transitions en provenance du caractère V_i.

Reconnaissance

Comme pour le premier ordre, la matrice $p(T,N)$ contient les triplets donnés par l'OCR pour chaque position $t = 1,\ldots,T$. La même contrainte d'optimisation $p_t(j) \neq \phi$ reste valable.

L'algorithme de parcours du réseau du second ordre découle directement du premier par extrapolation.

1. $\delta_1(i) = \log(\pi_i) + log[B(1,o_1)], \quad 1 \leqslant i \leqslant N$

   où π_i reste équivalent à $A(0,p_1(i))$, c'est-à-dire, la probabilité que l'alternative i soit le premier caractère du mot;

   o_t est la $t^{ème}$ observation, également équivalente à $p_t(1)$ (en indexant la matrice $p(T,N)$); le marqueur "-" de la figure 21 occupe la position 0, c'est-à-dire $A(0,p_1(i)) \equiv A(p_0(-),p_1(i))$.

   $$\delta_2(i,j) = \delta_1(i) + log(A(p_1(i),p_2(j))) + \log(B(2,o(2))), 1 \leqslant i,j \leqslant N$$

   Pour les mêmes raisons que pour le premier ordre, ψ_1 et ψ_2 ne sont pas initialisés.

2. Récursion

   $$\delta_t(j,k) = \max_{1\leqslant i\leqslant N}[\delta_{t-1}(i,j) + log(A(p_{t-2}(i),p_{t-1}(j),p_t(k)))] + log(B(t,o(t))),$$
   $$\psi_t(j,k) = arg_i \max_{1\leqslant i\leqslant N}[\delta_{t-1}(i,j) + log(A(p_{t-2}(i),p_{t-1}(j),p_t(k)))], \; 3 \leqslant t \leqslant T, \; 1 \leqslant j,k \leqslant N$$

3. Fin de procédure

   $$P^* = \max_{1\leqslant i,j\leqslant N}[\delta_T(i)]$$
   $$q^*_{T-1} = arg_i \max_{1\leqslant i,j\leqslant N}[\delta_T(i,j)]$$
   $$q^*_T = arg_j \max_{1\leqslant i,j\leqslant N}[\delta_T(i,j)]$$

4. Cheminement arrière

   $$q^*_t = \psi_{t+2}(q^*_{t+1}, q^*_{t+2}), \quad t = T-2,\ldots,1$$

   Comme pour le premier ordre, q^*_t sert comme indice dans la matrice p pour retrouver la suite la plus probable.

La figure 21 montre le réseau de Viterbi d'ordre 2 avec le chemin en gras emprunté par l'algorithme de reconnaissance. Le bon mot "malaise' ayant la plus grande probabilité $3.957e^{-15}$, a été choisi. Pour ne pas écarter définitivement certains chemins interdits pour cause de non validité selon les fréquences de tri-grammes, une probabilité très faible leur est affectée. Donc, les transitions nulles telle que "aiz" ont en fait des valeurs non nulles. Le but de cet exercice est d'empêcher le système d'écarter certaines branches de manière à prendre en compte des suites "valides" sorties par l'OCR mais qui ne sont pas autorisées par la langue (comme par exemple, les sigles et les mots étrangers).

Tableau 9. – Résultats de reconnaissance des mots.

Mot	Performances des trois méthodes ci-dessus					
	HMM1		HMM1 + conf. de l'OCR		HMM2	
	Probabilité	Choix	Probabilité	Choix	Probabilité	Choix
This	$1.393e^{-11}$	fhis	$9.648e^{-25}$	This	$6.555e^{-12}$	This
paper	$5.628e^{-11}$	paper	$5.113e^{-23}$	paper	$1.607e^{-11}$	paper
exposes	$4.607e^{-17}$	exposes	$1.054e^{-40}$	exposes	$6.252e^{-16}$	exposes
stochastic	$2.480e^{-24}$	stochastis	$3.072e^{-52}$	stochastic	$2.826e^{-24}$	stochastic
modeling	$9.005e^{-18}$	modeling	$6.465e^{-33}$	modeling	$2.500e^{-16}$	modeling
recognition	$1.255e^{-25}$	recognition	$3.385e^{-46}$	recognition	$2.096e^{-21}$	recognition
hidden	$5.360e^{-15}$	hidden	$1.376e^{-26}$	hidden	$2.438e^{-14}$	hidden
script	$2.813e^{-16}$	scripl	$1.138e^{-35}$	script	$7.551e^{-16}$	script
numerous	$4.070e^{-21}$	numerous	$2.099e^{-43}$	numerous	$4.425e^{-19}$	numerous
applications	$2.590e^{-22}$	apptications	$1.225e^{-50}$	applications	$8.898e^{-20}$	applizations
such	$2.058e^{-11}$	suck	$2.733e^{-24}$	such	$4.487e^{-11}$	suck
character	$2.040e^{-19}$	character	$4.317e^{-44}$	character	$9.141e^{-20}$	character
based	$2.603e^{-11}$	bazed	$6.449e^{-26}$	based	$1.191e^{-11}$	based
the	$2.635e^{-08}$	the	$2.029e^{-18}$	the	$8.343e^{-09}$	the

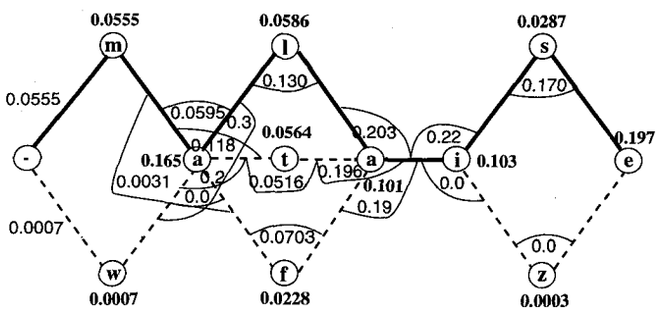


Figure 21. – Réseau de Viterbi d'ordre 2 pour une chaîne de 7 lettres.

Le tableau 9 montre quelques résultats correspondant aux décisions prises par chacune des trois méthodes exposées. On a pu remarquer que quand la longueur du mot n'était pas suffisante pour obtenir des tri-grammes, les di-grammes renforcés donnent de meilleurs résultats. Enfin, compte tenu de toutes les expérimentations que l'on a réalisées, on peut dire qu'en l'absence d'un correcteur lexical, le HMM du premier ordre ayant pris en compte des scores de confiance de l'OCR est la meilleure méthode.

5.3. RÉSULTATS

Le système a été testé sur 6 pages de texte, comportant chacune de 1 504 à 2 498 caractères et écrite dans cinq fontes différentes. La figure 22 montre un exemple de reconnaissance d'un bloc d'une de ces pages : on trouve le texte initial (a), les résultats de reconnaissance de la fonte plus des statistiques (b), les résultats de reconnaissance par les HMM1 (c), les corrections par les HMM1 (d) et les corrections par les HMM2 (e). Les taux de réussite, dépendant fortement de l'étape précédente (OCR), vont de 90% (pour les résultats bruts) à 99% pour la correction avec di-grammes plus le dictionnaire. Ces taux élevés s'expliquent par le fait que les erreurs de l'OCR se limitent parfois à quelques mots où les caractères se touchent ou sont coupés.

6. Conclusion

Nous avons présenté dans cet article un système complet de reconnaissance de textes imprimés multifontes opérant à plusieurs niveaux. Au premier niveau, le système détermine la fonte dominante dans chaque bloc de texte à partir d'un lot de fontes connues. La méthode utilisée est fondée sur d'abord une pré-classification des caractères à l'aide de quelques primitives, puis ensuite sur le calcul de la distance entre les familles homologues de caractères. Le processus de comparaison est allégé par recherche de représentants à la fois dans le texte et parmi les familles. Les tests réalisés ont montré que la recherche de la fonte se fait toujours avec une grande précision.

Au deuxième niveau, le système met à contribution plusieurs classifieurs de types différents pour réaliser la reconnaissance des caractères. Quatre d'entre eux sont de type Markov du premier et du second ordre. Les efforts ont été menés pour donner un sens à la distribution des primitives dans les modèles et pour étudier l'influence des paramètres sur les résultats. Les résultats obtenus sont tout à fait corrects. Des améliorations ont été obtenues en renforçant les algorithmes de base à l'aide de fonctions de pondération. Le vote majoritaire n'a pas montré beaucoup d'utilité dans le cas de textes bien écrits et son emploi peut mieux se justifier dans le cas de textes dégradés.

Au troisième niveau, le système utilise plusieurs types de correcteurs pour valider la reconnaissance de caractères. Les processus de Markov du premier et du second ordre ont été également utilisés sous forme de "réseaux de Viterbi". Ces réseaux utilisent des tables de di-grammes et de tri-grammes pour circonscrire les chemins inutiles et renforcer les successions statistiquement fréquentes entre les caractères. Les résultats de ces réseaux ont été renforcés par les scores de confiance de l'OCR.

BIBLIOGRAPHIE

[1] J. ANIGBOGU, A. BELAÏD. Reconnaissance de caractères multifontes par vote à la majorité. In A. Belaïd, editor, *Traitement de l'écriture et des documents, Actes du colloque CNED'92*, p. 91-99, Nancy, juillet 1992.

[2] J.C. ANIGBOGU. Un système stochastique de reconnaissance de caractères imprimés multipolices. Technical report, Université de Nancy I, sept. 1989.

This paper exposes a stochastic modeling approach to multifont printed text recognition using Hidden Markov Models. The system uses stable features like profiles, global character shape, presence and position of holes etc. to capture as much as possible the variations in character shapes in different fonts. We first determine the identity of the the predominant font, paragraph by paragraph so as to limit the number of models to deal with. This is achieved by projecting some features into hyperspace and using Euclidean distance measures to determine proximity to font prototypes constructed during a learning phase. Given that HMM presents a global view of the forms, deterministic decision-trees are used to channel the system towards appropriate models. We also use such heuristics as presence of ascenders and descenders to construct these trees.

(a)

```
Characters Read = 712  Distinct Shapes = 41

Compression = 94.24%

Seg., F. Extraction & Compression Time : 50.433 sec(s)

1. --> Chicago with distance 7
2. --> Helvetica with distance 15
3. --> Athens with distance 16

most unlikely --> Monaco with distance 22

Font Rec. + Wt. loading Time :  0.233 sec(s)

Loaded 1st order 6-state models in  3.850 sec(s)

Chars: 41  Recognition Time :  0.133 sec(s)

Word Formation Time (712 chars):  0.050 sec(s)
```

(b)

This paper exposes a stochastic modeling approach to multifont printed text recognition using Xidden Markov Models. The system uses stable features like profiles, global character shape, presence and position of holes etc. to capture as much as possible the variations in character shapes in different fonts. We first determine the identity of the the predominant font, paragraph by paragraph so as to limit the number of models to deal with. This is achieved by projecting some features into hyperspace and using Euclidean distance measures to determine proximity to font prototypes constructed during a learning phase. Given that XMM presents a global view of the forms, deterministic decision-trees are used to channel the system towards appropriate models. We also use such heuristics as presence of ascenders and descenders to construct these trees.

(c)

This paper exposes a stochastic modeling approach to multifont printed text recognition using Hidden Markov Models. The system uses stable features like profiles, global character shape, presence and position of holes etc. to capture as much as possible the variations in character shapes in different fonts. We first determine the identity of the the predominant font, paragraph by paragraph so as to limit the number of models to deal with. This is achieved by projecting some features into hyperspace and using Euclidean distance measures to determine proximity to font prototypes constructed during a learning phase. Given that HMW presents a global view of the forms, deterministic decision-trees are used to channel the system towards appropriate models. We also use such heuristics as presence of ascenders and descenders to construct these trees.

(d)

This paper exposes a stochastic modeling approach to multifont printed text recognition using Xidden Markov Models. The system uses stable features like profiles, global character shape, presence and position of holes etc. to capture as much as possible the variations in character shapes in different fonts. We first determine the identity of the the predominant font, paragraph by paragraph so as to limit the number of models to deal with. This is achieved by projecting some features into hyperspace and using Euclidean distance measures to determine proximity to font prototypes constructed during a learning phase. Given that XMM presents a global view of the forms, deterministic decision-trees are used to channel the system towards appropriate models. We also use such heuristics as presence of ascenders and descenders to construct these trees.

(e)

Figure 22. – Résultats entiers de la reconnaissance d'un paragraphe de texte, d'après [Anigbogu92].

[3] J.C. ANIGBOGU, A. BELAïD. Application of Hidden Markov Models to Multifont Text Recognition. In *First International Conference on Document Analysis and Recognition*, vol. 2, p. 785-793, 30 Sept.-2 Oct 1991.

[4] J.C. ANIGBOGU. Reconnaissance de textes imprimés multifontes à l'aide des modèles stochastiques et métriques. Doctorat de l'Université de Nancy I, septembre 1992.

[5] H.S. BAIRD, R. FOSSEY. A 100-font Classifier. In *First International Conference on Document Analysis and Recognition*, vol. 1, p. 332-340. 30 Sept.-2 Oct 1991.

[6] H.S. BAIRD, S. KAHAN, T. PAVLIDIS. Components of a Omnifont Page Reader. In *Proceedings of 8th International Conference on Pattern Recognition*, p. 344-348, Paris, 1986.

[7] J.K. BAKER. *Stochastic Modeling for Automatic Speech Understanding*, p. 541-542. D.R. Reddy Ed., Speech Recognition, Academic Press, New York, 1974.

[8] J.K. BAKER. The Dragon System – An Overview. *IEEE-ASSP*, ASSP-23(1) : 24-29, 1975.

[9] L.E. BAUM. *An Inequality and Associated Maximization Technique for Probabilistic Functions of Markov Process*, Inequalities, 3, p. 1-8, 1972.

[10] L.E. BAUM, J.A. EGON. *An Inequality with Applications to Statistical Estimation for Probabilistic Functions of a Markov Process and to a Model for Ecology*, Bull, Ams. vol. 73, p. 360-363, 1967.

[11] L.E. BAUM, T. PETRIE. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *Annals of Mathematics and Statistics*, 37 : 1554-1563, 1968.

[12] P. DEVIJVER, J. KITTLER. *Pattern Recognition : A Statistical Approach.* Prentice Hall, New York, 1982.

[13] J. DREYFUS, F. RICHAUDEAU. *La chose imprimée.* La bibliothèque du CEPL, 1977.

[14] Y. EPHRAIM, A. DEMBO, L.R. RABINER. A minimum Discimination Information Approach for Hidden Markov Modeling. In *IEEE-ICASSP*, vol. 1, p. 25-28. IEEE, 1987.

[15] R.F.H. FARAG. Word-Level Recognition of Cursive Script. *IEEE Transactions on Computer*, (28(2), February, 1979.

[16] G.D. FORNEY. The Viterbi Algorithm. in *Proceedings of IEEE*, vol. 61, No. 3, 1973.

[17] G. GAILLAT, M. BERTHOD. Panorama des techniques d'extraction de traits caractéristiques en lecture optique des caractères. In *Actes 2ème Congrès AFCET-IRIA de Reconnaissance des Formes et Intelligence Artificielle*, p. 353-368, Toulouse, 1979.

[18] Y. HE. Extended Viterbi Algorithm for Second Order Hidden Markov Process. *9th ICPR*, vol. 2, p. 718-720. IEEE, 1988.

[19] T.K. HO. *A Theory of Multiple Classifier Systems and Its Application to Visual Work Recognition.* PhD thesis, State University of New York at Buffalo, May 1992.

[20] J.J. HULL, SN SRIHARI, R. CHOUDHARI. An Integrated Algorithm for Text Recognition : Comparison with a Cascaded Algorithm. *IEEE-PAMI*, PAMI-5(4) : 384-395, July 1983.

[21] D. JOUVET. *Reconnaissance de mots connectés indépendamment du locuteur par des méthodes statistiques.* Thèse de Doctorat, Sup. Télécom, juin, 1988.

[22] B.H. JUANG, L.R. RABINER. Mixture Autoregressive Hidden Markov Models for Speech Signals. *IEEE-ASSP*, 33 : 1404-1413, 1985.

[23] K. KORDI, C. XYDEAS, M. HOLT. Printed Character Recognition Using Markov Models. *Onzième Colloque Gretsi*, p. 507-509, 1-5 Juin 1987.

[24] A. KRIOUILE. *La reconnaissance automatique de la parole et les modèles markoviens cachés.* Thèse de doctorat, Université de Nancy I, 1990.

[25] A. KUNDU, Y. HE, P. BAHL. Recognition and Handwritten Word : First and Second Order Hidden Markov Model Based Approach. *Pattern Recognition*, 22(3) : 283-297, July, 1989.

[26] F. JELINEK, L.R. BAHL, R.L. MERCER. A Maximum Likelihood Approach to Continuous Speech Recognition. *IEEE-PAMI*, PAMI-5(2) : 179-190, March 1983.

[27] F. LEBOURGEOIS. Approche mixte pour la reconnaissance de documents imprimés. Thèse de 3e cycle, INSA Lyon, 1991.

[28] L.C. LIU, D. CHIOU, H.C. WANG. A Speech Recognition Method Based on Feature Distributions. *Pattern Recognition*, 24(8) : 717-722, 1991.

[29] J. MANTAS. An Overview of Character Recognition Methodologies. *Pattern Recognition*, 19(6) : 425-430, 1986.

[30] J.F. MARI. Reconnaissance de mots enchaînés à l'aide de modèles markoviens discrets. In *Actes 5ème Congrès AFCET de Reconnaissance des Formes et Intelligence Artificielle*, p. 859-867, Grenoble, 1985.

[31] W.D. MAY. 3-D Images from Contour Maps. *Dr. Bobb's Journal*, p. 18-28, 1987.

[32] G. NAGY, S. SETH, S.D. STODDARD. Document Analysis with an Expert System. In *Pattern Recognition In Practice II*, Amsterdam, June 19-21, 1986.

[33] D.L. NEUHOFF. The Viterbi Algorithm as an Aid in Text Recognition. *IEEE Transactions Information Theory*, 21 : 222-226, 1975.

[34] L.R. RABINER. *Mathematical Foundations of Hidden Markov Models*, vol. 46 of *NATO ASI*, chapter Recent Advances in Speech Understanding and Dialog Systems. H. Niemann Ed., Springer-Verlag, Berlin, Heidelberg, 1988.

[35] L.R. RABINER. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceeedings of IEEE*, 77(2), February, 1989.

[36] L.R. RABINER, S.E. LEVINSON, M.M. SONDHI. On the Application of Vector Quantization and Hidden Markov Models to Speaker-Independent, Isolated Word Recognition. *The Bell System Technical Journal*, vol. 62 (No. 4) : 1075-1105, April 1983.

[37] L.R. RABINER, S.E. LEVINSON, M.M. SONDHI. *On the Use of Hidden Markov Model for Specaker-Independent recognition of Isolated Words from a Medium-size Vocabulary*, AT&T Bell Laboratory Technical Journal, p. 627-642, 1984.

[38] J. RATCLIFF. Pattern Matching by Geslat. *Dr. Dobb's Journal*, Issue 181, 1988.

[39] R. RUBINSTEIN. *Digital Typography : An Introduction to Type and Composition for Computer System Design.* Addison Wesley, 1988.

[40] J. SCHURMANN. A Multifont Word Recognition System for Postal Address Reading. *IEEE Transactions on computers*, C-27(8) : 721-732, 1978.

[41] R. SHINGHAL, G.T. TOUSSAINT. Experiments in Text Recognition with Modified Viterbi Algorithm. *IEEE Transactions on PAMI*, 2(1) : 184-192, March, 1979.

[42] R.M.K. SINHA. On Partionning a Dictionary for Visual Text Recognition. *Pattern Recognition*, 23(5) : 497-500, 1990.

[43] J.T. TOU, R.C. GONZALEZ. *Pattern Recognition Principles.* Addison-Wesley publishing Company, 1974.

Manuscrit reçu le 27 avril 1993.

LES AUTEURS

Abdel BELAID

Abdel Belaïd a fait ses études à l'Université Louis Pasteur à Strasbourg de 1972 à 1976, puis a rejoint l'Université de Nancy 1 et le CRIN (Centre de Recherche en Informatique de Nancy) où il a obtenu sa thèse de 3^e cycle en 1979 et sa thèse d'état en 1987. Après quelques années d'enseignement en tant qu'assistant, puis maître assistant à l'Université de Nancy 1, il est Chargé de recherche au CNRS depuis 1984. Ses domaines de recherche sont le traitement d'images et la reconnaissance des formes où il a publié une cinquantaine d'articles. Il est co-auteur d'un livre intitulé : Reconnaissance des formes : méthodes et applications. Il anime un groupe de recherche au CRIN autour de plusieurs projets sur l'analyse de documents et la reconnaissance de l'écriture. Il est membre de SPECIF et de l'Association Française pour la Cybernétique Economique et Technique (AFCET).

Julian C. ANIGBOGU

Julian C. Anigbogu est titulaire d'un D.E.A. obtenu en 1990 à l'Université de Nancy 1. Il a obtenu le titre de Docteur de l'Université de Nancy 1 en septembre 1992; le titre de sa thèse est : Reconnaissance de textes imprimés multifontes à l'aide de modèles stochastiques et métriques. Depuis décembre 1993, il travaille chez Schlumberger aux U.S.A. où il poursuit ses travaux de recherche sur les bases de données distribuées.