# Correction de formulaires basée sur des machines pondérées à états finis

C. Mancas-Thillous, R. Beaufort

# Correction de formulaires basée sur des machines pondérées à états finis

Céline Mancas-Thillou[1] – Richard Beaufort[2]

[1] Faculté Polytechnique de Mons
Mons, Belgique

[2] Université Catholique de Louvain
Louvain-La-Neuve, Belgique

celine.thillou@fpms.ac.be, richard.beaufort@uclouvain.be

**Résumé** : *Pour être rendue plus démocratique, surtout dans les applications industrielles, la reconnaissance de caractères manuscrits a besoin d'atteindre de très hauts taux de reconnaissance. Pour cela, une correction dédiée à un problème particulier le permet de manière efficace, en modélisant l'information a priori disponible. Dans ce papier, nous présentons une méthode complète de compréhension de formulaires, avec une attention toute particulière sur une correction automatique à plusieurs niveaux. Basée sur une grammaire, définie en fonction des champs du formulaire à reconnaître, la correction utilise des machines à états finis modulaires, robustes et faciles d'implémentation. De plus, ces dernières permettent de relâcher le nombre de contraintes lié au remplissage du formulaire, rendant son utilisation plus facile. Finalement, des considérations pratiques sur la consommation mémoire et le temps de calcul pour un lexique de 40 000 entrées seront également énoncées.*

**Mots-clés** : Caractères manuscrits, Correction linguistique, Machines (pondérées) à états finis

## 1 Introduction

La reconnaissance des formulaires remplis de manière manuscrite a été un sujet intensivement étudié ces dix dernières années. Cependant, leur utilisation réelle et courante dans la vie de tous les jours reste cantonnée à des applications industrielles très ciblées, comme certains formulaires administratifs, les codes postaux sur les enveloppes, etc. Le point le plus crucial est d'obtenir une bonne efficacité de traitement sans intervention humaine. Plusieurs étapes dans la chaîne de traitement global influent sur les résultats. En premier lieu, la segmentation de l'information manuscrite et des cases du formulaire, la reconnaissance elle-même de l'information textuelle (soit par caractères isolés, soit par mots) et finalement la modélisation de l'information a priori pour la correction des erreurs de reconnaissance. Néanmoins, des demandes toujours plus exigeantes de l'industrie continuent d'exister, notamment pour des raisons de productivité ou de sécurité croissante, comme c'est le cas dans l'application décrite. Par conséquent, ce papier vise à décrire une méthode tout automatique en réduisant le nombre de contraintes lié au remplissage du formulaire. Design du formulaire, prétraitement, reconnaissance et correction en utilisant des automates à états finis sont les étapes détaillées dans cet article. La section 2 décrira l'état de l'art pour la compréhension et correction de formulaires et sera suivie dans la section 3 par l'explication succincte du prétraitement proposé, du recalage à la segmentation des caractères possibles. La section 4 détaillera nos outils de reconnaissance in situ avant d'expliquer dans la section 5 notre algorithme original de correction. Des résultats détaillés, en termes de taux de reconnaissance, consommation mémoire et temps de calcul pour une application industrielle réelle seront fournis dans la section 6. Enfin, la section 7 conclura ce papier.

## 2 Etat de l'art pour la Compréhension et Correction de Formulaires

La reconnaissance de caractères, et a fortiori manuscrits, est sujet à bon nombre d'erreurs et la correction des caractères mal-reconnus est une étape obligatoire, surtout pour les applications industrielles exigeantes. Une des améliorations en terme de reconnaissance est la multiplication et fusion des classificateurs de manière à réduire la dérive d'un seul, tel qu'élaboré dans Gunter et Bunke [GUN 04].

Dans la compréhension de formulaires, le contexte d'application est habituellement connu et de l'information a priori est exploitable. Les solutions commerciales traditionnelles utilisent principalement un lexique pour valider des mots/champs appartenant à un dictionnaire défini et demande à l'utilisateur final une validation si la confusion est trop grande ou si plusieurs solutions sont possibles. L'intégration dynamique de nouvelles données à actualiser n'est pas prévue, ce qui amène à des solutions génériques, moins efficaces que celles plus pointues, répondant à un besoin particulier.

Les autres solutions, habituellement employées lorsque des méthodes rapides sont nécessaires, comme la recherche dans un dictionnaire de très grande taille, utilisent les techniques probabilistes et l'analyse des n-grammes d'un mot. Ils sont classiquement résolus grâce aux modèles de Markov cachés (HMM) ou à la programmation dynamique, utilisée en premier lieu par Neuhoff [NEU 75] dans la correction orthographique. Loudon et al. [LOU 00] ont également employé l'algorithme de Viterbi pour de la correction. Zimmermann et Bunke [ZIM 04] ont, quant à eux, utilisé des modèles de langues bigrammes et trigrammes (LM) et des HMM pour

corriger des phrases manuscrites. Koerich et al. [KOE 04] ont proposé plus récemment un algorithme de décodage rapide à deux niveaux et basé sur des HMM pour simultanément reconnaître et corriger des mots manuscrits à large vocabulaire. Dû à l'absence de lexiques, des mots corrects peuvent alors être corrigés en mots hors-vocabulaire. De plus, avec cette méthode, lorsque de nombreuses erreurs sont présentes dans un mot, il est très difficile d'atteindre le mot souhaité à cause du large vocabulaire.

Pour utiliser un lexique et pouvoir corriger efficacement, Carbonnel et Anquetil [CAR 04] ont utilisé un algorithme à deux passes qui réduit itérativement la taille du lexique souhaité, comme une recherche par dichotomie.

Finalement, Li et Tan [LI 04] ont étudié l'impact de plusieurs modèles de langues pour améliorer la précision de reconnaissance de caractères chinois à différents niveaux : sur le caractère, sur le mot et sur le domaine de définition du mot (ex : médical, scolaire, naturel, etc...). Ils ont ainsi défini une méthode qui sélectionne un modèle de langue pertinent en fonction d'une tâche de reconnaissance plus précise.

En reconnaissance de tableaux ou formulaires, l'ontologie est souvent utilisée pour réduire la taille du lexique pour une langue donnée ou pour dynamiquement construire un lexique quand le domaine d'application est inconnu, comme dans Tubb et Embley [TUB 02]. Les machines à états finis peuvent alors intervenir dans ce contexte.

Sauf dans l'emploi simultané de HMM et d'un dictionnaire, l'étape de reconnaissance de caractères est souvent considérée comme une boîte noire où seule la meilleure sortie est disponible. Nous proposons dans ce papier, après la segmentation du formulaire et la reconnaissance de caractères, une correction qui utilise l'information sémantique d'un champ et plusieurs sorties dynamiques du classificateur, combinés grâce à des machines à états finis.

# 3 Architecture Générale

## 3.1 Recalage

Comme le design du formulaire est connu et préalablement numérisé par scanner, un recalage assez immédiat est appliqué, basé sur la présence d'une ligne épaisse autour du formulaire et la détection de quatre coins, tel que montré dans la figure 1. Les quatre coins sont détectés en utilisant des filtres directionnels (horizontal et vertical) et un noyau de filtre de type ligne. La localisation des coins est une caractéristique utilisée dans la transformation affine deux-à-deux appliquée pour le recalage, en utilisant un formulaire de référence. Deux coins pourraient suffire mais les quatre sont utilisés pour augmenter la robustesse en appliquant une interpolation pour contrer de légères distorsions ou transformations non-affines.

## 3.2 Segmentation et binarisation des cases

Une fois le recalage effectué, l'étape de segmentation des cases est immédiate car la position de chaque case est connue. L'intérieur des cases est segmenté comme caractères potentiels et une binarisation globale [OTS 79] est ensuite appliquée. Une binarisation plus fine à cette étape n'est pas primordiale car la taille des case est petite et cela mène à des

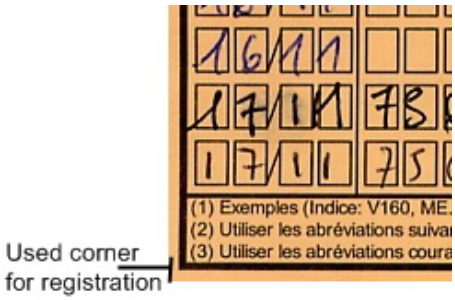


FIG. 1 – Extrait du formulaire en montrant la ligne épaisse de contour utilisée pour le recalage et un des quatre coins.

résultats satisfaisants, proche des résultats de binarisations locales plus complexes. En fait, aucune erreur n'est enregistrée due à une binarisation imprécise.

## 3.3 Suppression des contours des cases

Dus à de légères erreurs de recalage et surtout à la grande proximité des cases entre elles, le contour des cases peut être segmenté comme caractères potentiels. Par conséquent, une étape supplémentaire de suppression de lignes est nécessaire de manière à accroître le taux de reconnaissance. Pour cela, un filtrage directionnel de type ligne est une fois de plus utilisé, ce qui permet même d'enlever les contours des cases connectés aux caractères eux-mêmes.

## 3.4 Détection des cases vides

Cette étape est cruciale pour permettre la flexbilité dans le remplissage des formulaires. En effet, certains champs de formulaires sont de taille variable et peuvent être remplis de gauche à droite, de droite à gauche, en laissant des cases vides entre certaines données, etc. Donc la détection des cases vides est primordiale pour ensuite connaître la taille réelle du camp si besoin lors de la correction. Plusieurs méthodes peuvent être considérées. La première utilise le résultat de l'étape de reconnaissance (faible taux de confiance) et le nombre de composants connectés. Néanmoins, en présence de tâches, salissures, de fausses détections peuvent apparaître. La deuxième solution testée est l'utilisation d'un classificateur entraîné avec une classe supplémentaire dite 'vide' mais les résultats n'étaient pas encore optimaux. Finalement, nous avons opté pour une détection de texte plus robuste [MAN 07], expérimentée dans les scènes naturelles et basée sur plusieurs critères : la densité de texte, le seuil utilisé en binarisation globale, la direction des contours du supposé caractère et le nombre de composants connectés. Ces caractéristiques permettent ensuite l'entraînement d'un réseau de neurones pour une grande généricité et rapidité d'exécution.

# 4 Reconnaissance Isolée de Caractères

Pour effectuer la reconnaissance manuscrite, nous utilisons une version étendue de classificateur de Gosselin [GOS 96], basé sur des caractéristiques géométriques et un perceptron multi-couches (MLP) entraîné avec rétropropagation.

Pour reconnaître plusieurs variations d'un même caractère, les caractéristiques ont besoin d'être robustes contre le bruit, les distortions non-affines, la variation d'écriture inter et intra-scripteur, etc. Ainsi, pour être aussi invariant que possible, nos caractères à l'entrée du réseau de neurones ont une taille normalisée de $N \times N$ avec $N = 16$. Comme caractéristiques géométriques, nous utilisons le profil des contours des caractères. Une sonde est envoyée dans chaque direction (horizontal, vertical et diagonal) et pour réupérer l'information de trous comme pour la lettre 'B', des sondes partant de l'intérieur sont alors utilisées. De plus, une caractéristique supplémentaire est ajoutée : le ratio entre les hauteur et largeur initiales, avant normalisation pour facilement discriminer un 'i' d'un 'm'. Les explications des sondes sont montrées dans la figure 2.

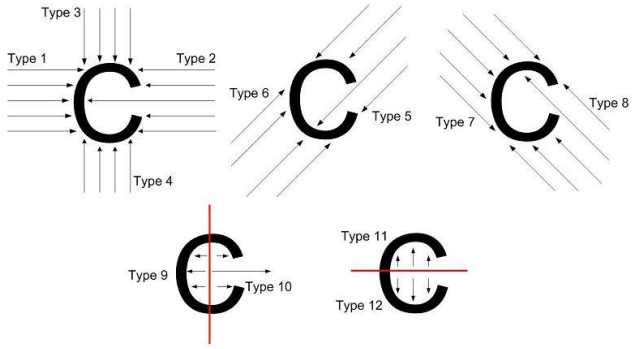


FIG. 2 – Calcul du profil des contours pour l'extraction de caractéristiques des caractères manuscrits.

Expérimentalement, pour obtenir de hauts taux de reconnaissance, nous complétons le vecteur de caractéristiques avec les moments de Tchebychev, qui sont des moments orthogonaux. Selon Mukundan et al. [MUK 01], nous utilisons les moments d'ordre 2 pour leur robustesse au bruit. Aucune sélection de caractéristiques n'est appliquée et l'ensemble des caractéristiques est un vecteur de 63 valeurs fourni au MLP contenant une couche cachée de 120 neurones et une couche de sortie de 34 classes pour les classificateurs les plus complets (chiffres arabes et lettres latines, avec fusion des '0' et 'O' et '1' et 'l'). Grâce à la correction basée sur une grammaire préalablement défini et présenté dans la section 5, notre classificateur a besoin d'être moins performant, ce qui permet un plus petit nombre de sorties mais a contrario, un nombre plus faible de confusions. Le nombre total du set d'entraînement est d'environ 56 000 caractères manuscrits, extraits sur des formulaires réalistes, remplis par plus d'une centaine de scripteurs, directement concernés par l'application, de manière à obtenir des données très représentatives, ponctuées de bruit et d'erreurs de remplissage réalistes. Plusieurs classificateurs ont été designés pour chacun des champs du formulaire, *i.e* pour la date, seulement un classificateur à 10 classes de sortie est nécessaire. Ainsi, la dimension de chaque classificateur est choisie avec soin pour accroître les taux de reconnaissance finaux.

# 5 Correction basée sur une Grammaire Dédiée

De manière à corriger les erreurs de reconnaissance, une correction qui est rapide, flexible et désignée pour chaque champ d'un formulaire, est nécessaire et présentée dans les prochaines sous-sections. Pour cela nous utilisons des machines à états finis pour modéliser l'information a priori. Cette approche a déjà montré d'excellents résultats en reconnaissance de caractères issus de scènes naturelles (Beaufort et Mancas-Thillou [BEA 07]) et en correction basée sur des caractères introduits par touche clavier (Beaufort [BEA 08]).

## 5.1 Correction à deux passes

Deux types différents de correction sont définis, à savoir passive et active, dépendant de l'information a priori disponible pour un champ du formulaire donné.

En effet, un champ d'un formulaire peut contenir une petite liste possible d'items pendant qu'a contrario un autre champ peut en comporter une très longue liste. Par conséquent, une première correction dite 'passive' est appliquée : nous considérons pour cela les 5 meilleures sorties du classificateur et leur combinaison pour un champ/mot donné, par l'analyse des $n$-meilleures chaînes possibles grâce aux machines à états finis. Ainsi toutes les séquences valides (existantes dans le lexique final) sont identifiées et triées par ordre de probabilité décroissant. En outre, avec une telle correction, si par exemple la seconde lettre d'un mot est 'g' et que le 'g' n'apparaît pas dans les 5 meilleures sorties, alors la correction est impossible (sauf à travers la liste de confusion, expliquée dans la sous-section 5.2). Par conséquent, cette première passe permet de valider les mots appartenant au lexique (sans correction nécessaire) ou pour corriger ceux qui ont une légère confusion avec les 5 meilleures sorties du reconnaisseur, ce qui est assez fréquent. En effet, très souvent lorsqu'un caractère est mal reconnu, la deuxième ou troisième sortie de l'OCR est souvent la bonne.
Une correction 'active' est ensuite appliquée pour les champs dont le lexique est de taille raisonnable ou qui présentent une forte discrimination entre chacun des items du lexique. Ainsi, dans ce cas, les machines à états finis aident à fournir toutes les chaînes possibles de caractère basées sur les 5 meilleures sorties du classificateur et une mesure de similarité basée sur la distance de Levenshtein est ensuite utilisée.

Chaque champ d'un formulaire a une information a priori propre et une correction dédiée à chaque champ est donc appliquée.

## 5.2 Concept des machines à états finis pour la correction

Les machines à états finis (FSMs), qui incluent des automates (FSAs), des transducteurs (FSTs) et leurs équivalents pondérés (WFSAs and WFSTs), peuvent être vus comme la définition d'une classe de graphes ou de langages.

De premier abord, un FSM est simplement un graphe orienté avec des labels sur chaque arc. Les transitions des automates sont quant à eux labellisées avec des symboles venant d'un unique alphabet $\Sigma$, pendant que celles des transducteurs sont labellisées avec des symboles d'entrée/sortie

appartenant à deux alphabets distincts $\Sigma_1$ et $\Sigma_2$. Les machines pondérées ajoutent des poids aux symboles sur les transitions.

De manière plus approfondie, les FSMs définissent des classes de langages réguliers. Dans cette définition, un automate est un *accepteur* : il représente l'ensemble des chaînes sur $\Sigma$ pour lequel il y a un chemin entre l'état initial et final d'un graphe. Un transducteur traduit alors les chaînes d'un premier langage sur $\Sigma_1$ en chaînes d'un second langage sur $\Sigma_2$ ; par conséquent, il est un lien entre les langages. Dans les machines pondérées, les poids, qui encodent les probabilités ou distances, sont accumulés tout au long des chemins pour définir le poids global d'une chaîne (dans les WFSAs) ou le poids global d'une transformation d'une chaîne d'entrée en une autre (dans les WFSTs).

Quelques propriétés fondamentales théoriques rendent les FSMs flexibles, puissants et efficaces. Parmi elles, la composition ($\circ$) qui est une généralisation de l'intersection des automates : partant d'un FST $T_1$ fonctionnant sur $\Sigma_1$ et $\Sigma_2$, et un FST $T_2$ sur $\Sigma_2$ et $\Sigma_3$, la composition calcule l'intersection sur $\Sigma_2$ et construit le FST $T_3$ sur $\Sigma_1$ et $\Sigma_3$. Cela permet ainsi de combiner différents niveaux de représentation simples pour construire des relations très complexes.

Le pseudo-code est illustré dans l'algorithme 1. Notre algorithme fonctionne sur une matrice $M$ contenant les 5 meilleures sorties issues de l'OCR pour chaque lettre d'un champ. Un WFSA dynamique est construit et stocké dans $W$, un pointeur sur un WFSA (ligne 1). Ce premier automate est pondéré pour laisser l'avantage à la meilleure sortie de l'OCR comparée aux 4 autres. Nous donnons (de manière expérimentale) un poids 3 fois plus grand pour la première sortie ($-\log_2(1)$) que le poids des autres ($-\log_2(0.33)$). Notons que ces poids sont exprimés en base logarithmique de manière à ensuite ajouter les poids plutôt que les multiplier dans les combinaisons des probabilités avec plusieurs transitions. $W$ est ensuite combiné, par composition, avec les $m$ (W)FSTs, qui représentent les différents modèles ($\mathcal{M}$) (lignes 4–6). Si le résultat $\rho$ de ces compositions n'est pas vide, le WFSA $B$ est créé (ligne 8) : il contient le meilleur chemin de $\rho$, *i.e* la forme la plus proche d'un item appartenant au lexique défini, étant données les modifications que nos modèles intermédiaires autorisent.

L'algorithme finit par construire, de $B$, la chaîne correspondant au champ complet (ligne 10).

**A propos des modèles $\mathcal{M}$.** Deux modèles sont utilisés : une liste de confusion et un lexique final.

– La liste de confusion contient des confusions pondérées comme 'i' avec 'l' ou '5' avec 's'. La liste et les poids associés ont été définis grâce aux erreurs propres à l'OCR que nous employons. De cet entraînement, la probabilité P$(l|i)$ de confondre un 'i' avec un 'l' est calculée comme suit :

$$\mathrm{P}(l|i) = \frac{\#(i \longmapsto l)}{\#(i)} \quad (1)$$

où $(i \longmapsto l)$ signifie la confusion de "$i$ avec $l$". Les erreurs qui n'apparaissent qu'une seule fois sont éliminées de la liste, ensuite compilée comme un WFST.

– Le dernier des modèles est et doit être le lexique puisqu'il permet l'acceptation finale d'une chaîne pour un langage ciblé. Jusqu'à maintenant, nos lexiques sont de simples FSAs, mais pourraient être des WFSAs si la probabilité d'apparition d'un item plutôt qu'un autre était définie. Pour cela, un très large corpus est nécessaire. Les lexiques dépendent de l'information a priori disponible et de la flexibilité voulue pour le remplissage d'un champ. Par exemple, lorsque très peu de flexibilité est permise, le lexique devient alors une longue liste éditable de tous les items possibles. En outre, pour d'autres champs, telle que la date, le lexique est codé avec des expressions régulières où la flexibilité est alors modélisée. Par exemple, le jour d'une date peut être écrit de 1 à 31 mais aussi de 01 à 31 et une corrélation avec le mois permet une correction plus fine (Avril a 30 jours et non 31). Par conséquent, toutes les règles pour définir une date et toutes les formes variées pour écrire une date peuvent être facilement supportées, de même que la modélisation des erreurs de reconnaissance (le jour '00' n'existe pas, etc). Cet exemple de modélisation est applicable à tous les champs de longueur variable et où des cases vides autour du champ ou à l'intérieur du champ peuvent être observées.

**Algorithme 1** Correction des FSMs

1: $W \leftarrow$ FSMVecCreate$(M)$
2: $B \leftarrow \emptyset$
3: $\rho \leftarrow W$
4: **for** $j = 0$ to $m$ **and** $\rho \neq \emptyset$ **do**
5: $\quad \rho \leftarrow \rho \circ \mathcal{M}[j]$
6: **end for**
7: **if** $\rho \neq \emptyset$ **then**
8: $\quad B \leftarrow$ FSMGetBestPath$(\rho)$
9: **end if**
10: **return** StringCreate$(B)$

Dans le cas de texte issu de scènes naturelles avec comme finalité un dictionnaire complet d'une langue donnée [BEA 07], d'autres modèles avaient été rajoutés comme un lié à la casse et un lié à la séparation des mots pour corriger ces erreurs après la reconnaissance de caractères.

## 5.3 Correction simultanée et fusion de données

Un aspect multi-niveaux a également été modélisé dans le cas de la compréhension de formulaires. Des champs peuvent avoir une information a priori commune et des exclusions mutuelles quand ils sont considérés ensemble plutôt que séparément. Par conséquent, nous appliquons au final plusieurs niveaux de correction. Nous avons présenté dans la sous-section 5.1 une correction à deux passes appliquée sur certains champs, qui est un niveau de correction indépendant des autres champs.

Néanmoins, pour modéliser la corrélation ou exclusion entre deux champs, une correction simultanée est possible et a été expérimentée dans notre application, qui concerne le routage de trains dans un pays et toutes les missions (étapes

du train) sont reportées sur un formulaire. Par exemple, le numéro et le type de train sont deux champs distincts. En premier lieu, une correction est appliquée sur chaque champ indépendamment : une correction passive seulement pour les numéros de trains, qui présentent une grande liste de numéros possibles et faiblement discriminants et une correction à deux passes pour le type de train. L'information a priori commune est ensuite modélisée, comme l'intersection des informations a priori indépendantes. Chaque train a un numéro avec seulement quelques types de train possibles attachés à ce numéro. Un automate est construit basé sur cette information modélisée avec des expressions régulières, une fois encore. Ce plus haut niveau de correction peut être fait sur différents champs et avec un nombre de champs croissant également. Cette correction simultanée permet de contraindre les erreurs et de s'assurer d'une solution valide pour plusieurs champs simultanément. En utilisant des machines à états finis, cette correction est supportée sans l'usage de larges et lourdes bases de données, pas facilement reconfigurables.

Néanmoins, la fusion de données devient primordiale après cette correction multi-niveaux. Les résultats peuvent être différents à des niveaux variés et le choix de la bonne solution doit être finalement pris. Le numéro de train corrigé indépendamment peut donner une solution $N_1$ et le type de train une solution $T_1$.

Pour une corrélation des deux champs, 4 cas sont possibles :

- Cas 1 : une solution confirmante $N_1 - T_1$
- Cas 2 : pas de solution ($N_1$ et $T_1$ sont incompatibles et aucune autre solution n'est possible)
- Cas 3 : une solution $N_1 - T_2$ ou inversement
- Cas 4 : une solution $N_2 - T_2$ totalement différente

Nous avons expérimentalement choisi de travailler avec les 5 meilleures sorties du reconnaisseur, ce qui est important pour avoir le plus de solutions valides possibles. Par conséquent, nous considérons que la corrélation à un plus haut niveau mène plus fréquemment à la meilleure réponse. Le cas 1 qui confirme les réponses indépendantes est le cas idéal et nous sommes sûrs de valider la correction (aucune erreur n'a d'ailleurs été enregistrée). Pour le cas 2, comme aucune solution commune n'est possible, nous considérons les meilleures corrections comme les correction indépendantes. En effet, une erreur peut être présente pour l'un des champs (*i.e* $T_1$) car la machine finale mène à un chemin valide en considérant les 5 meilleures sorties et la correction indépendante pour l'autre champ (*i.e* $N_1$) peut être juste. Dépendant de la qualité de la reconnaissance, de la qualité du remplissage du formulaire et du taux de discrimination entre les différents items d'un lexique, ce cas peut être fréquent. Pour les deux autres cas, nous calculons plusieurs chemins possibles après composition de tous les modèles de la machine finale. A l'instar de la considération des 5 meilleurs sorties pour l'OCR, nous considérons ensuite les 5 meilleurs chemins (si disponibles) pour chacune des corrections indépendantes. Par exemple, si le cas 3 avec une solution $N_1 - T_2$ révèle que le deuxième chemin valide pour la correction indépendante du type de train était $T_2$, alors le cas 3 est considéré comme valide. Si ce n'est le cas (dans les 5 meilleurs chemins pour chaque correction indépendante), la solution issue de la correction simultanée est validée néanmoins avec un faible taux de confiance, qui est ensuite possible de répercuter sur d'autres niveaux de compréhension du formulaire.

# 6 Résultats et Commentaires

Pour le calcul des résultats suivants, nous avons utilisé notre application de routage ferroviaire sur environ 1000 missions et 5 champs différents. La taille du dictionnaire de chaque champ est variable pouvant monter à plus de 39 000 mots.

Comme notre correction est basée sur les champs entiers ou même plusieurs champs simultanément, nous donnons comme information le taux de reconnaissance par caractère mais aussi par champ, ce qui est plus pertinent pour l'utilisabilité finale dans une application industrielle.

La détection des cases vides avec la technique décrite dans la sous-section 3.4 atteint un taux de bonne détection de 96% et un taux de faux positifs inférieur à 2%.

Le taux de reconnaissance de caractères calculé sur 3375 caractères était en moyenne pour tous les champs confondus sans correction de 74%. Avec une correction indépendante pour chaque champ, le taux atteint 85% et avec la correction multi-niveaux en utilisant les corrections croisées simultanées, il atteint une moyenne supérieure à 94%. Il est important de noter que les salissures, erreurs de remplissage, griffures, trous, etc ont bien été pris en compte dans ces résultats finaux.

Pour le taux de reconnaissance au niveau des champs entiers, si nous utilisons une correction indépendante passive basée sur l'information brute a priori, le taux moyen est de 47%. En outre, si nous utilisons la correction active et passive pour les champs qui le permettent, le taux atteint 53%. Finalement, si nous considérons notre correction multi-niveaux avec fusion de données, le taux dépasse 76%.

Pour le temps de calcul et l'occupation mémoire, il faut distinguer la taille des lexiques. Pour le plus petit automate, l'espace occupé est de 1 Ko, pour le champ 'date' avec une très large flexibilité de remplissage, on obtient alors un fichier de 59 Ko et pour le plus large automate avec plus de 39 000 entrées, la taille est de 407 Ko. Les FSMs qui utilisent une correction croisée ont une taille maximum de 3 Mo pour notre application. De plus, pour conclure, de la section 3 à la section 5, donc du recalage à la correction finale multi-niveaux, le temps de calcul est en moyenne inférieur à 1 seconde pour un formulaire comprenant de 15 à 20 missions. Ce résultat a été calculé sur un PC/Pentium Dual Core à 2 GHz avec 2 Go de RAM.

Les FSMs présentent donc de grands avantages ; parmi eux, la flexibilité, la modularité, la capacité d'actualisation, et un temps de calcul et une occupation mémoire faibles.

Ils permettent de modéliser toute information a priori, dès que cela peut être exprimé sous forme d'un language, ce qui est très souvent le cas. La flexibilité permet aussi de contourner les erreurs fréquentes dans la reconnaissance des caractères manuscrits, telle que la confusion pour les caractères graphiquement similaires.

En utilisant une composition créée à l'exécution du programme plutôt que compiler un modèle monolithique une

fois pour toutes (algorithme 1, lignes 4-6) permet de facilement supprimer ou ajouter un modèle dans le processus de composition et directement observer l'impact de la modification sur les résultats. La modularité est aussi vraie pour le processus global : au lieu de construire une chaîne de lettres à la fin (ligne 10), nous pouvons concaténer tous les automates pondérés de $W$, et composer la machine avec un modèle de langue (soit lexical, soit syntactique), donc représenté comme un automate pondéré complet.

La manière utilisée pour exprimer la grammaire liée à l'information a priori à travers les expressions régulières permet également d'ajouter un niveau d'actualisation souhaitable (même pour un non-expert) en vue d'une nouvelle version du logiciel. Un fichier texte sans lourdes bases de données est alors suffisant.

Comme exprimé dans les lignes précédentes de la section 6, les machines à états finis représentent des lexiques très larges avec une occupation mémoire faible. De plus, à l'exécution, il est possible de simplifier les résultats intermédiaires en utilisant diverses opérations [MOH 00] pour une efficacité encore plus accrue en termes de temps de calcul.

# 7 Conclusion

Dans ce papier, nous avons présenté une méthode complète pour la reconnaissance et correction de caractères manuscrits contenus dans des formulaires et testée sur une application industrielle réelle. La reconnaissance seule des formulaires sans correction intelligente n'atteint pas des taux de reconnaissance satisfaisants avec des techniques tout automatiques. L'étape additionnelle de correction d'erreur est donc obligatoire pour modéliser l'information a priori. Mais comment traduire cette information d'une manière modulaire et facile pour des utilisateurs non-experts, surtout dans le cas d'une actualisation nécessaire des données ? Ainsi, après avoir proposé un prétraitement pour la compréhension de formulaires et détaillé les outils de reconnaissance utilisés, nous avons présenté une correction à plusieurs niveaux, qui exploite les machines pondérées à états finis pour intégrer l'information grammaticale des champs du formulaire. La qualité de la reconnaissance de caractères manuscrits est primordiale dans les applications industrielles, mais l'impact est moindre lorsqu'est utilisé ce type de correction. Les applications deviennent alors moins contraignantes étendant ainsi leur démocratisation en tenant compte des impératifs terrain de l'industrie. L'extension de ce travail est maintenant d'affiner chacune des étapes dont notamment la reconnaissance en elle-même en utilisant des méthodes basées sur le traitement d'image ou la fusion de plusieurs classificateurs.

# 8 Remerciements

Les auteurs remercient Alain Ruelle pour son aide dans le design des machines à états finis pour l'application énoncée.

# Références

[BEA 07] R. Beaufort, C. Mancas-Thillou, "A weighted finite-state framework for correcting errors in natural scene OCR", *Proc. of IEEE/IAPR Int. Conf. on Document Analysis and Recognition*, 2007, pp. 889-893.

[BEA 08] R. Beaufort, "Application des Machines à Etats Finis en Synthèse de la Parole. Sélection d'unités non uniformes et Correction orthographique", *PhD Thesis, Facultés Universitaires Notre-Dame de la Paix*, 2008.

[CAR 04] S. Carbonnel, E. Anquetil, "Lexicon organization and string edit distance learning for lexical post-processing in handwriting recognition", *Proc. of the Int. Workshop on Frontiers in Handwriting Recognition*, 2004, pp. 462-467.

[GOS 96] B. Gosselin, "Application de réseaux de neurones artificiels à la reconnaissance automatique de caractères manuscrits", *PhD thesis, Faculté Polytechnique de Mons*, 1996.

[GUN 04] S. Gunter, H. Bunke, "Combination of three classifiers with different architectures for handwritten word recognition", *Proc. of the Int. Workshop on Frontiers in Handwriting Recognition*, 2004, pp. 63-68.

[KOE 04] A. Koerich, R. Sabourin, C. Suen, "Fast two-level HMM decoding algorithm for large vocabulary handwriting recognition", *Proc. of the Int. Workshop on Frontiers in Handwriting Recognition*, 2004, pp. 232-237.

[LI 04] Y-X. Li, C. L. Tan, "An empirical study of statistical language models for contextual post-processing of Chinese script recognition", *Proc. of the Int. Workshop on Frontiers in Handwriting Recognition*, 2004, pp. 257-262.

[LOU 00] G. Loudon, O. Pellijeff, L. Zhong-Wei, "A method for handwriting input and correction on smartphones", *Proc. of the Int. Workshop on Frontiers in Handwriting Recognition*, 2000, pp. 481-485.

[MAN 07] C. Mancas-Thillou, S. Ferreira, J. Demeyer, C. Minetti, B. Gosselin, "A multifunctional reading assistant for the visually impaired", *EURASIP Jour. on Image and Video Processing*, 2007.

[MOH 00] M. Mohri, F. Pereira, M. Riley, "The design principles of a weighted finite-state transducer library", *Theoretical Computer Science*, vol.231, n.1, 2000, pp. 17-32.

[MUK 01] R. Mukundan, S.H. Ong, P.A. Lee, "Discrete vs. continuous orthogonal moments in image analysis", *Proc. Int. Conf. On Imaging Systems, Science and Technology*, 2001, pp. 23-29.

[NEU 75] D. Neuhoff, "The Viterbi algorithm as an aid in text recognition", *IEEE Trans. Information Theory*, 1975, vol.21, pp. 222-226.

[OTS 79] N. Otsu, "A threshold selection method from gray level histograms", *IEEE Trans. System, Man and Cybernetics*, 1979, vol.9, n.1, pp. 62-66.

[TUB 02] K. Tubb, D. Embley, "Recognizing records from the extracted cells of microfilm tables", *Proc. of the ACM Symp. on document engineering*, 2002, pp. 149-156.

[ZIM 04] M. Zimmermann, H. Bunke, "N-gram language models for offline handwritten text recognition", *Proc. of the Int. Workshop on Frontiers in Handwriting Recognition*, 2004, pp. 203-208.