Fusion d'informations et adaptation pour la reconnaissance de textes manuscrits dynamiques

Information fusion and adaptation for on-line text recognition

Loïc Oudot, Lionel Prevost, Maurice Milgram

Laboratoire des Instruments et Systèmes d'Ile de France,
Groupe Perception, Automatique et Réseaux Connexionnistes
BC 252, 4 Place Jussieu
75252 Paris Cedex 05
lionel.prevost@lis.jussieu.fr

Manuscrit reçu le 19 mai 2004

Résumé et mots clés

Dans cet article nous présentons un nouveau système de reconnaissance de textes manuscrits scripts en mode omni-scripteur. Ce système utilise un lexique français de très grande taille (200 000 mots), qui couvre de nombreux champs d'application. Le processus de reconnaissance repose sur le modèle d'activation-vérification proposé en psychologie perceptive. Un ensemble d'experts code le signal d'entrée et extrait des informations probabilistes à différents niveaux d'abstraction (géométrique, morphologique). Un expert de segmentation neuronal génère un treillis d'hypothèses qui est exploré par un expert de fusion probabiliste qui utilise toute l'information disponible (géométrique, morphologique et lexicale) afin de fournir la meilleure retranscription du signal d'entrée. Nous avons expérimenté plusieurs stratégies d'adaptation non supervisée au scripteur. La meilleure, appelée « adaptation non-supervisée dynamique » agit en continu sur les paramètres du système. Elle permet d'atteindre des performances proches de l'une adaptation supervisée. Les performances, évaluées sur une base de données comportant 90 textes (5 400 mots) écrits par 38 utilisateurs différents, sont très encourageantes car elles atteignent un taux de reconnaissance de 90 %.

Écriture dynamique, reconnaissance de l'écriture scripte, adaptation automatique, fusion d'informations, concepts perceptifs.

Abstract and key words

In this paper, we present a new writer independent system dedicated to the automatic recognition of on-line hand-printed texts. This system uses a very large French lexicon (200000 words), which covers numerous fields of application. The recognition process is based on the activation-verification model proposed in perceptive psychology. A set of experts encodes the input signal and extracts probabilistic information at several levels of abstraction (geometrical and morphological). A neural expert generates a tree of segmentation hypotheses. It is explored by a probabilistic fusion expert that uses all the available information (geometrical, morphological and lexical) in order to provide the best transcription of the input signal. We experiment several strategies of self-supervised writer-adaptation on this system. The best one, called "dynamic self-supervised adaptation", modifies the recognizer parameters continuously. It gets recognition results close to supervised methods. These results are evaluated on a database of 90 texts (5400 words) written by 38 different writers and are very encouraging as they reach a recognition rate of 90%.

On-line handwriting, hand-printed recognition, automatic adaptation, information fusion, perceptive concept.

1. Introduction

La reconnaissance automatique de l'écriture est aujourd'hui plus que jamais un enjeu important. Il suffit d'observer la multiplication des appareils électroniques qui utilisent un stylo comme moyen d'interaction avec la machine pour s'en convaincre. Les ordinateurs, les assistants personnels, les livres électroniques et même les téléphones portables intègrent tous aujourd'hui un stylo. Leur succès est grandissant même s'ils ne possèdent pas toujours un système de reconnaissance de l'écriture efficace. Pour pouvoir être adopté comme moyen d'acquisition, le stylo doit amener un confort au moins équivalent voire supplémentaire par rapport au traditionnel couple clavier/souris.

Le problème réside dans l'absence d'algorithme de lecture automatique de l'écrit. Les systèmes de reconnaissance d'écriture actuellement commercialisés (*Calligrapher*, *Handwriting Recognition Microsoft*…) sont encore trop contraignants dès lors qu'ils imposent un style d'écriture fixe obligeant l'utilisateur à ré-apprendre à écrire. À l'opposé, les systèmes développés en laboratoire, capables de reconnaître l'écriture naturelle, n'obtiennent pas forcément des performances suffisantes pour envisager une commercialisation.

Nous présentons dans cet article un moteur de lecture automatique de textes scripts omni-scripteur. En choisissant de reconnaître l'écriture scripte, nous nous positionnons à un niveau de contrainte intermédiaire puisque nous libérons l'utilisateur de la contrainte sur le style d'écriture tout en maintenant une contrainte sur la segmentation.

L'adaptation de ce système à l'écriture d'un utilisateur particulier permet d'améliorer grandement ses performances. Les méthodes classiques demandent une intervention humaine pour cet apprentissage. Nous proposons donc plusieurs méthodes d'adaptation non-supervisée au scripteur et les comparons aux techniques déjà existantes de type supervisé ou à enrolement.

Dans la deuxième section nous présenterons l'état d'avancement des recherches en psychologie perceptive et des moteurs de lecture s'en inspirant. Puis une revue des différentes techniques d'adaptation supervisée et non supervisée est présentée. La troisième section décrit en détail le fonctionnement du moteur de lecture. Nous décrivons les techniques d'adaptation dans la quatrième partie. Enfin, les conclusions et perspectives sont présentées dans la cinquième et dernière partie.

2. État de l'art

2.1. Systèmes basés sur une modélisation perceptive de la lecture

Pour expliquer le déroulement du processus de lecture, les chercheurs en psychologie cognitive apparentent le cerveau du lecteur à un système informatique de traitement de l'information qui comprend dans sa version minimale un processeur, plusieurs mémoires, des organes récepteurs et d'éventuelles effecteurs [Baccino].

Les modèles cognitifs de lecture développés par les spécialistes ne cherchent pas seulement à modéliser ce traitement de l'information, mais tentent également de modéliser l'importance du contexte et des effets contextuels observés chez l'Homme (effet de supériorité du mot, effet d'amorçage… [Taft]) dans la reconnaissance visuelle des mots. En effet, la pertinence d'un modèle cognitif de lecture est précisément évaluée en fonction de son aptitude à rendre compte des différents effets contextuels.

Le modèle général à triple voie (sémantique, phonologiques [Coltheart]) n'a pas encore été exploité. Quelques systèmes [Côté, Pasquer] exploitent le modèle d'acti-vation interactive de [McClelland]. La plupart des systèmes ([Plamondon, Carbonnel, Connel 00]…), dont le notre, se basent sur le modèle d'activation vérification proposé par [Paap].

Dans ce modèle, le stimulus visuel d'entrée déclenche l'activation de certains mots du lexique (silhouette identique, orthographe proche…). Les mots ainsi activés constituent une liste de mots candidats. À l'étape de vérification, cette liste de mots est confrontée aux informations données en entrée (le stimulus de départ) afin de déterminer le meilleur candidat. La probabilité d'une réponse est la somme pondérée de la probabilité de réponse correcte basée sur une évidence lexicale et celle basée sur une évidence alphabétique.

2.2. Adaptation au scripteur

Le principe de l'adaptation au scripteur à également été mis en lumière par les travaux en psychologie perceptive. On peut remarquer que dans le cas d'une écriture difficile à lire, on lira plus facilement un mot d'une personne si on a auparavant lu un autre mot de cette même personne. Il s'agit d'un effet d'amorçage graphémique. Il est facile de constater ce phénomène dans la pratique lors de la lecture d'un texte très peu lisible. Quand un mot est illisible ou ambiguë, nous recherchons instinctivement dans ce texte, d'autres mots que nous pouvons lire pour comprendre ce premier mot. Nous apprenons donc les caractéristiques du scripteur à partir de mots que nous avons pu lire, pour ensuite utiliser ces nouvelles connaissances sur le reste des mots auparavant illisibles.

Dans la littérature on considère deux stratégies d'adaptation : les systèmes où l'adaptation a lieu une fois pour toute avant utilisation (hors-ligne ou *batch*) et les système à adaptation continue (en-ligne).

2.2.1. Adaptation hors-ligne

La plupart des systèmes utilisant une adaptation hors-ligne possèdent une base de données du scripteur étiquetée. Ces exemples permettent de réaliser un apprentissage supervisé

(enrolement) du système, c'est-à-dire d'apprendre le style d'écriture du scripteur, avant son utilisation.
Dans [Li] les lettres sont décrites par plusieurs modèles neuronaux markoviens en parallèles, chacun ayant appris un certain allographe de lettre. Ces derniers sont déterminés par un algorithme de clustering sur la base d'apprentissage. Ce système s'adapte au scripteur et réalise une modélisation plus fine des lettres en ré-estimant les paramètres des modèles sur les exemples étiquetés. Le système décrit par [Schomaker] utilise également un classifieur automatique de *strokes* par cartes de Kohonen pour identifier les primitives utilisées par le scripteur. Ainsi le classifieur de caractères n'utilise plus que les primitives utilisateurs au lieu de celles omni-scripteur. Dans [Brakensiek], les HMM sont entraînés dans un premier temps sur une base omni-scripteur puis spécialisés à un scripteur particulier. Dans [Connel 02], en plus de la ré-estimation des paramètres, de nouveaux modèles de caractères sont crées si nécessaire.

2.2.2. Adaptation en-ligne

Contrairement aux systèmes décrits plus haut qui sont figés une fois l'adaptation effectuée, les systèmes suivants évoluent en permanence au cours de l'utilisation.
Le système de lecture en-ligne avec adaptation au scripteur de [Platt] utilise une méthode d'adaptation incrémentale supervisée. Le système de base est omni-scripteur et utilise un unique réseau de neurones de type MLP (RN) possédant 72 classes de sortie (62 lettres et 10 signes de ponctuations). Un module d'adaptation disposé en sortie du RN permet de modifier son vecteur de sortie. Ce module d'adaptation est constitué par un réseau de type RBF (*Radial Basis Function*). À chaque erreur de classification, l'utilisateur prévient le système de l'erreur ce qui entraîne l'apprentissage des RBF (ré-estimation des centres existants ou ajout d'un nouveau centre) à partir de l'étiquette entrée par l'utilisateur.
Deux autres systèmes assez proches utilisent un TDNN (*Time Delay Neural Network*) comme classifieur à la place du RN. Ce TDNN est appris sur une base omni-scripteur et la couche de sortie de ce réseau est remplacée soit par un classifieur de type k-pvv [Guyon] soit par un classifieur discriminant [Matić].
Le système de [Vuori] est très proche du notre mais se limite à la reconnaissance de caractères alphanumériques isolés. Le classifieur de caractères est de type k-ppv et utilise comme métrique une distance élastique. L'adaptation consiste à ajouter les caractères non reconnus dans la base de prototypes. Les prototypes non utilisés peuvent être supprimés de la base pour éviter une croissance excessive de celle-ci.

3. Système de lecture

Pour les expérimentations, nous avons collecté une base de 90 textes écrits par 54 scripteurs différents réunissant 26 000 lettres, 5 400 mots et 3 000 signes de ponctuation (figure 1). Cette base a été divisée en deux parties égales : une première base pour l'estimation des paramètres du système de lecture (BA_{texte}) et une seconde pour son évaluation (BT_{texte}). Chaque base contient 45 textes. La base d'apprentissage comprend 25 scripteurs dont 14 en commun avec la base de test. La base de test quand à elle, contient 29 scripteurs et inclut donc 15 scripteurs totalement inconnus de la base d'apprentissage. Le fait de ne pas totalement différencier les scripteurs entre les deux bases permet de mieux évaluer les résultats et la pertinence du système et de comparer simultanément son comportement dans les cadres omni-scripteur et multi-scripteur.

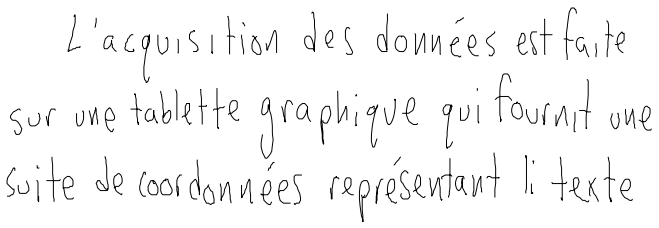

Figure 1. Exemple de texte de BA_{texte}.

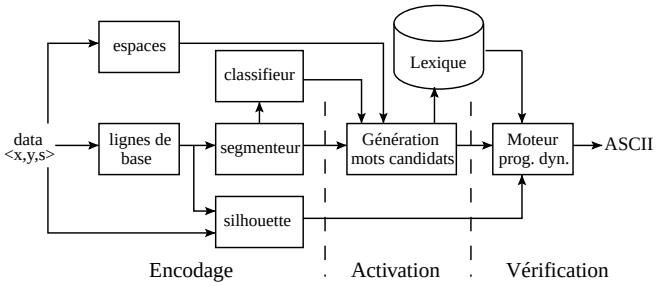

Figure 2. Structure générale du système de lecture.

La structure générale du moteur de lecture (figure 2) s'inspire du modèle d'activation vérification. Le système se présente sous la forme d'une série d'experts d'encodage qui permettent d'extraire les informations de type géométrique et morphologique du texte écrit par l'utilisateur. Ces experts fournissent des informations probabilistes au niveau *stroke*. Un premier expert de pré-traitement, non représenté sur la figure, segmente le flux d'entrée en lignes car notre système traite les informations ligne de texte par ligne de texte. Toutes ces informations, aussi bien au niveau local qu'au niveau global, permettent d'activer une liste de mots du lexique. La cohérence de chaque mot hypothèse de la liste est ensuite évaluée par un moteur probabiliste qui valide la meilleure hypothèse pour la retranscription.
Nous avons évalué ce système à l'aide de deux lexiques français de taille différente. Le premier ($Dico_{fr}$) contient 185 000 mots et couvre un très large champ d'applications. Le second ($Dico_{rd}$) contient les 8 000 mots les plus fréquents de la langue française. Les taux de bonne classification donnés par la suite ont tous été évalués sur la base de test BT_{texte}.

3.1. Experts d'encodage

3.1.1. Détection des sauts de ligne

Cet expert recherche les sauts de lignes pour segmenter le flux de données d'entrée en ligne. Les techniques les plus courantes utilisent le contour du texte [Senior] ou ajustent des lignes paraboliques parallèles par un processus itératif [Caesar].
Notre expert approxime les lignes d'assises par morceau avec des lignes droites. Pour cela, il recherche les strokes représentant des lettres médianes (*a, c, e, i, n…*) pour les relier ensuite par des lignes droites. On relie le bas des boîtes englobantes de ces derniers pour obtenir la ligne de base et le haut des boîtes pour obtenir la ligne de corps.

3.1.2. Caractérisation de la silhouette

Le rôle de cet expert est de déterminer la forme ou la silhouette des mots en utilisant les lignes de bases et d'extraire ainsi des informations globales [Powalka]. Cet expert estime pour chaque stroke s_i deux probabilités correspondant à la présence d'une hampe $p(h|s_i)$ et d'un jambage $p(j|s_i)$. De plus, à partir de ces deux informations, on attribue au stroke une classe d'activation Act_i parmi les quatre suivantes : *médiane (m), hampe (h), jambage (j)* et *hampe/jambage (f)*.
L'estimation de ces probabilités est basée sur l'utilisation des lignes d'assises calculées par l'expert précédent. L'estimation de $p(m|s_i)$ et $p(f|s_i)$ est déduite des deux probabilités précédentes :

$$- \; p(m|s_i) = \frac{1 - \max(p(h|s_i), p(j|s_i))}{2}$$

$$- \; p(f|s_i) = \frac{p(h|s_i) + p(j|s_i)}{2}$$

La classe d'activation Act_i est déterminée par un réseau de neurones de type MLP (2-3-4) entraîné sur la base d'apprentissage. Nous obtenons un taux de bonne classification de 94,7%.

3.1.3. Détection des espaces

Cet expert fournit pour chaque arc inter-strokes a_i une probabilité $p(e|a_i)$ d'être un espace séparant deux mots distincts. Cette information est précieuse pour la localisation des mots dans un ligne de texte.
Un réseau de neurones minimise la probabilité d'erreur de classification à partir de la distance horizontale inter-strokes. Afin de disposer d'une probabilité en sortie de ce réseau, la cellule de sortie utilise une fonction de transfert linéaire saturée entre 0 et 1. Le taux de bonnes classifications obtenu est de 95,4%. On constate que le détecteur a tendance à sous segmenter les mots en fusionnant deux ou plusieurs mots consécutifs. Mais ce phénomène est avant tout caractéristique de l'écriture scripte ; nombre d'utilisateurs séparant autant les lettres d'un même mot que les mots entre eux (égalité des espaces intra-mot et inter-mots).

3.1.4.Segmenteur

Les lever de stylo, tout comme les *strokes*, contiennent une grande quantité d'informations. Comme [Gader] on utilise un réseau de neurones pour classifier ces arcs inter-strokes. À partir des données d'entrée, on construit un graphe non orienté $G = (S, A)$ avec un ensemble de sommets constitués des strokes $S = \{s_1, s_2, \ldots, s_n\}$ et des arêtes $A = \{(s_1, s_2), (s_2, s_3), \ldots, (s_{n-1}, s_n)\}$ correspondant aux arcs reliant deux strokes consécutifs du point de vue temporel. On distingue cinq classes d'arcs : *intra-lettre* (A_1), *inter-lettres & intra-mot* (A_2), *inter-mots* (A_3), *diacritique* (A_4) et *ponctuation* (A_5).
32 paramètres servent à caractériser les arcs inter-strokes [Oudot]. Ils sont estimés en considérant pour deux strokes consécutifs s_n et s_{n+1}, leurs coordonnées, leurs boîtes englobantes, leurs positions par rapport aux lignes de bases et le lever de stylo (défini comme le segment joignant le dernier point de s_n au premier point de s_{n+1}). L'idéal étant de trouver des paramètres invariants aux scripteurs (paramètres omni-scripteurs).
Le meilleur classifieur obtenu est un réseau MLP (32-20-5). Le taux de reconnaissance est de 92,3% pour la première réponse (top_1) et de 99,1% pour le top_2 sur la base de test.

3.1.5. Classifieur de caractères

Le classifieur de caractères évalue pour chaque stroke les probabilités d'appartenance aux 62 classes (26 minuscules, 26 majuscules et 10 chiffres) : $p(c|data)\ c \in \Gamma\ \Gamma = \{A, \ldots, Z, a \ldots, z, 0 \ldots, 9\}$.
Le classifieur à base de prototypes [Prevost 98] est de type 1-ppv. L'exemple inconnu sera comparé au corpus de prototypes de la classe correspondante. La métrique utilisée est la distance élastique, largement utilisée en reconnaissance de caractères dynamiques [Vuori 99]. Les probabilités sont obtenues par l'utilisation de la règle du *softmax*.
Les bases de prototypes du classifieur sont obtenus par un algorithme de clustering nommé SMAC [Prevost 00]. Une première base de prototypes extraite de la base UNIPEN donne des résultats décevants. Pour améliorer ces résultats, deux autres bases ont été crées à partir de la base de texte d'apprentissage BA_{texte}. La première, extraite des textes écrits par les scripteurs qui n'apparaissent pas dans la base de test, permet d'évaluer le comportement du classifieur en contexte omni-scripteur. La seconde, extraite de l'ensemble des textes de la base d'apprentissage, correspond au contexte multi-scripteur. On constatera (tableau 1) les améliorations de performance successives.

Tableau 1. Taux de reconnaissance du classifieur dynamique sur la base d'évaluation BT_{texte} en fonction de la base de prototypes.

Base de prototypes	top_1	top_2
UNIPEN	78,9%	80,2%
Omni-scripteur	84,8%	87,3%
Multi-scripteur	88,7%	90,5%

3.2. Moteur de lecture

3.2.1. Génération du treillis de segmentation

Une technique classique pour retrouver la segmentation d'un texte, est de générer un treillis d'hypothèses de segmentation [Powalka, Henning] et de laisser aux modules suivants (en général le classifieur de caractère et/ou le lexique) la prise de décision. L'inconvénient avec cette technique est de trouver un bon compromis entre le nombre d'hypothèses générées et la vitesse de traitement. Générer beaucoup d'hypothèses permet d'augmenter la probabilité d'obtenir la bonne segmentation mais risque de perturber et ralentir le système par un trop grand nombre d'hypothèses. Il faut donc réussir à générer un minimum d'hypothèses les plus pertinentes possibles.

Générer simplement le treillis en ne conservant que les deux meilleurs résultats (top_2) du segmenteur permet d'atteindre 99,1% d'apparition de la bonne segmentation. Mais un tel treillis possède environ 10^{12} hypothèses pour une ligne moyenne de 40 *strokes* et est donc inutilisable. Rappelons que le taux d'apparition correspond à l'existence de la segmentation réelle dans le treillis et non au taux de bonne segmentation.

Pour réduire le nombre d'hypothèses de segmentation, nous utilisons le détecteur d'espaces inter-mots (classe A_3) pour fixer des points d'ancrage dans la ligne. Ces derniers découpent le treillis de segmentation en plusieurs sous-treillis plus petits et cassent la combinatoire. En considérant les arcs ayant une probabilité $p_{espace} = 1$ comme étant des point d'ancrage, on retrouve 50% des espaces inter-mots avec moins de 1% de fausses détections.

Le treillis de segmentation est généré progressivement en ajoutant les types d'arcs les plus probables jusqu'à atteindre N_{hypo} hypothèses de segmentation. La pertinence d'une classe d'arc pour un arc donné est calculée en fusionnant les probabilités *a priori* et *a posteriori* du segmenteur :

$$p(A_i|data) = \sum_{a=A_1}^{A_5} p(A_i|a)\,p(a|data)$$

Avec $p(a|data)$ la probabilité *a posteriori* issue du segmenteur et $p(A_i|a)$ la probabilité *a priori* déduite de la matrice de confusion de ce dernier. Cette matrice est estimée sur la base d'apprentissage BA_{texte} et synthétise le comportement du segmenteur observé sur cette base. Si la base d'apprentissage est statistiquement représentative, il y a de fortes probabilités que ce comportement soit généralisable.

Cette méthode donne de très bons résultats : avec un treillis limité à $N_{hypo} = 500$ hypothèses, on obtient 98,4% de bonne apparition de la segmentation et pour $N_{hypo} = 2\,000$ plus de 98,8%.

3.2.2. Activation des mots candidats

L'activation des mots candidats du lexique (modèle d'activation vérification) est introduit par une recherche lexicale dans une sous partie restreinte du lexique principal. En effet, la recherche ne s'effectue que sur les mots de même longueur, l'organisation du lexique étant basée sur le nombre de caractères des mots.

Les mots hypothèses sont obtenus en conservant pour chaque lettre hypothèse, la classe la plus probable du classifieur de caractères et possédant la même silhouette que celle observée par le détecteur de silhouette. Une recherche lexicale en tolérant N_{err} caractères de différence constitue la liste des mots lexicalement corrects. Le seuil $N_{err} = \mathrm{E}(N_{car}/2)$ a été choisit de manière à avoir le meilleur compromis entre le nombre de mots générés et l'apparition du bon mot dans cette liste. Nous obtenons 97,7% de bonne apparition du mot réel sur la base de test avec une moyenne de 50 mots candidats par proposition. Les 2,3% d'erreur restants sont dus à de trop nombreuses erreurs de classification, surtout sur les mots courts où plus de la moitié des lettres sont fausses.

3.2.3. Retranscription du texte

La retranscription d'une ligne de texte se décompose en deux étapes. La première assigne à chaque mot hypothèse une probabilité de cohérence et la seconde effectue une recherche de la ligne la plus probable.

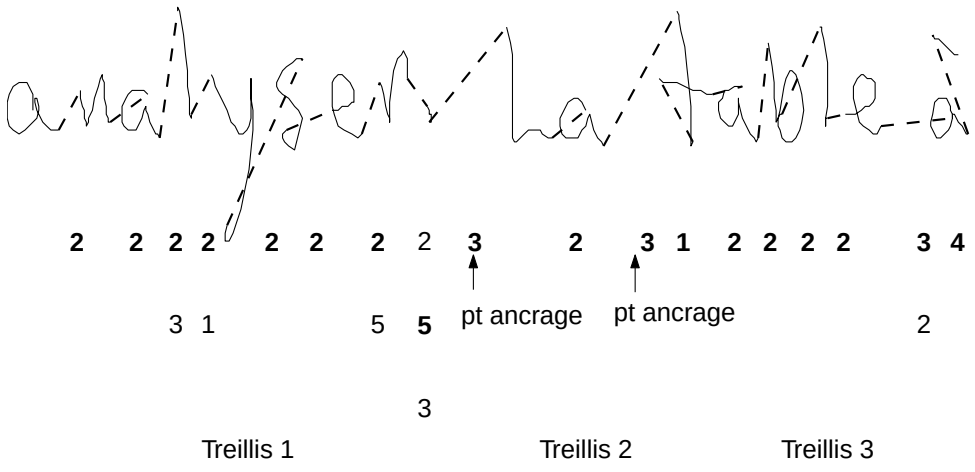

Figure 3. Exemple de treillis de segmentation avec points d'ancrages (26 hypothèses).

La probabilité d'un mot de la liste $p(mot|data)$ est le produit de la probabilité des caractères le composant $p(car|data)$ par la probabilité de sa segmentation $p(seg|data)$.
La probabilité de tous les caractères $p(car|data)$ composant un mot revient à estimer les probabilités de chaque caractère $p(car^i|data)$ à partir des données d'entrée: $p(car|data) = \prod_{i=1}^{N_{car}} p(car^i|data)$.
L'estimation de la probabilité des caractères tient compte des probabilités *a posteriori* du classifieur de caractères ainsi que du détecteur de silhouette et des probabilités *a priori* déduites de la matrice de confusion des ces deux experts.

$$p(car^i|data) = \sum_{(c\in\Gamma, s\in\{m,h,j,f\})} p(car^i|(c,s))\,p((c,s)|data)$$

Le terme $p((c,s)|data)$ correspond à la probabilité d'observer le caractère c ayant la silhouette s connaissant les données d'entrée. Ces informations sont données par deux experts: classifieur de caractères et caractérisation de la silhouette. Si nous considérons les deux sources indépendantes, alors: $p((c,s)|data) = p(c|data)\,p(s|data)$. Le terme $p(car^i|(c,s))$ est déduit de la matrice de confusion.
La probabilité de la segmentation est déduite des probabilité calculées précédemment lors de la génération du treillis (fusion des probabilités *a priori* et *a posteriori* du segmenteur): $p(seg|data) = \prod_{i=1}^{N_{car}} p(s|a_i)$
Chaque mot de chaque hypothèse de segmentation d'une ligne a donc une probabilité associée et une position de début et de fin (en numéro de *stroke*) dans la ligne. Ceci permet de générer un treillis de mots hypothèses. En utilisant un algorithme de programmation dynamique on retrouve le chemin le plus probable à l'intérieur du treillis de mots hypothèses. La force de cet algorithme réside dans le caractère global de la recherche. Le chemin le plus probable retrouvé est une combinaison de plusieurs hypothèses de segmentation (tableau 4).

3.3. Résultats du moteur de lecture

Les taux de reconnaissance en mots et en lettres obtenus pour $N_{hypo} = 500$ sont donnés dans le tableau 2.
On constate que l'utilisation d'un très grand lexique ne fait chuter le taux de reconnaissance en mots que d'environ 3%.

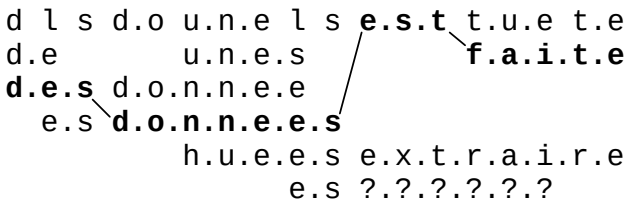

Figure 4. Exemple de treillis de mots (les points correspondent à des espaces intra-mots, les blancs à des espaces inter-mots).

Tableau 2. Taux de reconnaissance en mots et en lettres du moteur de lecture.

Base de proto.	$Dico_{fr}$	$Dico_{rd}$
Omni-scripteur		
mot	71,9%	74,8%
lettre	91,6%	92,0%
Multi-scripteur		
mot	74,8%	77,5%
lettre	93,2%	93,4%

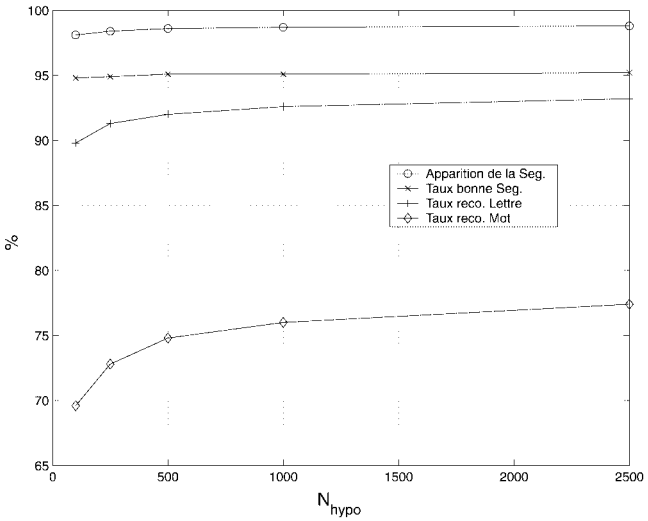

Figure 5. Évolution du taux de reconnaissance en fonction du nombre d'hypothèses de segmentation considéré.

L'amélioration du taux de reconnaissance observée en contexte multi-scripteur montre l'intérêt d'une adaptation du moteur de lecture à son utilisateur. La figure 5 montre l'évolution de l'apparition de la bonne segmentation dans le treillis ainsi que le taux de bonne segmentation retrouvée par la programmation dynamique en fonction du nombre d'hypothèses explorées N_{hypo}. Le taux de reconnaissance en lettres et en mots est également présenté et montre que les résultats peuvent être encore améliorés en considérant un plus grand nombre d'hypothèses.

4. Adaptation par modification des paramètres de classification

Lors de l'étude précédente du moteur de lecture, nous avons pu constater que la base de prototypes du classifieur, aussi riche soit-elle, n'est pas encore assez représentative, et que l'utilisa-

tion des caractères tirés de la base de textes d'apprentissage pour une utilisation en mode multi-scripteur, permet d'améliorer les taux de reconnaissance. Il y a deux sources majeures d'erreurs de classification :

- Le graphème est absent de la base omni-scripteur, il faut donc enrichir cette dernière : le graphème doit être stocké dans la base de prototypes (Ajout).
- Le graphème est confusif : pour un scripteur donné, les modèles de la base omni-scripteur doivent être supprimés afin d'éviter toute confusion (Suppression).

Les classifieurs à prototypes (1-ppv) peuvent s'adapter aisément en ajoutant un nouvel exemple de caractère dans la base de prototypes utilisateur et en désactivant les prototypes existants.

4.1. Adaptation supervisée

Pour évaluer la pertinence de l'adaptation au scripteur, nous avons utilisé les étiquettes des textes pour effectuer une adaptation supervisée en segmentation connue. Les formes du texte sont classifiées l'une après l'autre. Les hypothèses de classification (qui correspondent aux meilleures réponses, top_1, du classifieur de caractères) sont comparées aux étiquettes. Si elles ne correspondent pas, la forme en question est ajoutée à la base utilisateur (figure 6). Deux approches ont été envisagées : l'approche *texte* où les formes sont ajoutées en fin de texte et l'approche *ligne* où les formes sont ajoutées en fin de chaque ligne. Les résultats (tableau 3) montrent les gains d'adaptation de notre système de lecture lorsque la segmentation du texte en mots et en lettres est connue.

L'approche *ligne* permet une amélioration plus rapide du taux de reconnaissance tout en ajoutant moins de prototypes par scripteur que l'approche *texte*. En effet, en ajoutant les prototypes à la fin de l'analyse complète d'un texte, on peut ajouter plusieurs prototypes semblables d'un même caractère (d'où l'augmentation du nombre moyen de prototypes). L'approche ligne, par contre, ajoute le premier prototype du caractère mal classifié. Les caractères suivants étant reconnus grâce à cet ajout, ils ne viennent pas modifier la base de prototypes. Le nombre moyen de prototypes ajoutés est donc plus faible. D'un point de vue perceptif, cet ajout de prototypes mime – au niveau lettre – l'effet d'amorçage par répétition constaté au niveau mot : la présentation antérieure d'un mot réduit la quantité d'information nécessaire à son identification ultérieure et rend cette dernière plus rapide.

Tableau 3. Évolution du taux d'erreur mots (segmentation connue, adaptation supervisée). La dernière colonne correspond au taux de croissance de la base de prototypes utilisateur.

	Taux d'erreur			Base utilisateur
Mots	50	100	150	
Sans adaptation		15 %		100 %
App. texte : min	0 %	0 %	0 %	+3 %
moy	1,3 %	1,1 %	0,6 %	+6 %
max	10 %	5,1 %	4,5 %	+9 %
App. ligne : min	0 %	0 %	0 %	+2 %
moy	**1,1 %**	**0,7 %**	**0,4 %**	**+4 %**
max	6,2 %	5,2 %	3,7 %	+8 %

4.2. Adaptation non-supervisée

L'adaptation se fait maintenant de façon complètement transparente pour le scripteur qui n'est plus sollicité. Les informations produites par le classifieur de caractères d'une part et le correcteur lexical d'autre part, sont comparées pour déterminer les prototypes à ajouter (figure 6).

4.2.1. Ajout systématique

Dans la stratégie d'adaptation par ajout systématique, on considère que le taux d'erreur du correcteur lexical est nul. Donc, à chaque erreur (différence entre l'hypothèse de classification et

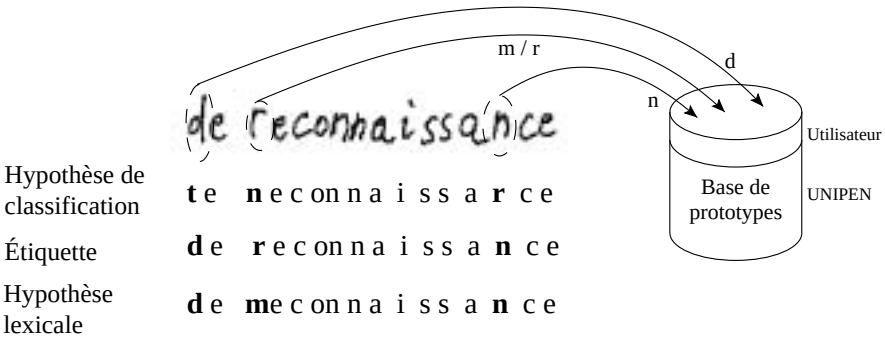

Figure 6. Ajout de prototypes dans la base utilisateur (comparaison étiquette hypothèse de classification). Adaptation supervisée et non-supervisée.

l'hypothèse lexicale) le caractère est ajouté à la base de prototypes utilisateur. Les erreurs de correction lexicale cumulées à celles de segmentation conduisent à des ajouts de prototype dans de mauvaises classes. Ces erreurs d'ajout entraînent par la suite de nombreuses erreurs de classifications. Les performances du système de lecture s'en ressentent et diminuent drastiquement (tableau 4) mais restent tout de même meilleures qu'en l'absence d'adaptation.

4.2.2. Ajout conditionnel

La procédure d'ajout systématique étant peu performante, il semble nécessaire d'étudier le comportement de l'analyseur lexical pour n'ajouter que des prototypes utiles. Deux conclusions s'imposent lors de cette étude [Oudot] :
- La fiabilité de l'analyseur lexical augmente avec le nombre de lettres composant le mot.
- La fiabilité de l'analyseur lexical diminue lorsque le nombre d'erreurs (lettres du mot mal classées) de classification augmente.
Le moteur de reconnaissance estime pour chaque mot une probabilité de cohérence lexicale (PCL). L'ajout conditionnel consiste à n'utiliser, pour l'ajout de prototype, que les mots dont la PCL est supérieure à un seuil α. Ce seuil à été déterminé par recherche exhaustive. Les meilleurs résultats ont été obtenus avec un $\alpha = 0.015$ et sont présentés dans le tableau 4.

Tableau 4. Ajout conditionnel : évolution du taux d'erreur mots et de la base utilisateur.

	Taux d'erreur			Base utilisateur
Mots	50	100	150	
Sans adaptation		28 %		100 %
Systématique: min	0 %	1,9 %	2 %	+2 %
moy	25 %	23 %	23 %	+6 %
max	53 %	73 %	51 %	+14 %
Conditionnel: min	0 %	0 %	2 %	+1 %
moy	**22 %**	**20 %**	**17 %**	**+2 %**
max	71 %	58 %	43 %	+3 %

L'ajout conditionnel permet de réduire considérablement les erreurs d'ajout de prototypes de la procédure d'ajout systématique. On voit très nettement l'amélioration du taux de reconnaissance avec un gain d'adaptation de 35 %.

4.2.3. Gestion dynamique des prototypes

Cette stratégie a deux buts. La suppression des prototypes erronés dus aux erreurs lors de l'ajout conditionnel (erreur du correcteur lexical) et la réduction de la taille de la base utilisateur en supprimant les prototypes inutiles [Vuori 02] afin d'accélérer la vitesse de classification. On attribue à chaque prototype (de la base omni-scripteur comme de la base utilisateur) une adéquation initiale ($Q(0) = 1\,000$). Cette adéquation sera modifiée à chaque occurrence (chaque fois que le prototype est le plus-proche-voisin du caractère à classer) en comparant l'hypothèse de classification à l'hypothèse lexicale (équation 1). Considérons le $i^{ème}$ prototype de la classe j, trois paramètres sont nécessaires à la gestion dynamique de son adéquation Q_j^i :
- **C** : Récompense (+) du prototype i quand l'hypothèse de classification est **C**orrecte (identique à l'hypothèse lexicale).
- **I** : Pénalisation (-) du prototype i quand l'hypothèse de classification est **I**ncorrecte (différente de l'hypothèse lexicale).
- **N** : Pénalisation (-) pour les autres prototypes **N**on utilisés de la classe j.

$$Q_j^i(n+1) = Q_j^i(n) + [C(n) - I(n) - N(n)]/F_j \qquad (1)$$

où F_j est la fréquence de la lettre j dans la langue française. Les trois paramètres sont mutuellement exclusifs *i.e.* à chaque occurrence, seul un paramètre est actif. Lorsque $Q_j^i = 0$, le prototype est éliminé.
Après recherche exhaustive des paramètres (C, I, N) optimaux pour une combinaison avec la méthode d'ajout conditionnel, les meilleurs résultats sont obtenus avec le jeux de paramètres suivant (30, 200, 8). Comme on peut le constater (tableau 5), cette procédure de gestion dynamique permet de conserver le taux de reconnaissance obtenu avec adaptation par ajout conditionnel, tout en diminuant de 40% la taille de la base de prototypes (et le temps de traitement).

Tableau 5. Gestion dynamique des prototypes : évolution du taux d'erreur et de la base utilisateur

	Taux d'erreur	Base utilisateur
Sans adap.	28 %	100 %
Gestion dyn.	**17 %**	**−40 %**

Nous allons étudier l'évolution de l'adéquation de quelques prototypes (figure 7). Pour certains utilisateurs, les prototypes omni-scripteur sont suffisants. Pour la classe 'a', 2 prototypes sont utilisés et donc l'adéquation des 45 autres décroît. Pour la classe 's', 4 prototypes sont utilisés alternativement (le scripteur a probablement une écriture instable, voir figure 8) et les 36 autres sont inactifs. Pour d'autres scripteurs (classe 's' et 'e'), des prototypes utilisateurs (en gras) sont nécessaires. Au départ, un prototype omni-scripteur est utilisé et après quelques occurrences, un prototype utilisateur est ajouté (l'utilisateur s'est habitué à la tablette). Après 150 mots, la taille de la base utilisateur a été réduite de 40 %. Une régression linéaire peut raisonnablement amener cette réduction à plus de 90 % de la taille initiale.

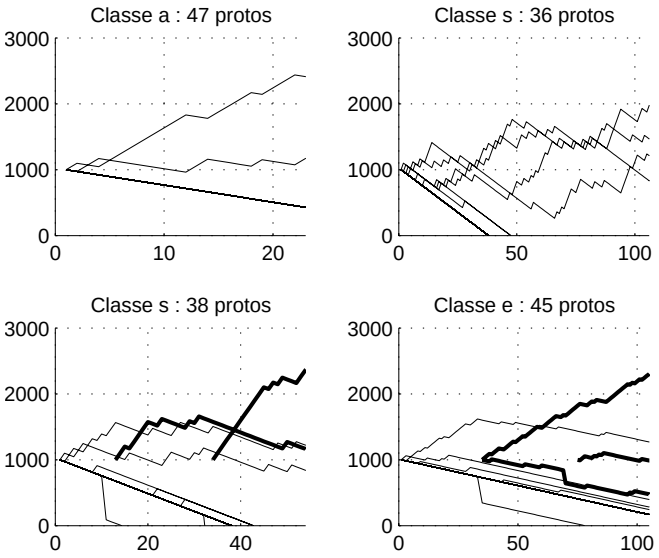

Figure 7. Évolution de l'adéquation des prototypes fonction du nombre d'occurence.

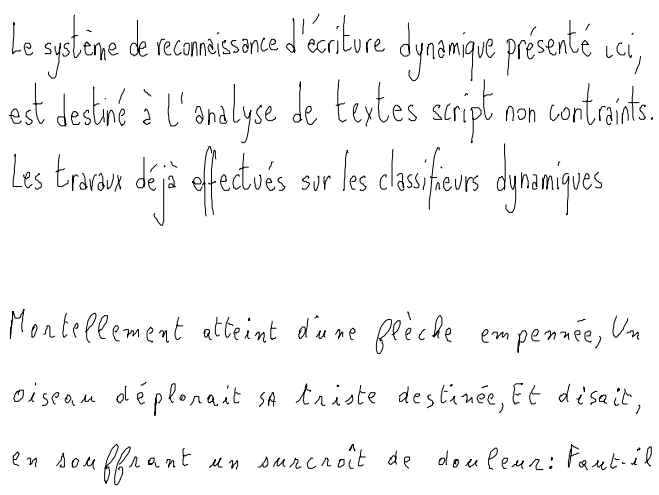

Figure 8. Scripteurs les mieux et moins bien reconnus (respectivement 99 % et 70 % de taux de reconnaissance).

4.3. Combinaison supervisée/non-supervisée

Les performances du système de lecture réel (segmentation inconnue) utilisant la base de prototype modifiée par ajout conditionnel et gestion dynamique s'améliorent avec un taux d'erreur de 12 % au lieu des 28 % du système initial. L'adaptation non supervisée atteint presque les performances du système après enrolement (17 % contre 12 %) sans aucune intervention de l'utilisateur. Solliciter l'utilisateur pour l'écriture de 150 mots est beaucoup trop contraignant, en revanche lui faire écrire quelques dizaines de mots est acceptable surtout si le taux de reconnaissance est grandement amélioré. Cette dernière stratégie combine l'adaptation supervisée du système sur quelques mots et l'adaptation non-supervisée par gestion dynamique. Comme on peut le constater (tableau 6), cette combinaison s'avère très efficace. Ainsi, imposer à l'utilisateur d'écrire une phrase de 30 mots permet d'améliorer *in fine* les performances en portant le taux de reconnaissance à plus de 90 %. Ces résultats pourraient encore s'améliorer en choisissant judicieusement lesdits mots (alors que dans la procédure actuelle, les 30 premiers mots écrits ont été utilisés pour l'adaptation supervisée).

Tableau 6. Taux de reconnaissance, adaptation semi-supervisée

Mot enrolement	Taux de reconnaissance	
	Après enrolement	100 mots
0	70 %	76 %
10	74 %	83 %
20	74 %	88 %
30	**74 %**	**90 %**
50	75 %	91 %

5. Conclusions et perspectives

Nous venons de présenter un système de lecture de textes scripts basé sur le modèle perceptif d'activation vérification. Ce système fonctionne dans un cadre omni-scripteur et est en mesure d'utiliser des lexiques de très grande taille en conservant des taux de reconnaissance élevés. Ce moteur possède de plus un seuil N_t qui permet de fixer la profondeur maximum du treillis de segmentation et ainsi de gérer le compromis *temps de calcul / taux de reconnaissance*. On peut donc choisir le taux de reconnaissance optimum suivant la puissance de calcul disponible.

Notre méthode de génération / validation de la segmentation donne également de très bons résultats avec une moyenne de 95 % de bonne segmentation. De plus, la majorité des erreurs de segmentation sont de fausses détections d'espace inter-mots ce qui entraîne inévitablement une chute du taux de reconnaissance en mots. Les mauvaises détections apparaissent pour des scripteurs peu habitués à l'écriture scripte et qui séparent les lettres d'un même mot autant que les mots entre eux, rendant inefficace le détecteur d'espace. Lorsque la segmentation en mots est connue, le système de lecture voit son taux de reconnaissance augmenter, dépassant 92 % sur les mots de trois lettres et plus.

La très nette augmentation du taux de reconnaissance obtenue sur la base multi-scripteur, montre l'intérêt de s'orienter vers un

mode de reconnaissance multi-scripteur voire même mono-scripteur. Le classifieur de caractères, grâce à son architecture modulaire et à sa base de prototypes indépendante, est parfaitement adapté pour prendre en compte de nouveaux prototypes. Ce qui nous amène à présenter plusieurs stratégies d'adaptation non-supervisée. Le processus d'adaptation combinant les stratégies non-supervisée et supervisée diminue le taux d'erreur (de 28% à 10%) et accroît la vitesse de classification (près du double). De plus, il spécialise une base de prototypes omni-scripteur en une base mono-scripteur de très grande qualité et de faible encombrement mémoire de manière automatique.

Il serait intéressant d'évaluer une stratégie semi-supervisée où l'utilisateur n'est sollicité que dans les cas ambigus. Il s'agit maintenant de modifier l'étape de segmentation qui génère le plus d'erreur : une ré-estimation des paramètres des experts d'encodage devrait être bénéfique.

Références

[Baccino] T. BACCINO, P. COLE, *La Lecture experte*, Presse universitaire de France, 1995.

[Brakensiek] A. BRAKENSIEK, A. KOSMALA, G. RIGOLL, « Comparing adaptation techniques for on-line handwriting recogntion », ICDAR, Vol. 1, 2001.

[Caesar] T. CAESAR, J.M. GLOGER, E. MANDLER, « Preprocessing and Features Extraction for a Handriting recognition System », ICDAR, Vol. 1, 1993, p. 408–411.

[Carbonnel] S. CARBONNEL, E. ANQUETIL, « Modélisation et intégration de connaissance lexicales pour le post-traitement de l'écriture manuscrite en-ligne », RFIA, Vol. 3, 2004, p. 1313–1322.

[Coltheart] M. COLTHEART, *Lexical access in simple reading task*, Underwood, Strategies of information processing, Academic Press, 1978.

[Connel 00] S.D. CONNEL, « On-line Handwriting Recognition Using Multiple Pattern Class Models », PhD thesis, Departement of Computer science and enginnering, Michigan State University, 2000.

[Connel 02] S.D. CONNEL, A.K. JAIN, « Writer adaptation of online handwriting models », PAMI, Vol. 24, #3, 2002, p. 329–346.

[Côté] M. COTE, « Utilisation d'un modèle d'accès lexical et de concepts perceptifs pour la reconnaissance d'images de mots cursifs, PhD thesis, ENST, 1997.

[Gader] P.D. GADER, M. MOHAMED, J.-H. CHIANG, « Handwritten Word Recognition with Character and Inter-Character Neural Networks », IEEE Transaction on systems, Man and Cybernetics, Vol. 27, #1, 1997, p.158–164.

[Guyon] I. GUYON, D. HENDERSON, P. ALBRECHT, Y.L. CUN, J. DENKER, *Writer independent and writer adaptative neural network for on-line character recognition*, From Pixel to Features III, Elsevier, 1992.

[Henning] A. HENNING, N. SHERKAT, « Cursive script recognition using wildcards and multiple experts », Pattern Analysis & Applications, Vol. 4, 2001, p. 51–60.

[Li] H. LI, « Traitement de la variabilité et développement de systèmes robustes pour la reconnaissance de l'écriture manuscrite en-ligne », PhD thesis, UPMC Paris VI, 2002.

[Matić] N. MATIC I. GUYON, J. DENKER, V. VAPNIK, « Writer-adaptation for on-line handwritten character recognition », ICDAR, Vol. 1, 1993.

[McClelland] J.L. MCCLELLAND, D.E. RUMELHART, « An interactive activation model of context effects in letter perception », Psychological Review, Vol. 88, 1981, p. 375–407.

[Oudot] L. OUDOT, « Fusion d'informations et adaptation pour la reconnaissance de textes manuscrits dynamiques », PhD Thesis, Université Pierre & Marie Curie, 2003.

[Paap] K. PAAP, S.L. NEWSOME, J.E. MCDONALD, R.W. SCHVANEVELDT, « An activation-verification model for letter and word recognition: The word superiority effect », Psychological Review, Vol. 89, 1982 p. 573–594.

[Pasquer] L. PASQUER, « Conception d'un modèle d'interprétation multi-contextuelle, application à la reconnaissance en-ligne d'écriture manuscrite », PhD thesis, Université Rennes I, 2000.

[Plamondon] R. PLAMONDON, S.N. SRIHARI, « On-line and off-line handwriting recognition: A comprehensive survey », PAMI, Vol. 22, #1, 2000, p. 63–84.

Platt] J.C. PLATT, N.P. MATIC, « A constructive RBF network for writer adaptation », Advances in neural information procesing systems, Vol. 1, 1997, p. 765–771.

[Powalka] R.K. POWALKA, N. SHERKAT, R.J. WHITROW, « Word shape analysis for a hybrid recognition system », Pattern Recognition, Vol. 30, #3, 1997, p. 421–445.

[Prevost 98] L. PREVOST, « Reconnaissance de l'ecrit dynamique : Application à l'analyse des expressions mathématiques », PhD thesis, Université Paris VI, 1998.

[Prevost 00] L. PREVOST, M. MILGRAM, « Modelizing character allographs in omni-scriptor frame: a new non-supervised algorithm », Pattern Recognition Letters, vol. 21, #4, 2000, p. 295–300.

[Schomaker] L. SCHOMAKER, E.H. HELSPER, H.-L. TEULINGS, G.H. ABBINK, « Adaptative recognition of online, cursive handwriting », ICOHD, Vol. 1, 1993.

[Senior] A.W. SENIOR, A.J. ROBINSON, « An off-line cursive handwriting recogntion system », PAMI, Vol. 20, #3, 1998, p. 309-321.

[Taft] M. TAFT, *Reading and mental lexicon*, Erlbaum Edition, 1991.

[Vuori 99] V. Vuori, J. LAAKSONEN, E. OJA, « On-line adaptation in recognition of handwritten alphanumeric characters », ICDAR, Vol. 1, 1999, p. 792–795.

[Vuori 02] V. VUORI, J. LAAKSONEN, J. KANGAS, « Influence of erroneous learning samples on adaptation in on-line handwriting recognition », Pattern Recognition, Vol. 35, #4, 2002, p. 915–926.