Introduction de mécanismes probabilistes dans un analyseur grammatical bidimensionnel pour la reconnaissance d'images de documents

André Oliveira Maroneze

UMR IRISA

Master Recherche en Informatique

Rapport de stage

Introduction de mécanismes probabilistes dans un analyseur grammatical bidimensionnel pour la reconnaissance d'images de documents

André Oliveira Maroneze

Encadrants : Bertrand Coüasnon et Aurélie Lemaitre

Juin 2010

Table des matières

Chapitre 1

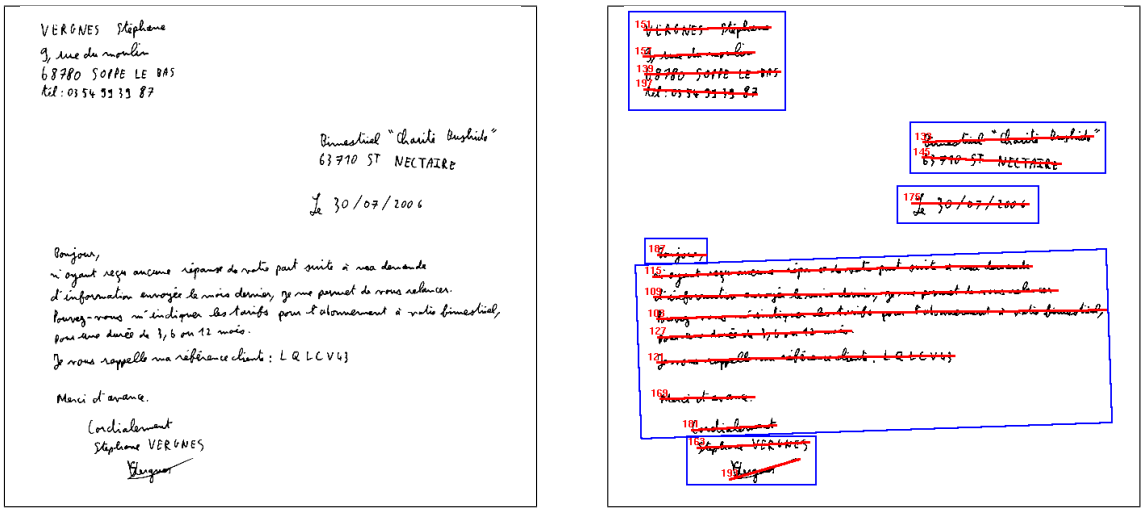
Introduction

L'analyse d'images de documents est le processus qui permet, à partir d'une image de document, d'obtenir une *interprétation* de cette image, c'est-à-dire, d'établir l'organisation et le contenu du document en question. Nous nous intéressons en particulier à l'analyse de la structure de documents : pour un document source donné (par exemple, le courrier de la figure 1.1a), nous cherchons à localiser les éléments significatifs. Le plus souvent, ces documents comprennent du texte, mais ils peuvent aussi représenter des diagrammes, des partitions de musique ou même des images quelconques contenant une structure reconnaissable. Un exemple d'analyse est indiqué dans la figure 1.1b : les rectangles indiquent des zones homogènes du document, auxquelles nous pouvons attacher des étiquettes telles que « expéditeur », « destinataire », « ouverture », « date », « message » et « signature ». Selon nos besoins, nous pouvons conserver tous les détails (jusqu'aux lignes, mots et caractères) ou ne garder qu'un résultat de plus haut niveau, comme c'est le cas dans la figure 1.1b.

Plusieurs méthodes de reconnaissance de la structure de documents existent, dont les modèles syntaxiques (aussi dits *grammaticaux*). L'analyse à base de ces modèles est, le plus souvent, effectuée de manière déterministe. Cependant, pour l'analyse de la structure de documents, l'absence d'une structuration unique (p. ex., dans la reconnaissance de documents manuscrits) et la présence de bruit génèrent des ambiguïtés qui ne peuvent pas être traitées correctement de façon déterministe. Par exemple, nous pouvons remarquer dans la figure 1.2 trois blocs de texte contenant des adresses suivis de deux règles grammaticales [1] pour essayer de les analyser correctement. Nous considérons, dans cette analyse, que seule la structuration des lignes (position, largeur, pente, etc.) nous est disponible, mais pas leur contenu. La première règle R_1 [2] réussit sur B_1, échoue sur B_2 et réussit à tort sur B_3 (la dernière ligne ne contient pas de code postal) ; avec R_2, nous avons la situation inverse : réussite sur B_2 et sur B_3 (en ne consommant pas la dernière ligne), mais réussite à tort sur B_1. Nous sommes donc amenés à insérer les deux règles dans notre grammaire, ce qui empêche une analyse déterministe : si R_1 est appliquée systématiquement avant R_2, il y a une erreur dans l'analyse de B_3 ; du contraire, c'est B_1 qui est analysé incorrectement.

L'incorporation de mécanismes probabilistes, tels que les grammaires stochastiques, permet de faire face à ces difficultés. Ainsi, dans l'exemple précédent, nous pourrions incorporer les résultats d'un classifieur qui indiquerait la probabilité d'avoir un code postal sur la troisième ligne, pour ensuite décider si R_1 ou R_2 doivent être appliquées, comme dans la figure 1.3. Nous supposons

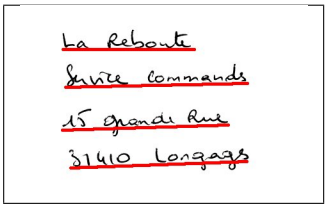
1. Ces règles sont dans la forme de Backus-Naur et sont basées sur la syntaxe d'EPF (décrite dans la section 3.1).
2. Cette règle peut être « lue » de la manière suivante : « les coordonnées sont formées par trois lignes d'adresse et, située sous ces lignes, une ligne de code postal ».



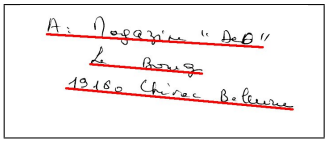
(a) Le document source : un courrier manuscrit.

(b) Résultat d'une analyse de la structure superposé au document. Les éléments significatifs (lignes et blocs de texte) sont mis en évidence (traits rouges et rectangles bleus).

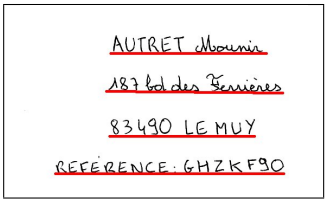
FIGURE 1.1: Exemple d'un document structuré, avec l'image source à gauche et le résultat d'une analyse à droite. Cette analyse reconnaît des lignes et les organise en blocs de texte.



(a) B_1



(b) B_2



(c) B_3

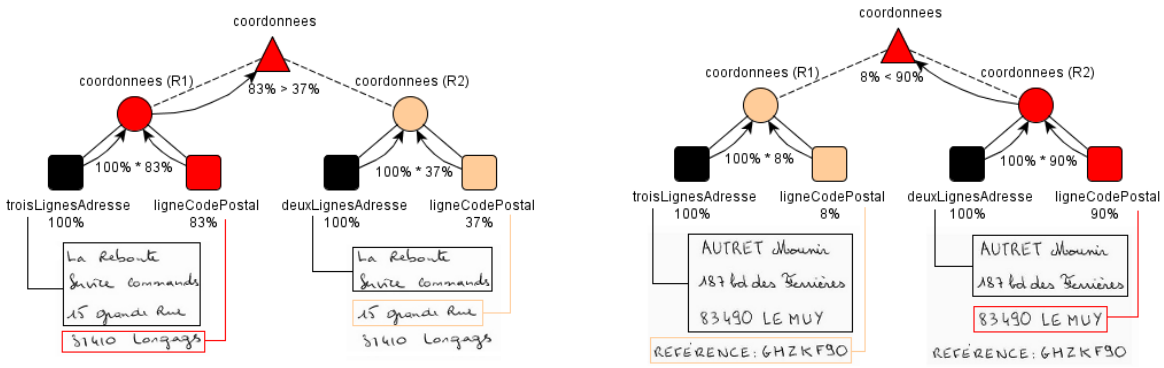
R_1 : `coordonnees ::= troisLignesAdresse && AT(sousLignes) && ligneCodePostal.`

R_2 : `coordonnees ::= deuxLignesAdresse && AT(sousLignes) && ligneCodePostal.`

FIGURE 1.2: Blocs de texte (B_1, B_2 et B_3) contenant des adresses et deux possibles règles pour les analyser (R_1 et R_2). Aucune de ces règles ne peut, individuellement, analyser correctement les trois blocs : R_1 échoue pour B_2 et réussit à tort sur B_3, tandis que R_2 réussit à tort sur B_1. Cela nous amène à insérer les deux règles dans la grammaire, ce qui peut générer des ambiguïtés.

ici que le classifieur utilisé pour la ligne code postal a une incertitude associée, mais aucune n'est associée aux branches `deuxLignesAdresse` et `troisLignesAdresse`. Les probabilités sont propagées de la façon suivante : pour les branches de type ET (cercle), la probabilité conjointe (multiplication) est utilisée, tandis que pour les branches de type OU (triangle), la branche avec la probabilité la plus élevée est choisie. Il est ainsi possible de choisir la meilleure règle face aux ambiguïtés.

Ce stage a pour objectif l'intégration d'un tel mécanisme au sein de DMOS-P (une méthode d'analyse de documents basée sur des formalismes grammaticaux), actuellement déterministe. Le but du stage est de permettre, avec un système d'analyse stochastique, l'incorporation des résultats



(a) Arbre d'analyse de B_1 : R_1 est préférée

(b) Arbre d'analyse de B_3 : R_2 est préférée

FIGURE 1.3: Résultat d'une analyse avec des connaissances sur le contenu : le classifieur associé à la ligne de code postal permet de choisir la meilleure règle face aux ambiguïtés.

des classifieurs (donc de la connaissance du contenu) au sein de l'analyse de la structure, sans passer par un mécanisme de binarisation (qui engendrerait une perte d'information). Avec cette intégration, le concepteur d'une grammaire pourra prendre en compte le contenu dans sa description des documents, ce qui augmentera le pouvoir expressif de la grammaire. Il pourra aussi incorporer d'autres aspects numériques (tels que des métriques définies manuellement : hauteur des lignes, distances entre mots, etc.) et les fusionner aux aspects syntaxiques de l'analyse grammaticale.

Plusieurs difficultés se posent dans le cadre de l'utilisation d'une grammaire stochastique pour l'analyse de documents : habituellement utilisées dans l'analyse syntaxique du langage naturel, les grammaires stochastiques sont le plus souvent associées à des probabilités statiques. Ici, nous avons des caractéristiques (variables selon l'instance analysée) qui amènent à l'utilisation de probabilités dynamiques. De plus, ces caractéristiques sont de différentes natures : les unes liées au bruit présent dans l'image, les autres liées aux variations de la structure. Il est nécessaire de combiner ces deux facteurs pour obtenir un bon résultat. En outre, le nombre d'options à explorer augmentant de manière exponentielle selon la taille du document (dû à la combinatoire entre les plusieurs règles ambiguës), il faut aussi considérer l'efficacité de l'analyse dans le processus : avant, il n'y avait qu'un chemin exploré pour l'obtention du résultat ; ici, tous les chemins sont analysés dans la recherche du meilleur.

Ce rapport est structuré de la façon suivante : nous commençons par une présentation de l'état de l'art dans le chapitre 2, qui introduit la problématique de l'analyse grammaticale de documents et les possibles avantages et difficultés de l'utilisation de grammaires stochastiques dans ce domaine. Le chapitre 3 détaille le contexte du stage : nous présentons la méthode DMOS-P, à laquelle nous voulons intégrer un mécanisme stochastique, en insistant sur les éléments qui nous seront utiles pour l'implémentation. Le chapitre 4 décrit les contributions à ce mécanisme : l'intégration de scores et d'opérateurs d'analyse stochastique, facilement utilisables par le concepteur d'une grammaire. Nous les présentons tant sous une optique utilisateur que du point de vue de l'implémentation. Enfin, le chapitre 5 présente la validation de cette mise en œuvre avec deux contextes d'application distincts : l'analyse de 1250 courriers manuscrits ainsi que des pages de documents d'archives. Nous montrerons que ces travaux permettent une amélioration du taux de reconnaissance et une simplification de la grammaire, tout en gardant la compatibilité avec le mécanisme existant.

Chapitre 2

État de l'art

L'analyse de la structure de documents peut s'effectuer à l'aide de plusieurs techniques, parmi lesquelles nous rencontrons les méthodes grammaticales (partie 2.1). Un état de l'art de ces techniques nous permettra de comprendre leurs capacités et surtout leurs limitations actuelles pour l'analyse de documents ; une solution proposée par plusieurs auteurs consiste dans l'utilisation de grammaires stochastiques (partie 2.2), qui prennent en compte l'incertitude et l'ambiguïté présentes dans le domaine. Les possibilités et les difficultés de l'application de ces grammaires dans l'analyse de documents nous permettront d'établir un bilan (partie 2.3) qui sera utile par la suite du stage.

2.1 L'analyse grammaticale de documents structurés

L'analyse d'images de documents est une tâche complexe avec plusieurs niveaux d'abstraction ; le tableau 2.1 indique une division possible en cinq niveaux d'abstraction, du bas vers le haut [Nag00].

Il n'est pas possible de séparer ces tâches d'une manière précise : plusieurs tâches sont effectuées entre ces deux niveaux, par exemple la segmentation de mots à partir de composantes connexes, via des primitives, ainsi que de la structure. Néanmoins, le tableau donne un aperçu qui permet

Niveau d'abstraction (du bas vers le haut)	Tâches associées
Pixel	Binarisation, réduction de bruit, détection de pente, segmentation de caractères
Primitive	Reconnaissance de glyphes, composantes connexes, segments de droite et courbes
Structure	Segmentation de mots, reconstruction de lignes, analyse de tableaux, analyse syntaxique et sémantique
Document	Séparation entre texte et non-texte, analyse de composantes physiques et logiques, analyse de la mise en page
Corpus	Extraction de l'information, classification et indexation, recherche, validation

TABLE 2.1: Schéma de l'analyse de documents structurées, divisée en cinq niveaux d'abstraction avec les tâches liées à chaque niveau. Schéma basé sur celui de Nagy [Nag00].

d'établir, globalement, le niveau d'abstraction considéré par la suite de ce rapport : nous nous situons entre les niveaux structure et document, en supposant que les tâches dont ils dépendent (binarisation, détection de composantes connexes, segments de droite, etc.) sont déjà effectuées.

Dans cette section, nous présentons d'abord les aspects généraux de l'analyse de documents (partie 2.1.1), avec les différentes classifications et méthodes d'analyse couramment utilisées. Dans la partie 2.1.2 nous introduisons les grammaires multidimensionnelles, qui sont très intéressantes pour l'analyse de documents, notamment pour les structures plus complexes. Nous finissons par la problématique de l'approche syntaxique, en expliquant ses limitations dans la partie 2.1.3.

2.1.1 Classification des types et méthodes d'analyse

Quelques auteurs [MRK03, Tru05] distinguent l'analyse physique de l'analyse logique des documents : les colonnes, paragraphes, lignes, tableaux, etc. sont considérés des éléments physiques, tandis que les titres, sections, signatures, destinataires, etc. constituent des éléments logiques. Dans cet état de l'art nous considérons l'analyse de documents sans cette distinction.

Parmi les différentes classifications des méthodes d'analyse de documents [MRK03, Tru05], nous retiendrons trois classes : les méthodes à base de règles, les méthodes syntaxiques et les méthodes stochastiques. Cette division comprend la grande majorité des méthodes existantes et nous permet de bien les distinguer par rapport à nos besoins. Par la suite, nous considérons brièvement les méthodes à base de règles pour justifier leur non-utilisation dans une analyse de documents générique ; nous nous concentrerons après sur les méthodes syntaxiques, où l'étude de sa problématique nous amènera à y intégrer les méthodes stochastiques (section 2.2).

Les méthodes à base de règles disposent, comme leur nom l'indique, d'un ensemble de règles de reconnaissance ou transformation [MRK03, Tru05]. Ces règles, définies de manière arbitraire, sont associées le plus souvent à une base de connaissances qui évolue au fur et à mesure que l'analyse du document progresse. Elles sont proches des méthodes syntaxiques, mais diffèrent de celles-ci notamment par l'absence de règles d'inférence aussi restreintes que celles d'un analyseur syntaxique. Cette liberté leur confère un pouvoir expressif élevé mais constitue aussi un inconvénient : les règles peuvent devenir assez arbitraires [MRK03], *ad hoc* et indépendantes les unes des autres, et donc difficiles à généraliser ou à maîtriser d'une manière plus formelle, raison pour laquelle ces méthodes ne seront pas considérées par la suite.

Les méthodes syntaxiques [CD04, Coü01] s'appuient sur les grammaires de Chomsky et traitent le document comme une séquence (pas forcément monodimensionnelle) d'éléments. Ces méthodes permettent d'exprimer les connaissances de la structure du document à l'aide d'un formalisme intuitif [NPH04]. La composition arborescente des motifs présents dans les documents à partir de sous-motifs plus simples ressemble à la composition de mots à partir de lettres, et de phrases à partir de mots [FA77]. Les similarités suggèrent l'usage de ces méthodes pour l'analyse de documents.

Enfin, les méthodes stochastiques utilisent entre autres les HMMs (*Hidden Markov Model*s, modèles de Markov cachés) pour compenser l'incertitude et l'ambiguïté présentes dans la plupart des analyses [MRK03] et obtenir ainsi un résultat qui soit globalement plus fiable. Ces méthodes peuvent aussi tirer profit de données statistiques [Tru05], si elles sont disponibles, pour diriger l'analyse.

Les méthodes syntaxiques et les méthodes stochastiques ne sont pas mutuellement exclusives ; nous verrons plus tard que la problématique de l'usage des techniques à base de grammaires nous incitera à y intégrer des mécanismes stochastiques. Nous passons maintenant à l'analyse de méthodes syntaxiques particulièrement utiles pour l'analyse de documents.

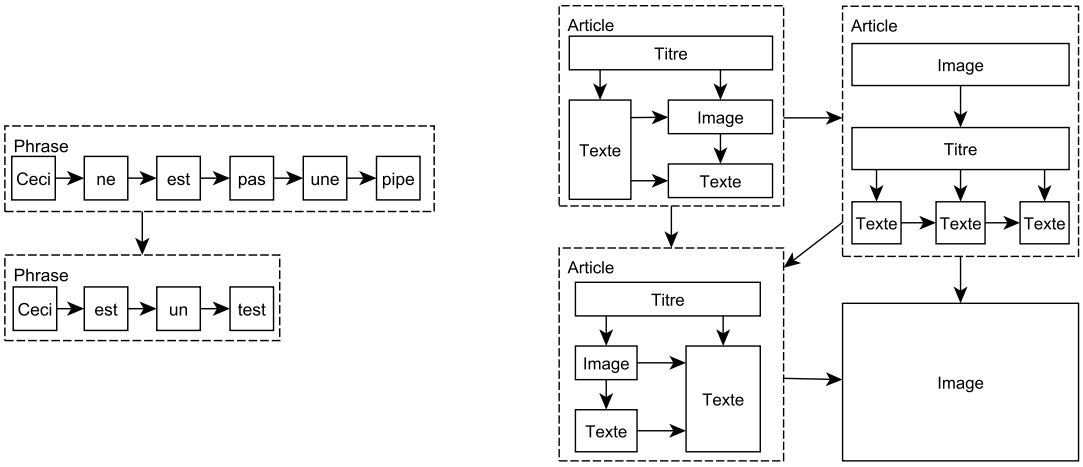


FIGURE 2.1: Les grammaires à chaînes ne possèdent qu'un type de relation : élément suivant (représentée par les flèches). Dans l'analyse de documents, il est souvent difficile de définir un seul élément suivant (par exemple, dans la structure d'une page de quotidien à droite, nous avons multiples possibilités de choix de l'élément suivant).

2.1.2 Grammaires multidimensionnelles

L'analyse syntaxique traditionnelle est effectuée sur des chaînes de caractères, où chaque élément ne possède qu'un prédécesseur et un successeur, ce qui les empêche de représenter convenablement des structures bidimensionnelles [NPH04], comme le montre l'exemple de la figure 2.1. Cette limitation n'est pas adaptée pour l'analyse de documents et peut forcer l'introduction de difficultés artificielles pour la définition de la grammaire.

Pour cette raison, d'autres types de grammaires ont été étudiées et sont couramment utilisées dans l'analyse de documents : des grammaires multidimensionnelles telles qu'EPF [Coü01] et des grammaires d'ordre supérieur, présentées dans [CD04]. L'étude de ces grammaires nous permettra de voir leurs avantages et limitations dans le cadre de l'analyse de documents.

Conway [Con93] propose une *grammaire de pages* pour la reconnaissance de documents. Cette grammaire possède la particularité de définir, via des opérateurs de position (tels que `above`, `over` et `leftof`), les relations entre les éléments constituant les terminaux (blocs de page, définis comme des groupes de segments voisins) et les non-terminaux (titre, colonne, paragraphe, etc.) de la grammaire. Ces opérateurs relient les coordonnées des éléments (définies via leurs rectangles englobants) à leurs voisins. Chaque élément possède un nombre variable de voisins et la grammaire exprime de manière naturelle le positionnement entre les blocs de base (figure 2.2).

Cette expressivité a naturellement un coût. Le *parsing* du document ne peut plus se faire de la manière traditionnelle : l'existence d'une deuxième dimension implique une croissance exponentielle du nombre de combinaisons d'éléments à considérer. Pour compenser cette perte d'efficacité, un analyseur tabulaire (*chart parser*) est utilisé. Cet analyseur conserve une table de sous-chaînes reconnues et un agenda d'éléments qu'il reste à traiter. Ces deux structures se combinent pour produire le prochain élément de l'analyse et éviter de refaire un même calcul. L'auteur mentionne que le coût de l'analyse reste assez élevé, même si la structure presque linéaire des documents

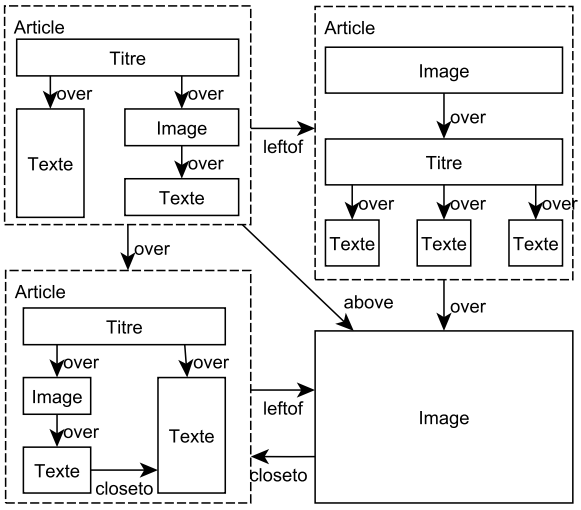


FIGURE 2.2: Exemple de relations d'adjacence entre les éléments du document de la figure 2.1 en utilisant une grammaire multidimensionnelle.

textuels analysés évite que le pire cas théorique arrive.

Nous pouvons considérer les grammaires de chaînes comme un cas particulier d'un type plus générique de grammaire : les grammaires de graphes [CD04]. Ces grammaires autorisent l'existence d'un nombre quelconque de voisins pour un élément donné, et l'analyse avance par substitution de sous-graphes selon les règles de la grammaire. Ce grand pouvoir expressif apporte un coût important à l'analyse : il faut résoudre un problème d'isomorphisme de sous-graphes pour appliquer les règles de la grammaire, ce qui est beaucoup trop coûteux pour être effectué de manière efficace. Sauf dans quelques cas particuliers (où les graphes sont restreints à des arbres, par exemple), ce type de grammaire n'a pas rencontré beaucoup de succès.

Malgré leur plus grande expressivité, nous verrons que les grammaires multidimensionnelles possèdent encore des limitations pour l'analyse de documents.

2.1.3 Problématique liée à l'analyse grammaticale de documents

Les méthodes syntaxiques, destinées d'abord aux problèmes liés aux langages de chaînes, présentent des difficultés particulières dans le domaine de l'analyse de documents. Plusieurs auteurs [MRK03, CD04, Tru05, NPH04] mentionnent les limitations des analyses déterministes pour la gestion des ambiguïtés et du bruit. En effet, dans l'analyse grammaticale traditionnelle, les éléments de base sont des symboles (le plus souvent des caractères) bien définis sans aucune incertitude associée. Ce n'est pas le cas dans l'analyse de documents, où le plus souvent la source de données est une image, et les éléments terminaux en sont extraits par des classifieurs ou des procédés particuliers qui associent une incertitude aux lexèmes. À cette incertitude il faut ajouter la nécessité de traiter des structures de documents ambiguës : les documents complexes peuvent contenir des erreurs ou des différences de structure (comme celles indiquées précédemment dans la figure 1.2) qui sont facilement perçues et correctement analysées par un humain, mais qui exigent un traitement robuste

et complexe par un ordinateur.

Jurafsky [JM00] présente les grammaires stochastiques comme une solution pour le traitement des ambiguïtés ; lui et d'autres auteurs indiquent que ces grammaires apportent souvent de meilleurs résultats que les méthodes déterministes [JWS+95, Min98] dans des domaines tels que la reconnaissance de la parole. En suivant les recommandations des auteurs des divers états de l'art dans l'analyse de documents [MRK03, NPH04], nous nous penchons vers ce type de grammaire pour chercher des solutions qui peuvent s'incorporer à l'analyse syntaxique de documents.

2.2 Grammaires stochastiques

Les grammaires stochastiques (aussi appelées *grammaires probabilistes*, ou encore *grammaires statistiques*) sont des grammaires où chaque règle de production a une probabilité associée. Formellement [Hav08], ce sont des quintuplets $G =< V_T, V_N, P, W, S >$, où V_T est l'alphabet (ensemble de terminaux), V_N est l'ensemble de non-terminaux, P est l'ensemble de règles de transition (ou dérivation), W est l'ensemble de probabilités de transition associées à P et S est l'ensemble d'états initiaux. La probabilité d'une chaîne (aussi dite *séquence* ou *phrase*) est le produit des probabilités des règles de dérivation qui génèrent cette chaîne ; si plusieurs chemins de dérivation peuvent générer une même chaîne, alors sa probabilité est la somme des probabilités de chaque chemin. La figure 2.3 contient un exemple de grammaire stochastique très simple. Dans cet exemple, nous avons :

- V_T = {lignesCorps, troisLignesAdresse, deuxLignesAdresse, ligneCodePostal, ligneReference}
- V_N = {PageCourrier, Coordonnees}
- S = {PageCourrier}
- P et W sont représentés dans la figure 2.3, où les probabilités de W correspondent aux valeurs dans $\rightarrow_p$.

PageCourrier $\rightarrow_1$ Coordonnees, lignesCorps
Coordonnees $\rightarrow_{0,69}$ deuxLignesAdresse, ligneCodePostal
Coordonnees $\rightarrow_{0,31}$ troisLignesAdresse, ligneCodePostal

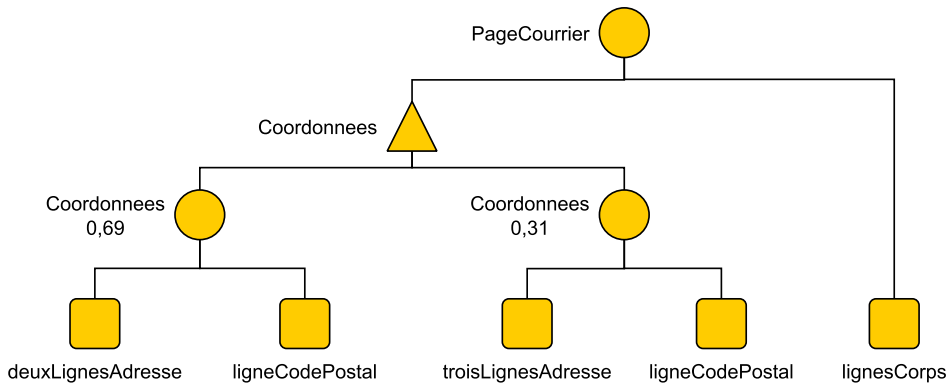


FIGURE 2.3: Exemple d'une grammaire stochastique : les non-terminaux commencent par des majuscules et les terminaux par des minuscules. Le non-terminal `Coordonnees` possède deux règles de dérivation avec probabilités 0,69 et 0,31. Les probabilités non affichées sont égales à 1.

Cette grammaire correspond à celle qui a produit les arbres d'analyse de la figure 1.3, avec les poids des règles (83% et 37%, propagés des feuilles vers les nœuds) normalisés pour correspondre à des probabilités.

Les grammaires stochastiques sont particulièrement adaptées pour traiter les questions d'ambiguïté structurelle. Ces ambiguïtés, bien étudiées dans le cas du langage naturel [JM00], sont aussi présentes dans l'analyse de documents, d'où la recommandation [MRK03] de l'application de ces grammaires dans ce domaine.

Nous définirons d'abord les *grammaires avec poids* (*weighted grammars*), une généralisation des grammaires stochastiques, pour situer celles-ci par rapport aux premières (partie 2.2.1) ; ensuite, nous établirons une relation entre les grammaires stochastiques régulières et les HMMs (*Hidden Markov Models*, modèles de Markov cachés) dans la partie 2.2.2, pour ensuite introduire la notion d'analyse syntaxique stochastique (partie 2.2.3). Nous présenterons après un aspect important lié à l'usage de ces grammaires dans le contexte de l'analyse de documents : l'attribution des probabilités de dérivation (partie 2.2.4), liée à l'intégration du contenu dans l'analyse structurelle.

2.2.1 Grammaires avec poids

Les grammaires avec poids généralisent les grammaires stochastiques : les poids associés aux règles peuvent avoir une valeur quelconque entre 0 et $+\infty$, et la somme des poids d'un non-terminal n'est pas contrainte à 1. Cependant, dans le cas d'usage le plus fréquent (PCFGs, grammaires hors-contexte probabilistes), cette généralisation n'augmente pas leur pouvoir expressif [SJ07]. En pratique, les deux types ont des contextes d'usage très similaires et une normalisation des poids permet en général la conversion vers une grammaire probabiliste.

2.2.2 Grammaires stochastiques régulières et HMMs

Un modèle de Markov caché (HMM) est, formellement [Hav08], un quintuplet $< E, K, \Pi, A, B >$, où E est un ensemble d'états, K un ensemble de symboles de sortie, Π un ensemble de probabilités initiales, A un ensemble de probabilités de transition (liées aux états dans E) et B un ensemble de probabilités d'émission de symboles. Il est équivalent à une grammaire stochastique régulière, c'est-à-dire, un automate à états finis avec des probabilités associées aux transitions et aux états (qui conditionnent l'émission de symboles de K). Ces probabilités ne dépendent que du non-terminal à être dérivé, c'est-à-dire, de l'état courant de l'automate, ce qui assure l'*invariance temporelle* caractéristique des processus markoviens. Une discussion formelle avec des exemples est disponible dans [Hav08]. Nous nous intéressons ici aux opérations sur les HMMs pour les intégrer à l'analyse syntaxique par la suite.

Les HMMs sont principalement utilisés dans l'un des trois contextes suivants [NPH04, Hav08] :

- *évaluation* : étant donné un HMM avec paramètres (A, B, Π) connus, et étant donné une séquence d'observations (une chaîne de terminaux), déterminer la probabilité d'occurrence de cette chaîne ;
- *décodage* : étant donné une séquence d'observations, trouver la séquence d'états la plus probable qui aurait pu avoir généré cette séquence ;
- *apprentissage* : étant donné un ensemble de séquences d'observations, trouver les paramètres du modèle qui maximisent la probabilité de générer cet ensemble.

Chacune de ces opérations est effectuée à l'aide d'algorithmes particuliers, qui possèdent des versions correspondantes pour d'autres types de grammaires (hors-contexte, attribuées, etc.). Les grammaires

hors-contexte probabilistes (PCFGs, *Probabilistic Context-Free Grammars*), en particulier, sont le modèle le plus simple pour les structures arborescentes [Hav08]. Nous pouvons étendre les opérations définies pour les HMMs de manière à obtenir des correspondances pour les grammaires hors-contexte [Hav08]. Ces extensions, à l'aide d'algorithmes spécifiques, fournissent les bases pour l'analyse syntaxique stochastique. Nous utiliserons donc la même terminologie vue dans le cadre des grammaires régulières pour les opérations sur tous les types de grammaires.

2.2.3 Analyse syntaxique stochastique

Parmi les opérations citées précédemment pour les HMMs, le *décodage* est celle qui correspond, dans le cas général, à l'analyse syntaxique. Ce décodage peut être effectué à l'aide de l'algorithme de Viterbi (décrit dans [Hav08]), qui utilise une méthode de programmation dynamique pour trouver efficacement, à partir d'une séquence donnée, l'arbre d'analyse *le plus probable*. L'existence d'ambiguïtés est transparente à ce *parsing* : même si plusieurs arbres de dérivation existent, celui qui a la probabilité la plus élevée est obtenu en premier. Ceci implique une notion de *préférence* lors de l'analyse syntaxique ; cette préférence est explicitée via les probabilités de dérivation. Nous verrons donc comment attribuer les probabilités aux règles de dérivation de la grammaire ; la définition de ces probabilités varie selon le domaine d'application et la présence de caractéristiques et incertitudes dans les éléments terminaux.

2.2.4 Attribution des probabilités de dérivation

L'une des difficultés d'utilisation d'une grammaire stochastique est l'attribution des probabilités de transition. Il y a, de manière générale, deux facteurs qui peuvent influencer cette probabilité : des connaissances *a priori* (liées à l'apprentissage statistique) et les attributs déterminés *a posteriori* (spécifiques à chaque instance, tels que l'incertitude et les caractéristiques associées aux terminaux, qui peuvent correspondre à la connaissance du contenu). L'importance de chacun de ces facteurs varie selon le domaine d'application de la grammaire.

Pour la résolution d'ambiguïtés, les connaissances *a priori* permettent de donner la préférence à l'arbre d'analyse le plus probable ; dans l'absence d'incertitude pour les éléments terminaux, ces connaissances seules peuvent suffire pour déterminer les probabilités. Par exemple, dans l'analyse syntaxique du langage naturel, ce sont le plus souvent les *treebanks* [Hav08] (arbres d'analyse annotés) qui fournissent les données pour l'attribution des probabilités. Dans leur absence, les algorithmes liés à l'opération d'*apprentissage* citée dans 2.2.2 permettent l'obtention de ces probabilités, avec potentiellement l'ajout d'informations contextuelles [JM00].

Dans la reconnaissance de motifs, les connaissances *a priori* ne jouent pas toujours un rôle très important. Plusieurs auteurs dans le domaine qui ont utilisé les grammaires stochastiques [HNZ05, IB00, MRK03] ont incorporé des caractéristiques quantitatives (telles que la hauteur des caractères et des lignes) à leurs éléments terminaux pour guider l'analyse ; cela revient à intégrer de manière indirecte le contenu à l'analyse structurelle. D'autres auteurs [Art03, FGK06, TI94] ont incorporé des fonctions/coûts sur les branches (par exemple, l'angle et la distance entre deux mots dans une règle qui les concatène) pour prendre en compte la variabilité des images.

Dans tous les cas, l'incorporation de caractéristiques quantitatives amène à l'utilisation de grammaires attribuées à la place de simples PCFGs. Ces grammaires, soit à l'aide de mécanismes d'unification (comme les DCGs - *Definite Clause Grammars*, grammaires logiques basées sur les CFGs), soit à l'aide de contraintes (comme la grammaire floue de Fitzgerald et coll. [FGK06]), permettent

la prise en compte des caractéristiques, tout en étant compatibles avec un mécanisme d'analyse stochastique [Hav08] : les probabilités de dérivation sont implicitement ou explicitement incorporées dans les attributs, avec optionnellement de la connaissance a priori obtenue à partir d'ensembles d'entraînement. Ces deux sources d'information guident l'analyse de façon à générer le meilleur résultat face à l'ambiguïté.

2.3 Bilan de l'existant

L'étude des solutions existantes en termes de grammaires stochastiques pour l'analyse de documents nous amène à nous concentrer sur les grammaires attribuées pour la prise en compte des caractéristiques des éléments terminaux (notamment des attributs quantitatifs, tels que l'incertitude), ainsi que pour apporter des informations de contexte.

Une grammaire hors-contexte bidimensionnelle est plus adaptée pour traiter une grande variabilité dans les types de documents : simples ou complexes, plats ou avec plusieurs niveaux d'imbrication, linéaires ou non. Le coût en complexité de ces grammaires, par contre, peut se révéler assez élevé dû à l'explosion combinatoire de l'augmentation du nombre de dimensions d'analyse [Con93, NPH04] et doit être toujours pris en compte. Les solutions actuelles à cette question [Art03, HNZ05, TI94] sont assez *ad hoc*, spécifiques aux domaines d'application où elles ont été implémentées.

L'incorporation de probabilités dans les méthodes syntaxiques d'analyse de documents est indiquée comme nécessaire pour une bonne qualité de l'analyse [MRK03, NPH04], compte tenu du bruit et des ambiguïtés présents dans les images de documents. Cependant, comme nous l'avons vu, le passage d'un mécanisme déterministe à un probabiliste n'est pas toujours facile : en plus des questions d'attribution et propagation des probabilités, nous avons des coûts d'analyse plus importants, compte tenu du fait d'explorer plusieurs solutions pour trouver la meilleure.

Pour une application étendue dans le domaine de l'analyse de documents, les grammaires stochastiques doivent gérer les informations de contexte (via des attributs) et avoir un bon niveau d'expressivité (ce qui exclut les grammaires régulières, donc les HMMs) ; l'entraînement peut aider à la définition des probabilités pour les règles, mais nous avons toujours besoin de considérer l'incertitude des terminaux, car nous cherchons à intégrer l'information provenant du contenu dans l'analyse structurelle. Pour simplifier l'écriture, nous utilisons des grammaires ambiguës, ce qui nous oblige à ajouter des informations (numériques ou symboliques) pour avoir un critère de comparaison entre plusieurs solutions ambiguës. Ce critère définit la qualité de la solution et peut aussi aider dans l'élagage de solutions, s'il peut être utilisé en tant qu'heuristique ; il peut être provenant des classifieurs, de l'entraînement statistique, ou des deux combinés.

Tous ces aspects ont été considérés pour l'introduction du mécanisme stochastique au sein de la méthode existante DMOS-P. Nous commençons le chapitre suivant par une analyse du contexte spécifique au sujet de stage. Cette analyse a permis de répertorier les besoins spécifiques, en prenant compte des contraintes d'implémentation, pour la mise en œuvre proprement dite, présentée dans le chapitre 4. Enfin, la validation du mécanisme, avec des résultats chiffrées, est présentée dans le chapitre 5.

Chapitre 3

Contexte d'implémentation : la méthode DMOS-P

Pour effectuer l'analyse de la structure de documents, nous disposons de la méthode DMOS-P (présentée dans la partie 3.1). Cette méthode, actuellement basé sur un mécanisme déterministe, est implémentée avec le langage λ-Prolog (brièvement présenté dans la partie 3.2) ; l'implémentation existante de DMOS-P utilise le mécanisme déterministe fourni par ce langage, mais nous verrons que celui-ci possède aussi des outils intéressants pour l'intégration d'un analyseur stochastique.

3.1 Présentation de DMOS-P et EPF

DMOS-P [Coü01, LC08, LCC10] (*Description and MOdification of Segmentation with Perceptive vision*) est une méthode générique de reconnaissance de documents basée sur un formalisme grammatical (EPF, *Enhanced Position Formalism*) et disposant d'un analyseur associé. Cette méthode permet l'analyse de la structure de plusieurs types de documents avec des modifications minimales du système ; ce n'est que la connaissance *a priori* (la grammaire de description d'une classe de documents) qui doit être modifiée, tout le reste du système pouvant être réutilisé. La figure 3.1 présente un schéma simplifié de la méthode DMOS-P (inspiré de celui dans [LC08]).

Nous présentons d'abord le formalisme EPF, avec un exemple de grammaire (partie 3.1.1) ; ensuite, nous détaillons les deux aspects qui seront impactés par l'introduction du mécanisme stochastique : la compilation avec EPF (partie 3.1.2) et l'intégration de classifieurs, avec leur incertitude associée sous la forme de scores (partie 3.1.3).

3.1.1 Une grammaire bidimensionnelle : le formalisme EPF

L'analyse de documents avec EPF est effectuée à un niveau relativement élevé (aux environs du niveau « Structure » dans le tableau 2.1), grâce aux fonctionnalités d'analyse de l'image disponibles dans DMOS-P. Le formalisme grammatical EPF utilise comme éléments terminaux des segments de droite et des composantes connexes (ensembles contigus de pixels noirs), comme ceux indiqués dans la figure 3.2, ce qui le rend capable de reconnaître documents de plusieurs types (partitions musicales, expressions arithmétiques, formulaires, etc.). Un mécanisme de vision perceptive [LC08] permet, pour les documents textuels, l'obtention de lignes de texte à partir de composantes connexes et segments de droite ; d'un point de vue d'utilisation, ce sont donc des éléments terminaux aussi.

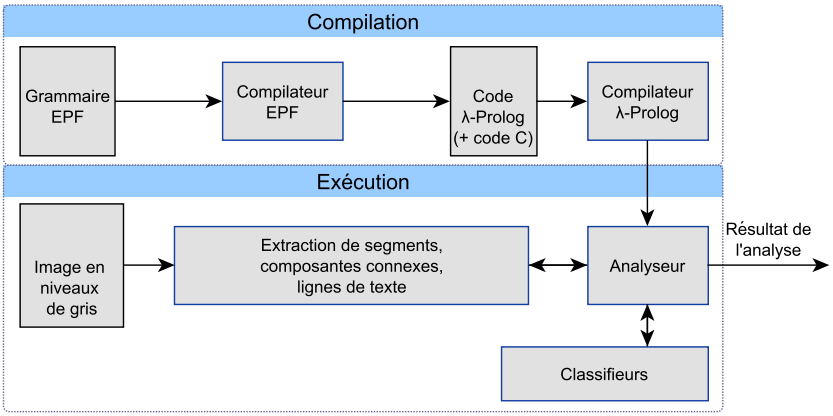
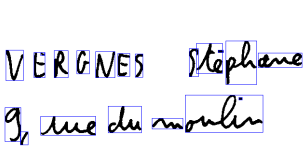
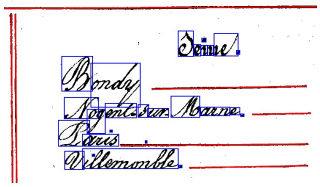


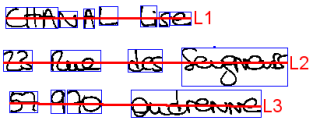
FIGURE 3.1: Schéma simplifié de la méthode DMOS-P.



(a) Composantes connexes extraites de l'image



(b) Composantes connexes et segments de droite



(c) Lignes de texte (segments incorporant des composantes connexes)

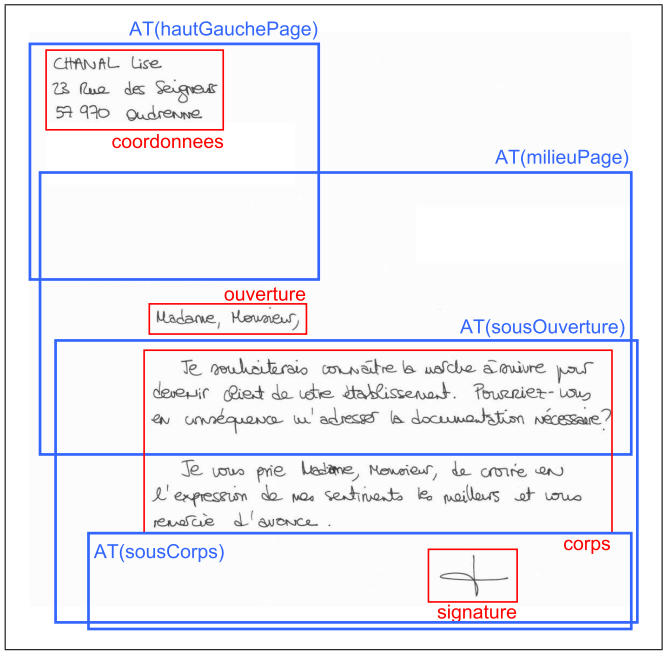
FIGURE 3.2: Exemples d'extraction de primitives avec DMOS-P : (a) des composantes connexes en bleu ; (b) des segments de droite en rouge et des composantes connexes ; et, finalement, (c) des *lignes de texte*, segments créés à partir de composantes connexes d'après la vision perceptive [LC08].

Le formalisme EPF est basé sur une extension bidimensionnelle d'une DCG ; il dispose d'opérateurs de position, similaires à ceux des grammaires de page (partie 2.1.2), mais plus génériques et extensibles : l'opérateur de base, `AT(pos)`, permet à l'utilisateur de définir de manière précise la relation entre deux éléments quelconques de la grammaire. Nous avons dans la figure 3.3 un exemple de grammaire [1] qui utilise plusieurs fois l'opérateur de position `AT(pos)`, spécifiant à chaque fois une relation spatiale distincte. Nous pouvons remarquer que les zones définies par cet opérateur sont des rectangles qui peuvent se recouvrir.

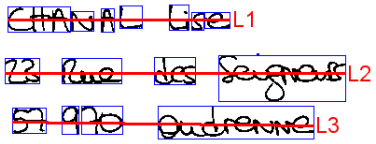
1. Pour aider à distinguer visuellement le code EPF du code Prolog qui sera présenté plus tard, nous utilisons un cadre différent pour chaque type de code.

```
pageCourrier ::=
        AT(hautGauchePage) && coordonnees &&
        AT(milieuPage) && ouverture &&
        AT(sousOuverture) && corps &&
        AT(sousCorps) && signature.

coordonnees ::=
        TERM_SEG(LigneNom) && deuxLignesAdresse.

deuxLignesAdresse ::=
        TERM_SEG(LigneAdresse1) &&
        TERM_SEG(LigneAdresse2).
```

(a) Exemple de grammaire EPF, appelée G_1



(b) Résultat de l'analyse de la grammaire G_1



(c) Détail de la règle `coordonnees` de G_1

FIGURE 3.3: Exemple de grammaire pour l'analyse d'une page de courrier, avec différents usages de l'opérateur de position `AT`. L'application de cette règle engendre l'analyse indiquée dans (b) ; les rectangles bleus indiquent les zones délimitées par l'opérateur `AT` et les rectangles rouges indiquent les éléments identifiés (composés de sous-éléments non affichés). Dans (c) nous détaillons l'application de la règle qui génère le bloc `coordonnees`.

La définition de la zone de recherche modifie aussi un curseur, qui indique la direction de recherche de l'élément suivant. Une fois qu'une zone de recherche est définie, les éléments terminaux présents dans cette zone peuvent être consommés à l'aide des opérateurs `TERM_SEG` et `TERM_CMP`, pour les segments de droite (ou les lignes de texte) et les composantes connexes, respectivement. L'analyseur DMOS-P ne possède pas actuellement de mécanismes probabilistes pour gérer le bruit ; il utilise à la place des opérateurs de pré- et de post-conditions dans les terminaux, ainsi qu'un

opérateur de recherche (`FIND`), qui augmentent la robustesse de l'analyse en échange d'un effort additionnel d'écriture de la grammaire. La vérification de ces conditions permet à l'analyseur d'ignorer le bruit jusqu'à ce qu'elles soient satisfaites.

L'analyse est effectuée de façon descendante, comme habituellement dans les DCGs. La compilation d'une grammaire EPF suit le modèle de compilation des DCGs standards ; nous allons détailler ce mécanisme car il aura des implications pour la mise en œuvre de l'analyseur stochastique.

3.1.2 Compilation avec EPF

Les grammaires dans EPF étant basées sur les DCGs, le mécanisme de compilation est le même : les règles de la grammaire sont traduites vers des prédicats avec des paramètres implicites d'entrée et sortie (figure 3.4), enchaînés entre les appels successifs des prédicats qui composent la règle. EPF est implémenté sur λ-Prolog, mais la notation Prolog standard de la figure 3.4 reprend les mêmes idées, à quelques détails d'implémentation près [LHLR93].

```
coordonnees ::=
        AT(hautGauchePage) &&
        deuxLignesAdresse LignesAdr &&
        AT(sousLignes LignesAdr) &&
        ligneCodePostal LignesCp.
```

```
coordonnees(In, Out) :-
        op_at(hautGauchePage, In, Tmp1),
        deuxLignesAdresse(LignesAdr, Tmp1, Tmp2),
        sousLignes(LignesAdr, Tmp2, Tmp3),
        ligneCodePostal(LigneCp, Tmp3, Out).
```

FIGURE 3.4: Exemple de code EPF et une transcription simplifiée en Prolog. Les paramètres d'entrée/sortie implicites `In`, `TmpX` et `Out` sont enchaînés automatiquement par le compilateur. Les opérateurs EPF tels que `AT(pos)` sont convertis vers les prédicats correspondants.

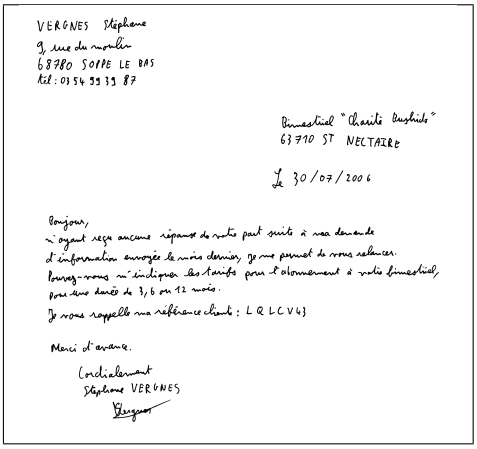
Nous reprendrons ce schéma lors de l'introduction de l'opérateur d'analyse stochastique, dans la partie 4.3. Nous passons maintenant à l'intégration des classifieurs à l'analyseur DMOS-P.

3.1.3 Classifieurs, incertitudes et scores

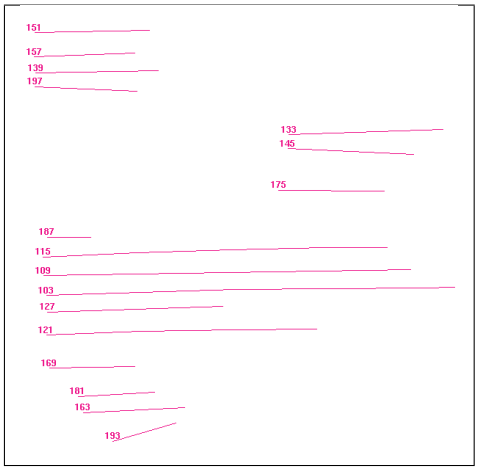
Nous cherchons à intégrer les informations provenant de classifieurs à l'analyse structurelle de documents, car ces classifieurs nous permettent d'inférer le contenu de chaque ligne. Sans cette information, il est beaucoup plus difficile d'effectuer une analyse correcte. La figure 3.5 présente les différentes visions pour un même document, selon la présence ou non de classifieurs.

Des classifieurs tels que celui de Guichard et coll. [GHTC10], intégré à DMOS-P, permettent d'obtenir des scores sous la forme de probabilités (ou log-probabilités) : il suffit d'appeler un prédicat du type `reconnaitreLigne(+Classifieur, +Ligne, -Score)`[2] pour obtenir le score associé à la zone de l'image (qui peut représenter un caractère, un mot ou une ligne, selon le type de classifieur). Ce score

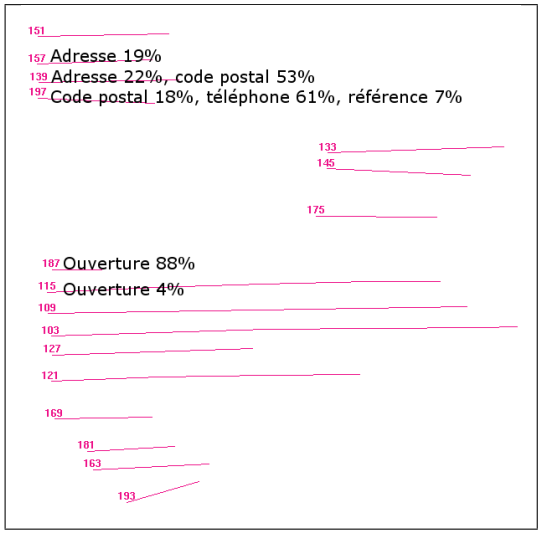
2. En reprenant la convention de notation Prolog, un argument `+Arg` est une « entrée » (instancié à l'entrée) et `-Arg` est une sortie (non-instancié à l'entrée).



(a) Document original



(b) Vue sans classifieurs



(c) Vue avec quelques classifieurs

FIGURE 3.5: Une image de document (a), sa perception en l'absence de classifieurs (b) et sa perception avec quelques classifieurs (*adresse*, *code postal*, *téléphone*, etc. dans (c)). L'information structurelle permet de cibler les lignes à reconnaître. Les classifieurs leur attribuent des étiquettes avec une incertitude associée, dont la somme est inférieure à 100% (pour l'étiquette implicite « autre »).

doit être converti et, au besoin, normalisé, pour ensuite être utilisé par le mécanisme stochastique (dont les détails sont discutés dans la partie 4.1).

Pour l'implémentation de l'analyseur stochastique dans DMOS-P, nous utilisons quelques concepts qui ne sont pas spécifiques à cette méthode, mais au langage sur lequel il est basé, λ-Prolog. Nous voyons maintenant les aspects de ce langage pertinents pour l'analyse stochastique.

3.2 Recherche de solutions avec λ-Prolog

λ-Prolog [BR92, LHLR93] est un langage de programmation logique typé qui incorpore λ-termes et apporte, entre autres, la notion de *portée* à plusieurs niveaux du langage. Parmi les caractéristiques qui l'ont rendu le langage de choix pour l'implémentation de DMOS-P, nous pouvons citer : son mécanisme grammatical similaire aux DCGs de Prolog, mais avec possibilité de manipulation de l'ordre supérieur [LHLR93] ; son adaptabilité à la création de compilateurs ; et le fait que la compilation dans ce langage génère du code C, ce qui permet l'intégration avec les outils d'analyse d'image (majoritairement en C++).

L'implémentation de λ-Prolog utilisée, appelée Prolog/Mali [BR92], offre des prédicats et possibilités de gestion mémoire qui dépassent les limitations d'un mécanisme d'analyse logique pure. Parmi ces prédicats, nous citons `findall`, qui permet la récupération de toutes les solutions à un prédicat donné. Nous avons vu lors de l'étude bibliographique que c'est un des besoins d'un analyseur stochastique, raison pour laquelle nous le détaillons ici.

Recherche de toutes solutions : prédicat `findall` dans Prolog/Mali

Le prédicat `findall(Term, Goal, Sols)` est une extension considérée utile par la majorité de la communauté Prolog [Nai86]. Il permet la récupération de toutes les solutions de la forme `Term` à un prédicat `Goal`, les unifiant dans la liste `Sols`. Voici un exemple[3] très simple de son utilisation, en Prolog :

```
boit(jean, eau).
boit(jean, lait).
boit(jacques, lait).
?- findall(P, boit(P, lait), Sols).
        Sols = [jean, jacques]
```

Naish [Nai86] reprend une définition plus formelle pour ce prédicat :
$findall(Term, Goal, Solutions) \iff$
$\forall X(\exists L_1, \ldots, L_n(Goal \wedge X = Term) \iff member(X, Solutions))$
$L_1, \ldots, L_n$ est une liste de variables locales dans $Goal$.

Prolog/Mali possède une définition de ce prédicat dans sa librairie standard [HRB], dont une implémentation simplifiée est présentée dans la figure 3.6.

Ce prédicat est implémenté à l'aide de mécanismes non standards : nous pouvons remarquer que `add_solution` a besoin de conserver les résultats même après le retour arrière provoqué par le `fail` qui suit. Pour éviter que l'unification soit défaite (et donc la perte des termes dans `Solutions`), l'implémentation Prolog/Mali utilise du code C particulier pour ce prédicat. Ce code permet la copie des termes qui doivent être sauvegardés par la suite.

L'utilisation de ce prédicat a d'importantes conséquences sur la performance en temps de calcul et mémoire : d'abord, le mécanisme de recherche étant celui de la résolution standard, aucune heuristique n'est utilisée pour guider la recherche ; deuxièmement, il faut trouver toutes les solutions avant de pouvoir continuer, et chaque nouvelle solution requiert une copie de termes qui peut s'avérer assez coûteuse, car le mécanisme standard d'unification est défait lors de cette copie. Nous verrons

3. Exemple inspiré de [HRB].

```
%findall((input) Term, (input) Goal, (output) Solutions)
findall(Term, Goal, Sols) :-
        Goal(Term), % tentative d'obtention d'une solution
        add_solution(Term, Solutions), % stockage de la solution
        fail. % échec pour forcer la recherche d'une prochaine solution
findall(_, _, Sols). % réussite après fin de la collecte
```

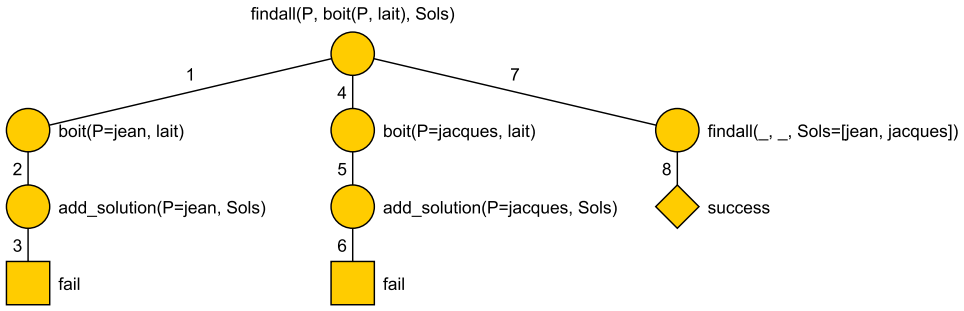


FIGURE 3.6: Implémentation simplifiée de `findall`, avec l'arbre d'analyse associé. Les numéros indiquent l'ordre d'exécution.

quelques détails de ces conséquences dans la partie 5, qui présente une évaluation quantitative de l'analyse.

Avec la présentation de DMOS-P et des outils pertinents dans Prolog/Mali, nous procédons à la description de l'implémentation effectuée lors du stage, avec les contributions apportées.

Chapitre 4

Implémentation d'une extension stochastique de DMOS-P

L'extension stochastique de DMOS-P a pour but de permettre au concepteur de la grammaire d'utiliser l'information du contenu (provenant des classifieurs), avec son incertitude associée, ou d'ajouter d'autres sources d'information numérique à l'analyse stochastique. Nous voulons que l'intégration de cette information soit aussi simple et transparente que possible pour l'utilisateur. Nous définissons d'abord la sémantique des scores au sein d'EPF (partie 4.1), pour ensuite présenter les trois opérateurs proposés aux concepteurs des grammaires : deux liés directement aux scores (`ADD_SCORE` et `SELECT_SCORE`, décrits dans 4.2) et le dernier lié au processus d'exploration des solutions (`FIND_BEST_FIRST`, décrit dans 4.3). Dans chacune de ces parties, nous présentons d'abord une vision utilisateur, ensuite les détails d'implémentation.

4.1 Insertion des scores dans EPF

Au sein d'EPF, nous appelons les probabilités associées aux nœuds des *scores*. Nous définissons d'abord son mode d'utilisation pour ensuite expliquer les raisons de ce choix.

4.1.1 Scores du point de vue de l'utilisateur

Le concepteur d'une grammaire EPF doit traiter les valeus attachées aux règles comme des pénalités (ou coûts) : dans l'existence de plusieurs résultats, celui de plus petit score est rendu en premier. Le concepteur doit aussi veiller à ce que les scores soient des valeurs positives (ou nulles). La propagation des scores au long d'un chemin de dérivation se fait par addition des pénalités dans chaque règle. Toute règle a, par défaut, un score nul, sauf si son concepteur l'ajoute une pénalité explicitement.

4.1.2 Sémantique des probabilités en tant que pénalités

Le processus d'analyse stochastique incorpore une probabilité à chaque règle de la grammaire, mais dans notre contexte d'utilisation nous ne nous intéressons pas à tous les aspects probabilistes ; comme vu dans la partie 2.2.3, les probabilités incorporent une notion de *préférence* et permettent l'ordonnancement de plusieurs résultats d'analyse autrement équivalents. Nous traitons donc ces

probabilités comme un score, ou plutôt une pénalité, de façon analogue au coût des différentes règles dans [TI94] : le chemin choisi est celui qui a le plus petit score.

Compte tenu du fait que la propagation des probabilités au long d'un chemin donné, dans les grammaires stochastiques, se fait à travers la multiplication des probabilités dans chaque nœud, le nombre résultant diminue continuellement au long d'un chemin, parfois de plusieurs ordres de magnitude. Pour éviter des *underflows*, nous utilisons des log-probabilités à la place de probabilités. Jurafsky [JM00] discute cet usage dans le contexte de n-grammes, mais il est aussi applicable à l'analyse stochastique. Nous utilisons l'opposé de cette valeur, en autorisant une valeur positive quelconque pour les coûts. Cela implique que leur propagation se fait par *addition dans un espace logarithmique* au lieu de *multiplication dans un espace linéaire*. Si nous reprenons alors les scores probabilistes des classifieurs dans DMOS-P, il suffit de les convertir vers des pénalités (en utilisant $(1 - \log(score))$ à la place de $score$) pour qu'ils rentrent dans le cadre souhaité de pénalités.

4.2 Opérateurs `ADD_SCORE` et `SELECT_SCORE`

4.2.1 Définition et utilisation

Opérateur `ADD_SCORE(+Score : float)` [1]

L'opérateur `ADD_SCORE` au sein d'une règle permet l'ajout d'une pénalité à cette règle. Cette pénalité est, le plus souvent, originaire de l'appel à un classifieur, comme l'appel à `recognize_line` dans l'exemple ci-dessous :

```
% reconnaitreLigne +Classifieur -Ligne
reconnaitreLigne Classifieur Ligne ::=
        TERM_SEG(Ligne) && % consomme un terminal
        recognize_line(Classifieur, Ligne, Prob) && % obtient l'incertitude
        incertitude_vers_score(Prob, Score) && % Score = (1 - log(Prob))
        ADD_SCORE(Score). % ajoute le score à l'analyse courante
```

Le score rendu par le classifieur n'est pas automatiquement ajouté à l'analyse ; le prédicat `reconnaitreLigne` de l'exemple encapsule l'appel au classifieur avec la conversion et l'ajout du score à l'analyse. Le concepteur de la grammaire n'a donc pas besoin de le traiter directement ; il peut se contenter de faire un appel direct à `reconnaitreLigne` au lieu de récupérer le terminal avec `TERM_SEG` et ensuite appeler le classifieur.

Opérateur `SELECT_SCORE(+Regle : RegleEPF, -DeltaScore : float)`

L'opérateur `SELECT_SCORE` a deux finalités distinctes, mais liées :

1. il permet d'effectuer une analyse sans modifier le score courant ; il est l'équivalent de l'opérateur `SELECT(+Regle : RegleEPF)` (qui effectue une analyse sans modifier les terminaux consommés) pour les questions relatives au score ;
2. il permet la récupération du score produit par une règle (via `DeltaScore`) ; les scores ne sont pas directement accessibles à l'extérieur des règles qui les ajoutent.

1. Nous reprenons ici, et dans le reste du rapport, la notation des modes d'utilisation de Prolog [HRB] : de manière générale, un paramètre d'entrée est précédé d'un (+) et un paramètre de sortie est précédé d'un (−).

Un exemple d'utilisation de cet opérateur (figure 4.1) est l'implémentation d'un opérateur d'analyse « glouton » (contraire à l'analyse stochastique) qui essaie, en séquence, l'application d'une liste de classifieurs sur une ligne, en s'arrêtant dès que le score obtenu par un des classifieurs est suffisamment bon (ne dépasse pas le seuil établi). Une fois que le score obtenu est accepté, nous pouvons l'ajouter à l'analyse courante avec un appel à `ADD_SCORE`.

```
% reconnaitreLigneGlouton +ListeClassifieurs +Seuil -Ligne
reconnaitreLigneGlouton [Cl|_] Seuil Ligne ::=
        SELECT_SCORE(reconnaitreLigne(Cl, Ligne), Score) && % score non ajouté
        Score < Seuil, ! && % si le score est assez petit, on l'accepte
        ADD_SCORE(Score).
reconnaitreLigneGlouton [_|Reste] Seuil Ligne ::= % sinon, on essaye le suivant
        reconnaitreLigneGlouton Reste Seuil Ligne.
```

FIGURE 4.1: Exemple d'utilisation de l'opérateur `SELECT_SCORE`.

4.2.2 Implémentation du score

Le score, incorporé en tant que pénalité croissante au long de l'analyse, doit être enregistré quelque part, et ça de manière transparente pour l'utilisateur. Le mécanisme standard des DCGs, la propagation via des variables d'entrée/sortie implicites (décrit dans la partie 3.1.2), est la première solution envisagée pour la gestion du score. Cependant, cet ajout requiert une modification du compilateur (pour l'intégration d'une autre paire d'arguments implicites d'entrée/sortie tels que `ScoreIn`/`ScoreOut`), ce qui peut avoir des conséquences sur la performance, notamment car nous souhaitons conserver les analyses déterministes existantes telles quelles, sans forcer l'ajout d'attributs inutiles.

Les langages de programmation logique, par défaut, ne possèdent pas d'opérateur d'affectation. Prolog/Mali, par contre, offre un mécanisme de substitution de valeurs (via la directive `subst` et l'opérateur d'affectation `<-`) qui permet d'échapper à l'unification logique. Avec cet outil, nous pouvons remplacer une valeur par une autre directement comme dans un langage impératif, sans passer par les paramètres implicites. Cet opérateur est conforme au mécanisme de retour arrière, ce qui assure son bon fonctionnement au sein de l'analyse. Pour incrémenter le score d'une analyse, il suffit donc de récupérer la valeur courante, lui ajouter le nouveau score et enfin remplacer l'ancienne valeur.

Le résultat pratique de ce mécanisme est l'introduction des scores de manière transparente, de façon optionnelle et sans nécessité de modification profonde au sein du compilateur EPF. Toutes les règles qui n'ont pas de pénalité associée n'ont besoin d'aucune modification ; pour les autres, le concepteur de la grammaire peut ajouter des pénalités de façon arbitraire avec `ADD_SCORE`. Son implémentation [2] est alors triviale (figure 4.2), une fois que le prédicat qui garde le score (convenablement appelé `score`) ait été impliqué précédemment [3].

L'implémentation de `SELECT_SCORE` (figure 4.2) se fait de manière similaire : nous gardons le score avant l'appel à la règle en paramètre, nous appelons la règle et nous restaurons l'état original. La différence de score avant et après la règle est rendue via `DeltaScore`.

2. Par convention, nous implémentons les opérateurs EPF par des prédicats λ-Prolog de même nom, mais en minuscules.

3. En Prolog, les prédicats dynamiques sont implémentés avec le mécanisme `assert/retract` ; dans λ-Prolog, ces

```
add_score(Score) :-
        score(ScoreAvant), % score est un prédicat dynamique impliqué
        ScoreApres is ScoreAvant $+ Score,
        ScoreAvant <- ScoreApres. % substitution de la valeur

select_score(Regle, DeltaScore) :-
        score(ScoreAvant),
        call Regle, % appels à ADD_SCORE dedans modifient le score courant
        score(ScoreApres),
        DeltaScore is ScoreApres $- ScoreAvant,
        ScoreApres <- ScoreAvant. % restauration du score original
```

FIGURE 4.2: Implémentation simplifiée des opérateurs `ADD_SCORE` et `SELECT_SCORE`.

Bien sûr, la simple insertion des scores ne change en rien l'ordre d'analyse ; mais leur utilisation permet de trouver la meilleure solution, avec une exploration exhaustive des possibilités.

4.3 Exploration exhaustive : opérateur `FIND_BEST_FIRST`

4.3.1 Définition et utilisation

L'opérateur `FIND_BEST_FIRST(-Valeur : T) FOR (+RegleT : RegleEPF_T)` applique la règle `RegleT` pour produire en sortie une valeur `Valeur` de type `T` (un type polymorphe quelconque) en assurant deux propriétés fondamentales pour une analyse stochastique correcte :

1. si plusieurs arbres de dérivation existent pour la règle `RegleT`, le meilleur (celui de score le plus petit) sera utilisé en premier ;
2. tous les autres arbres de dérivation resteront disponibles pour le mécanisme de retour-arrière : en cas d'échec après l'application de l'opérateur, le deuxième meilleur arbre de dérivation sera utilisé, et ainsi de suite. Cela revient à une exploration extensive de toutes les possibilités de `RegleT`, dans l'ordre dicté par le score.

Le manque d'une de ces propriétés invalide le comportement attendu d'un analyseur stochastique.

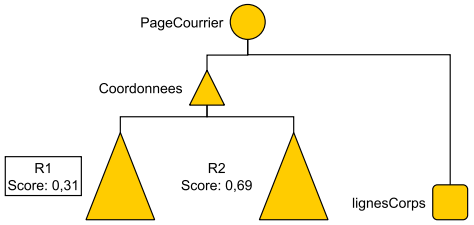
`RegleT` est une règle EPF quelconque, sauf que son dernier paramètre (explicite) est obligatoirement de type T. En notation fonctionnelle, si une règle EPF est de type $RegleEPF$, alors `RegleT` est de type $T \rightarrow RegleEPF$.

La figure 4.3 montre un exemple d'utilisation de cet opérateur, avec son incorporation au sein de la grammaire de la figure 2.3. Nous supposons ici que `coordonnees` est une règle qui possède comme dernier paramètre une valeur de même type que celui de `Coord`. Nous avons deux possibilités pour la règle `coordonnees`. Le score de l'arbre à gauche est plus petit, donc `FIND_BEST_FIRST` nous rend d'abord le résultat avec application de `troisLignesAdresse` ; si `lignesCorps` échoue, `FIND_BEST_FIRST` rendra le résultat de l'arbre à droite.

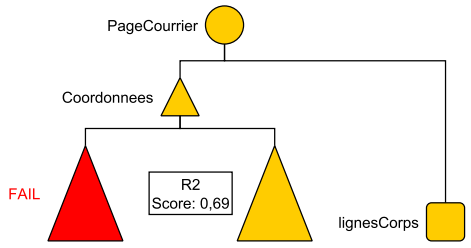
prédicats peuvent être impliqués au fur et à mesure de l'analyse avec l'opérateur d'implication =>.

```
%pageCourrier -Coord -Corps
pageCourrier Coord Corps ::=
        FIND_BEST_FIRST(Coord) FOR (coordonnees) && lignesCorps Corps.
```

(a) Grammaire avec utilisation de `FIND_BEST_FIRST`



(b) Exploration de la branche de plus petit score



(c) Exploration de solution alternative après échec

FIGURE 4.3: Exemple d'utilisation de `FIND_BEST_FIRST` au sein d'une grammaire EPF (a) et un arbre d'analyse associé : l'opérateur essaye d'abord avec l'arbre de plus petit score (b), mais un échec lors de l'application de `lignesCorps` amène à l'exploration de la deuxième meilleure solution (c).

4.3.2 Implémentation

Nous voulons intégrer l'analyse stochastique de manière aussi transparente que possible au sein de DMOS-P ; l'intégration de l'analyse stochastique de façon imposée à tout l'arbre de recherche est contraignante et ne permet pas de profiter des connaissances du concepteur de la grammaire. L'utilisation d'un opérateur pour activer ce mécanisme a l'avantage de limiter la taille de l'arbre d'exploration. En plus, cela assure le fonctionnement standard partout ailleurs, en évitant la nécessité de gérer deux types de grammaires, avec et sans analyse stochastique.

L'analyse stochastique doit faire face à plusieurs parcours possibles pour en choisir le meilleur (comme dans l'exemple de l'introduction, figure 1.3) ; en l'absence d'heuristiques, la seule solution consiste à explorer toutes les analyses possibles, pour ensuite les trier et choisir la meilleure. La façon la plus simple de le faire est à travers le prédicat `findall` (déjà mentionné dans la partie 3.2). Ce prédicat nous fournit une liste de solutions, que nous pouvons trier grâce aux scores pour rendre toujours la meilleure d'abord. Une implémentation simplifiée de l'opérateur `FIND_BEST_FIRST` à l'aide de `findall` est présentée dans la figure 4.4. Cet opérateur prend les mêmes paramètres mentionnés précédemment ; les paramètres implicites `In` et `Out` sont explicités ici parce qu'ils sont utilisés par l'implémentation.

L'utilisation transparente de `findall` cache un piège de l'utilisation du mécanisme extra-logique : dû au retour arrière implicite, toutes les instanciations qui ne sont pas explicitement sauvegardées sont perdues après l'appel à l'opérateur `FIND_BEST_FIRST`. Les éléments présents dans les paramètres implicites sont toujours sauvegardés (car ils représentent l'évolution réelle de l'analyse), mais les paramètres des règles appelées ne le sont pas. Pour cette raison, toutes les instanciations produites par la règle à l'intérieur de `FIND_BEST_FIRST` que l'utilisateur veut que soient gardées (notamment, les valeurs produites par l'analyse) doivent être indiquées via le premier paramètre de l'opérateur

```
% find_best_first(-Valeur, +RegleT, +In, -Out)
find_best_first(Valeur, RegleT, In, Out) :-
        findall(Etat, construire_etat(RegleT, In, Etat), Etats), %
            application de la règle et stockage des solutions dans Etats
        score_sort(Etats, EtatsTries), % tri selon le score
        member(etat(Valeur, Out, Score), EtatsTries). % récupération avec
            retour arrière

% construit un "état d'analyse", composé des valeurs de sortie (Valeur + Out)
     ainsi que du score
construire_etat(+RegleT, +Out, -Etat) :-
        RegleT(Valeur), % application de la règle
        score(Score), % prédicat interne, récupère le score obtenu
        Etat = etat(Valeur, Out, Score). % score gardé dans les états pour
            pouvoir les trier
```

FIGURE 4.4: Implémentation simplifiée de `FIND_BEST_FIRST` (les variables implicites `In` et `Out` sont créées par le mécanisme de compilation EPF→λ-Prolog). Nous procédons en trois étapes : récupération de toutes les solutions, tri selon le score et enfin rendu du(des) résultat(s).

(`Valeur`). Nous verrons plus tard dans la partie 5.2 que ce mécanisme nécessite quand même que l'utilisateur soit attentif par rapport au paramètre de sortie : les copies implicites, notamment pour les listes, peuvent engendrer un surcoût mémoire excessif.

L'implémentation de cet opérateur complète le mécanisme fondamental d'analyse stochastique. Nous verrons maintenant les résultats de notre application de ce mécanisme dans DMOS-P.

Chapitre 5

Validation : évaluation et résultats

L'analyseur stochastique a été testé et évalué sur deux types de documents : des courriers manuscrits (dans le cadre de la campagne d'évaluation nationale RIMES, détaillée dans la partie 5.1) et des documents d'archives (pour l'analyse de tableaux contenant des numéros de séquence, détaillée dans la section 5.2). La première évaluation a permis de comparer le taux de reconnaissance de la grammaire par rapport à un mécanisme structurel déterministe existant, tandis que la deuxième a fourni des informations liées surtout à la performance de l'analyse.

5.1 Intégration de l'analyse stochastique au sein d'une grammaire existante : courriers manuscrits

Pour l'évaluation du mécanisme stochastique, nous introduisons les opérateurs `FIND_BEST_FIRST` et `ADD_SCORE` au sein d'une grammaire existante (déterministe) d'analyse de courriers manuscrits. Cette grammaire a participé au concours d'évaluation RIMES en 2008. Nous présentons ce concours et les critères pour la définition des taux d'erreur, pour ensuite montrer les résultats obtenus avec nos modifications.

5.1.1 Présentation de RIMES

La campagne d'évaluation RIMES [GCG+06], Recherche et Indexation de données Manuscrites et fac-similES, a créé une base de courriers destinée à l'évaluation de techniques de reconnaissance de documents. Ces courriers comprennent des lettres manuscrites écrites par des individus et destinées à des entreprises.

Parmi les différentes tâches qui peuvent être effectuées dans cette base, figure la reconnaissance structurelle, dont le but est de localiser des blocs de texte représentant des zones de même contenu sémantique. Nous cherchons à identifier jusqu'à huit types de contenus distincts (les couleurs entre parenthèses font référence à l'exemple de la figure 5.1) :

- coordonnées de l'expéditeur (rouge) ;
- coordonnées du destinataire (orange) ;
- date/lieu (bleu) ;
- objet (absent dans l'exemple) ;
- ouverture (gris) ;
- corps de texte (magenta) ;

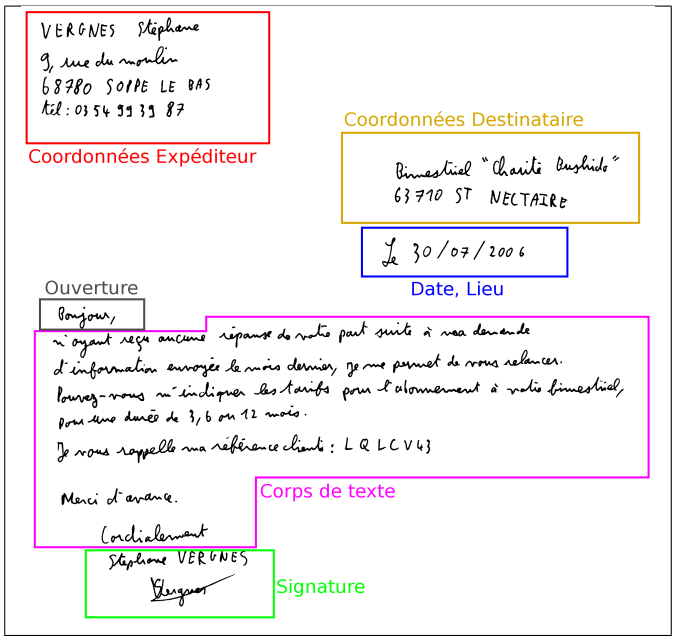


FIGURE 5.1: Un document du corpus RIMES avec les blocs définis par la vérité terrain.

– signature (vert) ;
– PS/pièce jointe (absent dans l'exemple).

Chaque vérité terrain est formée d'une zone ayant une des étiquettes précédentes. Tous les pixels noirs dedans sont comptés pour cette zone, et ensuite comparés avec la vérité terrain. Le taux d'erreur est défini par rapport au rappel (nombre de pixels noirs corrects / nombre de pixels noirs de la référence). Ceci permet une granularité très fine et ignore les zones blanches de l'image.

Lors du concours RIMES 2008, une grammaire a été créée avec DMOS-P pour participer au concours. Cette grammaire a obtenu de bons résultats [LC08] (taux d'erreur global de 7,44% pour un lot de 1250 images analysées ; le détail par classe est affiché dans le tableau 5.1) en utilisant seulement l'information structurelle des documents.

L'intégration de classifieurs au sein de cette grammaire permet l'inférence du contenu des lignes, ce qui engendre une amélioration de l'analyse ; nous verrons comment cette intégration est faite au niveau DMOS-P et son impact sur la grammaire d'analyse de courriers existante.

5.1.2 Intégration des opérateurs stochastiques

La grammaire existante (déterministe) n'admet plus qu'une solution par document : de possibles ambiguïtés dans la structure des documents à analyser sont traitées avec une approche gloutonne basée sur l'analyse descendante des DCGs : s'il y a un nombre variable de terminaux (lignes) qui peuvent être consommées, alors la grammaire essaye de consommer le maximum de lignes d'abord, pour ensuite décroître le nombre d'éléments à consommer jusqu'au minimum possible. C'est le cas, par exemple, de la règle qui reconnaît le bloc de coordonnées du destinataire : elle essaye de consommer 5 lignes d'abord et diminue d'une ligne à chaque échec de l'analyse. Ce mécanisme est une approche standard pour compenser le manque de flexibilité imposé par une analyse déterministe.

Pour effectuer l'intégration de notre opérateur stochastique, nous procédons en deux étapes :

1. recherche de l'endroit où l'intégration de l'analyse stochastique peut effectivement apporter un gain sur le taux de reconnaissance (partie 5.1.3) ;
2. modification de l'analyse avec l'insertion de l'opérateur `FIND_BEST_FIRST` (partie 5.1.4) .

Nous finissons cette section par des commentaires sur les résultats obtenus (partie 5.1.5).

5.1.3 Définition du point d'insertion pour maximiser le rapport gain/effort

Une intégration naïve de classifieurs à la grammaire existante n'apporte pas de grands avantages : d'abord, le taux d'erreur des classifieurs eux-mêmes ne doit pas être trop élevé, sinon nous risquons de substituer une erreur par une autre de taux équivalent. En effet, nous ne disposons pas pour l'instant de classifieurs conçus pour analyser une ligne de texte manuscrit ; à la place, nous avons des classifieurs de reconnaissance de mots. En plus, nous nous plaçons dans un cadre où nous ne voulons pas qu'effectuer de la reconnaissance, mais aussi du rejet (car nous utilisons le contenu pour guider l'analyse de la structure) ; ce mode d'utilisation impose quelques contraintes sur l'efficacité espérée des classifieurs.

Ensuite, la structuration déterministe de la grammaire, avec plusieurs coupures, et la considération implicite de l'approche gloutonne exigent des modifications non-triviales pour que les scores fournis par les classifieurs puissent avoir un impact significatif sur la qualité du résultat : une insertion immédiate de l'opérateur `FIND_BEST_FIRST` à la racine de la grammaire n'a aucun impact, car il n'y a pas de branches alternatives à explorer ; si nous éliminons tout simplement les coupures, nous avons une explosion combinatoire de possibilités d'analyse. Enfin, nous ne pouvons pas oublier que plusieurs types de lignes (celles du corps de texte, par exemple) ne sont pas très adaptées à l'analyse par les classifieurs, car il n'y a pas de mots-clés assez discriminants.

Nous partons donc d'une solution minimaliste, où nous essayons d'apporter un minimum de modifications à la grammaire existante ; cette approche permet de contrôler les modifications et comprendre, via des comparaisons avec la grammaire initiale, les forces et faiblesses de la nouvelle grammaire. Pour essayer de maximiser l'impact de l'introduction du mécanisme stochastique tout en minimisant l'effort de réécriture, nous avons considéré les données du tableau 5.1, qui nous indique les principaux contributeurs à l'erreur globale dans la grammaire initiale.

Classe	Erreur (%)	% de pixels noirs de l'image	Contribution à l'erreur globale (%)
PS/PJ	**84,81%**	0,19%	2,16%
Signature	11,31%	4,12%	6,25%
Corps de texte	3,08%	**61,48%**	**25,43%**
Ouverture	18,61%	2,90%	7,25%
Coordonnées Expéditeur	7,33%	15,10%	14,86%
Coordonnées Destinataire	13,30%	9,06%	16,18%
Date, Lieu	23,53%	3,19%	10,07%
Objet	33,40%	3,97%	17,80%

TABLE 5.1: Taux d'erreur par classe, contribution relative en nombre de pixels et erreur relative par rapport à l'erreur globale, de la grammaire initiale de reconnaissance de courriers manuscrits [LC08].

Si nous ne considérons que le taux d'erreur dans la classe, celle qui a le plus besoin d'être améliorée est PS/PJ ; cependant, elle répond pour moins de 2% de l'erreur globale ; le corps en est sûrement le principal responsable, mais nous avons déjà presque 97% de rappel dans cette classe. Parmi les restantes, nous synthétisons dans le tableau 5.2 les principales raisons qui motivent/empêchent le choix de ces classes pour l'introduction de l'opérateur stochastique.

Caractéristique	Motifs	Favorise le choix des classes	Défavorise le choix des classes
Commence par un mot-clé (sur la première ligne)	Facilite la reconnaissance par le classifieur	**Ouverture** (commence par *Madame/Monsieur/Bonjour* dans 90% des cas) Objet (commence par *Objet* dans 80% des cas)	Coordonnées Signature Date, Lieu
Toujours écrit sur une seule ligne	Pas de variabilité structurelle à traiter	**Ouverture** Date, Lieu	Coordonnées Objet Signature
Présent sur la grande majorité des documents	Évite le besoin de traiter le cas où la classe est absente	Coordonnées **Ouverture** Signature	Objet Date, Lieu

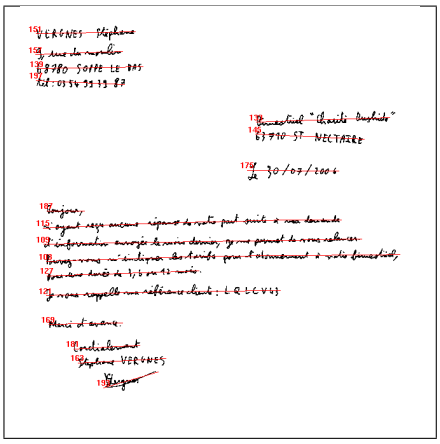
TABLE 5.2: Critères de choix pour la classe à être ciblée par l'introduction de l'opérateur stochastique.

La classe la plus favorisée est l'Ouverture ; en plus de ces raisons, elle a un avantage particulier : la ligne entière possède très peu de variations. Un dictionnaire contenant huit expressions (*Madame*, *Monsieur*, *Messieurs*, *Madame/Monsieur*, *Monsieur/Madame*, *Bonjour*, *Monsieur le Directeur* et *Monsieur le Maire*) est capable de représenter correctement plus de 90% des occurrences. Nous pouvons alors envoyer la ligne entière au classifieur [1], ce qui évite le cumul des erreurs de segmentation des mots (par exemple, *Objet* incorrectement segmenté en *Obj et*). En plus, le classifieur obtient de meilleurs résultats avec plus d'information disponible (par rapport à un mot court isolé). Ces raisons nous ont amené à introduire des informations stochastiques dans la règle qui reconnaît le bloc d'ouverture.

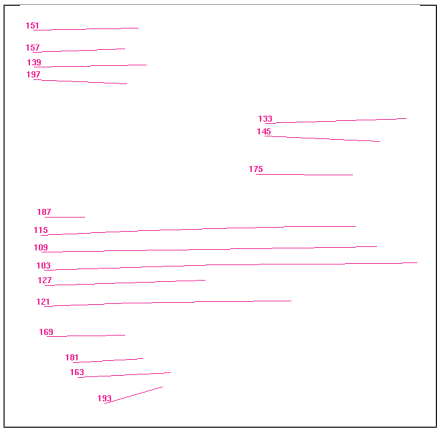
5.1.4 Modification de l'analyse grâce au mécanisme stochastique

Dans la grammaire existante [LC08], l'analyse d'un courrier se fait selon le schéma de la figure 5.2 ; la dernière ligne (ou les deux dernières lignes) sont reconnues en tant que la signature, et les lignes au-dessus font partie du corps, jusqu'à ce qu'une ligne avec les bonnes propriétés (absence de ligne au-dessus, ou une ligne courte sans ligne longue au-dessus, etc.) soit rencontrée. Elle est considérée la ligne d'ouverture dans l'analyse purement structurelle (que nous appellerons « méthode structurelle », ou « de référence »). La règle qui analyse le corps de texte, ens particulier, est composée de huit variantes et fonctionne de manière similaire à celle commentée en début de la partie 5.1.2 : la version qui consomme le plus de lignes est tentée d'abord, ensuite les autres. Cette règle ne dispose que de la longueur et de la distance entre les lignes pour pouvoir inférer l'appartenance de celles-ci au corps de texte, ce qui exige un traitement assez complexe.

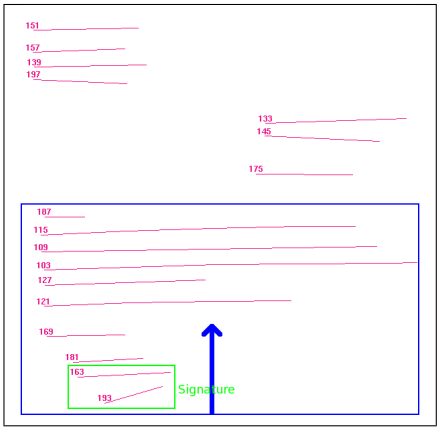
1. Le classifieur arrive à gérer les espaces entre les mots, si la séquence complète lui est fournie.



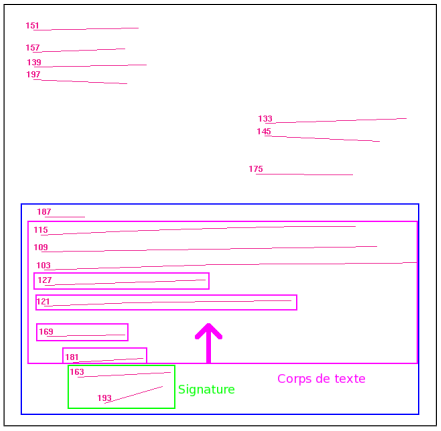
(a) Extraction des lignes de texte



(b) Vue des terminaux (sans classifieurs)



(c) Reconnaissance de la signature



(d) Reconnaissance du corps de texte et de l'ouverture

FIGURE 5.2: Schéma de l'analyse de la grammaire initiale de reconnaissance de courriers manuscrits [LC08] : (a) extraction des lignes de texte (pour les obtenir en tant que terminaux (b)), ensuite analyse du bas vers le haut (c) pour la reconnaissance de la signature, du corps de texte (d) et enfin de l'ouverture. Le reste de l'analyse se fait du haut vers le bas avec les lignes restantes.

En introduisant l'opérateur `FIND_BEST_FIRST` au sein de cette règle, nous pouvons travailler de 3 manières. Nous présentons d'abord ces méthodes pour ensuite discuter leurs résultats :

1. Méthode stochastique (ou *dirigée par le contenu*) : utilisation d'un classifieur pour chercher la ligne d'ouverture, avec un dictionnaire contenant les huit expressions mentionné précédemment. Le score rendu par ce classifieur permet de trouver (avec grande probabilité) la ligne d'ouverture correcte ; cela revient donc à n'utiliser presque aucune information structurelle, en appliquant le classifieur sur toutes (ou presque toutes) les lignes du document et en récupérant le premier résultat fourni ; c'est la méthode indiquée dans la figure 5.3 ;
2. Méthode hybride : applique la méthode 1 (stochastique) et, selon le score obtenu par rapport à un seuil fixe (défini empiriquement), garde le résultat ou applique la méthode structurelle

(de référence) à sa place. L'idée est de n'utiliser le contenu que lorsqu'il est considéré « assez fiable » ;

3. Méthode hybride avec segmentation : même principe que la méthode 2, sauf que le dictionnaire utilisé ne contient que les quatre mots suivants : *Monsieur*, *Madame*, *Bonjour* et *Cher(s)* ; et au lieu d'envoyer la ligne entière au classifieur, nous n'envoyons que le premier mot. La segmentation des mots est basée sur l'algorithme de Voronoï [LCL06].

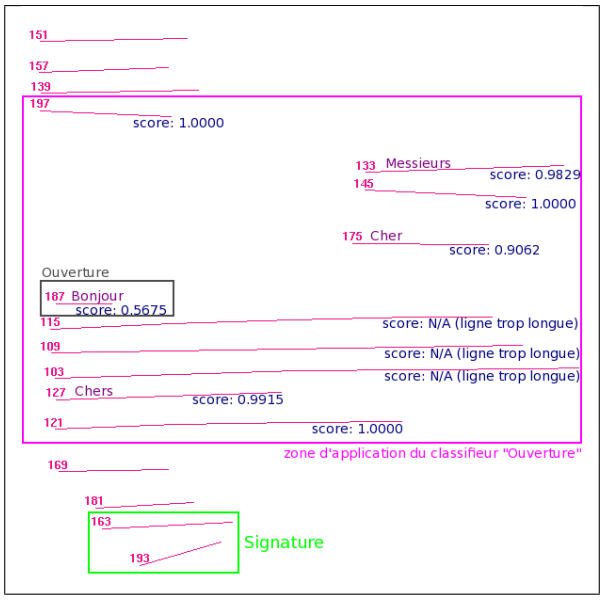


FIGURE 5.3: Méthode d'analyse dirigée par le contenu : sauf pour les lignes trop hautes, trop basses ou trop longues, toutes les autres sont analysées pour chercher celle qui ressemble le plus à la ligne d'ouverture. Le reste de l'analyse est effectué de manière structurelle.

Le résultat de l'application de ces méthodes, appliquées sur l'ensemble de 1250 images de test, est affiché dans le tableau 5.3 et dans la figure 5.4. Dû à la façon dont l'analyse se déroule (figure 5.2), les classes Signature et PS/PJ n'ont subi aucun changement et ont été omises.

Classe	Erreur méthode structurelle (référence) (%)	Erreur méthode 1 (stochastique) (%)	Erreur méthode 2 (hybride) (%)	Erreur méthode 3 (avec segmentation) (%)
Corps de texte	3,08%	1,54%	1,61%	2,35%
Ouverture	**18,61%**	8,50%	**6,37%**	9,17%
Coordonnées Destinataire	7,33%	7,69%	7,21%	7,24%
Coordonnées Expéditeur	13,30%	13,58%	11,13%	12,03%
Date, Lieu	23,53%	21,40%	20,34%	20,78%
Objet	33,40%	25,61%	24,85%	26,46%
Total	7,44%	5,91%	**5,53%**	6,23%

TABLE 5.3: Taux d'erreur par classe selon les différentes méthodes d'analyse. Le graphe de la figure 5.4 permet une comparaison visuelle des valeurs pour les méthodes 1 et 2.

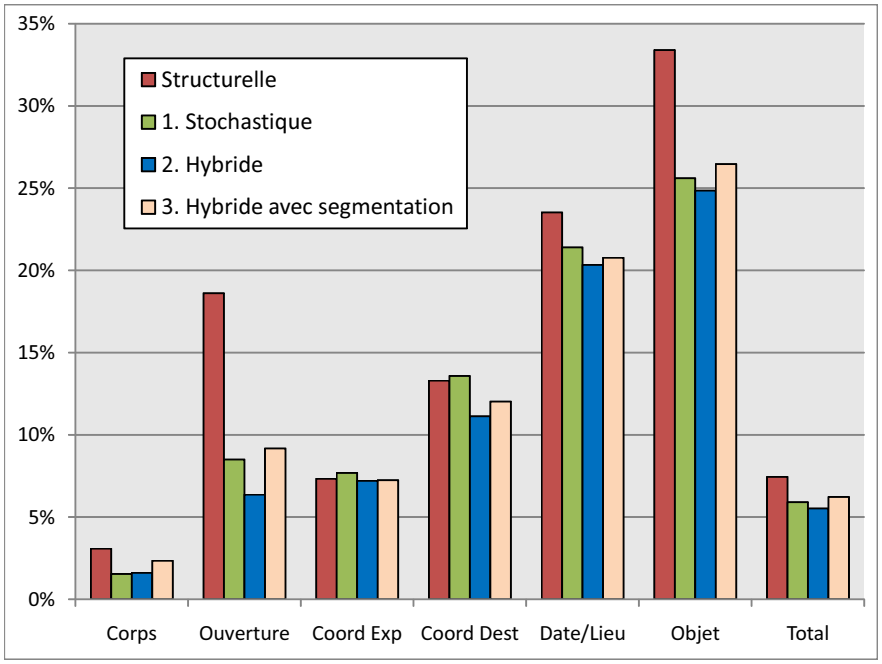


FIGURE 5.4: Erreur par classe selon les méthodes : structurelle pure (référence), stochastique (1), hybride (2) et hybride avec segmentation de mots (3). Le gain le plus important est, comme espéré, celui de la classe Ouverture. La méthode globalement meilleure est la 3 (hybride), qui utilise le plus d'information parmi toutes.

5.1.5 Commentaires sur les résultats

Performance Voici quelques commentaires à propos des taux d'erreur, toujours par rapport à la méthode structurelle (de référence) :

- la méthode 2 (hybride) a obtenu le meilleur résultat global (5,53% d'erreur, une diminution de 25% par rapport aux 7,44% initiaux) ;

- la classe Ouverture a été, comme espéré, celle où la réduction de l'erreur a été la plus importante : au moins 50% de réduction dans toutes les méthodes, presque 66% dans le meilleur cas (méthode 2) ; les trois nouvelles méthodes ont plus de 90% de rappel sur cette classe, contre moins de 82% pour la méthode structurelle ;
- les classes physiquement les plus proches de l'ouverture ont été les plus avantagées (diminution en considérant la méthode 2) : Corps (47%), Objet (25%), Date/Lieu (13%) et Coordonnées Destinataire (16%), et enfin Coordonnées Expéditeur (13%) ;
- la méthode 3 (hybride avec segmentation) a une performance globale meilleure que la structurelle, mais moins bonne que les deux autres (due principalement à des erreurs de segmentation des mots).

Particularités de chaque méthode L'intérêt d'avoir ces trois méthodes distinctes, en plus de comparer les taux d'erreur, est de pouvoir comparer les caractéristiques propres à chacune, qui leur donnent des avantages particuliers selon le contexte :

- la méthode 1 (stochastique) a un avantage en termes de simplicité de la grammaire, car elle n'utilise pas la règle structurelle destinée à l'analyse du corps de texte (qui, comme mentionné précédemment, est composée de huit variantes avec des conditions complexes à tester). Cette règle est remplacée par une autre, qui se déroule en deux étapess : d'abord, nous filtrons les lignes non relevantes, telles que les lignes trop longues (qui dans environ 95% des cas ne contiennent pas d'ouverture) ; ensuite, nous appliquons le classifieur Ouverture sur chaque ligne à la fois, en ajoutant le score à l'analyse courante. Le mécanisme stochastique se charge alors de trouver la meilleure solution ;
- la méthode 2 (hybride) agrège le plus d'information possible : la connaissance sur le contenu ainsi que l'information structurelle (la règle originale). Son résultat excellent est conséquence de la combinaison de ces sources d'information ;
- la méthode 3 (avec segmentation), malgré son taux d'erreur un peu élevé dû aux erreurs de segmentation, sert d'exemple à la généralisation des méthodes basées sur le contenu vers les autres classes : pour n'utiliser qu'un seul mot au lieu de la ligne entière, elle peut être facilement transposée à d'autres lignes qui contiennent des mots-clés : objet, téléphone, référence, etc.

Surcoût en temps de calcul Une dimension qui n'est pas capturée par les données liées aux taux d'erreur est le surcoût en temps de calcul : l'introduction des classifieurs au sein de l'analyse correspond à l'injection d'une nouvelle source d'information, et le traitement de cette information exige plus de temps. En ignorant les surcoûts de nature technique (dus à l'implémentation et non aux méthodes elles-mêmes), nous avons deux sources majeures d'augmentation du temps de calcul : le traitement de chaque ligne par un (ou plusieurs) classifieurs, ainsi que l'exploration de toutes les solutions.

Pour diminuer le surcoût dû au traitement des lignes par les classifieurs, nous avons deux aspects qui permettent d'établir des compromis liés au temps de calcul :

- utilisation de contraintes structurelles pour élaguer l'analyse (comme déjà fait pour la ligne d'ouverture) : l'ajout de règles permet de contraindre l'analyse et donc d'éviter des appels inutiles aux classifieurs ; mais cela exige plus de temps de réflexion du concepteur. Le compromis s'établit donc entre le temps de l'humain et le temps de la machine ;
- segmentation des mots : l'utilisation d'un mécanisme de segmentation permet la réduction de l'image qui sera analysée par le classifieur ; en plus, il limite la taille et la longueur des mots

du dictionnaire utilisé par le classifieur. Par contre, il y a moins d'information à utiliser, ce qui risque d'augmenter l'erreur. Le compromis s'établit entre le temps de calcul et le taux d'erreur (défini par la quantité d'information).

Ces deux aspects, ainsi que d'autres paramètres des classifieurs eux-mêmes, permettent de trouver le point d'équilibre entre temps de calcul, temps de conception et taux d'erreur, si jamais le premier devient trop important.

Conclusion : améliorations significatives, mais quelques aspects non validés La validation de la méthode stochastique sur les documents manuscrits de la base RIMES a permis de vérifier que la méthode apporte des résultats de bonne qualité et allège la description de la grammaire, avec une grande tolérance par rapport aux ambiguïtés. Toutes les méthodes testées apportent une amélioration du taux de reconnaissance ; en plus, chacune a des particularités qui peuvent la rendre utile pour d'autres raisons : simplification de la grammaire ou généralisation de la méthode vers d'autres types de lignes.

Cependant, il n'y a pas eu de combinaison de scores et de branches d'analyse à plusieurs niveaux, pour que nous puissions avoir une idée de la complexité en termes de chemins à explorer. L'opérateur d'analyse stochastique a été placé dans un niveau relativement bas de l'arbre d'analyse, ce qui exige une validation complémentaire par rapport à cet aspect. L'analyse des documents d'archives (partie 5.2) permettra d'effectuer des considérations plus précises par rapport au temps de calcul et au coût mémoire de l'analyse stochastique avec un nombre plus importat de solutions à explorer.

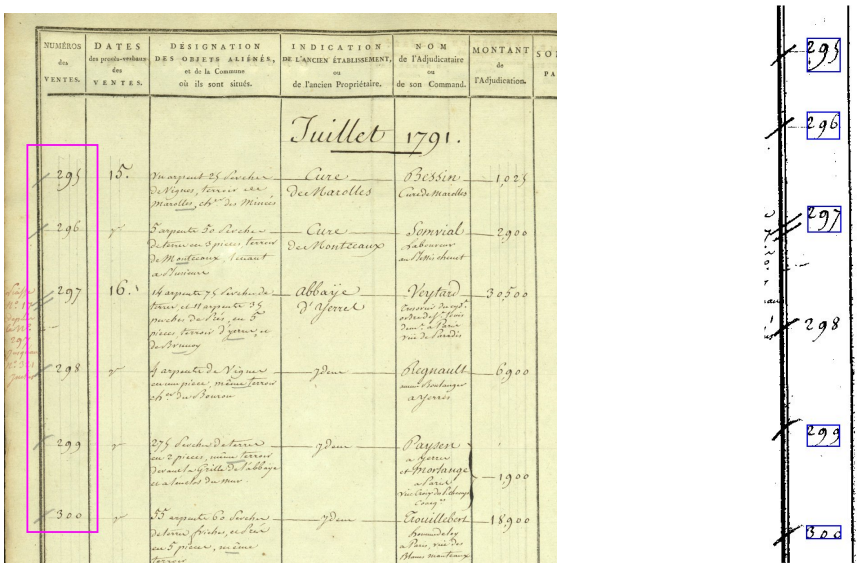
5.2 Considérations sur la performance : séquences de numéros de documents d'archives

L'analyseur stochastique de DMOS-P a été validé sur un autre type de document : des pages d'archives [2], plus spécifiquement des pages de registres de ventes de l'époque de la Révolution Française. Ces pages, dont un exemple est présenté dans la figure 5.5a, contiennent des tableaux avec des listes d'items manuscrits. Nous nous concentrons sur un aspect particulier de ces pages, les numéros de vente associés à chaque registre.

5.2.1 Description de la tâche de reconnaissance

Nous avons, du côté gauche des pages de ventes, des numéros de séquence des registres de ventes (détail dans la figure 5.5b), qui ont la particularité d'être fortement contraints les un par rapport aux autres : normalement, chaque numéro est le successeur direct du numéro qui se trouve avant lui, sauf s'il y a des numéros répétés (par exemple, « 111 bis » reconnu en tant que « 111 » une deuxième fois). Notre objectif est de valider les numéros reconnus (grâce à une liste de résultats fournie par un classifieur de chiffres manuscrits), en s'assurant qu'ils respectent les contraintes de succession ; ces contraintes nous permettent d'obtenir des taux de reconnaissance extrêmement élevés. En outre, elles permettent d'inférer, dans quelques cas, la présence d'erreurs de localisation (comme le numéro 298 dans la figure 5.5b). Notre liste de résultats à analyser porte sur une soixantaine de pages contenant 616 lignes ; notre liste d'hypothèses fournie par le classifieur contient jusqu'à 10 hypothèses par ligne (pour un total de 6156 numéros).

2. Nous remercions les Archives Départementales des Yvelines pour leur collaboration avec l'équipe IMADOC, qui nous a permis d'étudier ces documents.



(a) Extrait d'une page de ventes

(b) Détail des numéros de séquence

FIGURE 5.5: Exemple d'une page de registre de ventes (a), avec un détail des numéros de vente (b) binarisés segmentés (298 n'a pas été localisé ni segmenté).

Numéro (vérité)	Hypothèses (étiquette/score)									
295	395	0,462	295	0,464	595	0,499	495	0,532	895	0,539
296	296	0,000	206	0,983	266	0,983	246	0,983	286	0,984
297	2297	0,364	5297	0,393	4297	0,424	8297	0,439	3297	0,453
299	2929	0,376	2919	0,416	2959	0,425	229	0,444	2949	0,457
300	300	0,001	320	0,939	390	0,939	340	0,939	350	0,939

FIGURE 5.6: Scores d'après les résultats du classifieur pour la page de la figure 5.5.

Par la suite, nous supposons que la segmentation des chiffres dans la page ait déjà été traitée par un autre mécanisme ; nous ne traitons que les résultats du classifieur comme dans le tableau 5.6. Nous pouvons voir dans ce tableau que le meilleur score n'est pas toujours attribué au bon numéro, mais le contexte permet de détecter si les contraintes de succession ne sont pas respectées. Par exemple, si nous choisissons l'hypothèse 395 pour la première ligne au lieu de 295, aucune solution n'est viable pour la ligne contenant 296. Nous cherchons donc à optimiser la séquence entière en minimisant le score global. Le classifieur ne rend pas toujours, parmi les N meilleures solutions trouvées, celle qui est correcte (dans la figure 5.6, c'est le cas pour les lignes 297 et 299) ; dans ce cas, nous nous contentons d'indiquer que le numéro n'a pas été trouvé. Même si plusieurs numéros consécutifs ne sont pas reconnus correctement (dans l'exemple, tous entre 297 et 299), le contexte permet d'éventuellement récupérer la bonne séquence.

5.2.2 Définition d'une grammaire stochastique pour les numéros de séquence

Idéalement, la description d'une grammaire pour cette tâche est très simple (figure 5.7) : un `FIND_BEST_FIRST` appliqué à la racine assure que toutes les solutions possibles sont explorées et la meilleure est rendue en résultat. `verifierContrainteSequentialite` élimine les solutions ne respectant pas la croissance de la séquence. Si aucune hypothèse fournie par le classifieur n'est valide, nous ajoutons un « -1 » à la place avec un score légèrement supérieur au pire résultat trouvé. Cela évite l'application de la contrainte de succession, qui forcerait un échec sinon. Pour l'extrait de la figure 5.5, nous avons la séquence $[295, 296, -1, -1, 300]$.

```
recoNumeros Numeros ::=
        FIND_BEST_FIRST(Ns\(recoNumerosRec [0] Ns)) FOR (Numeros).

recoNumerosRec NumsIn NumsOut ::=
        reconnaitreProchainNumero N,
        append NumsIn N NumsTmp,
        verifierContrainteSequentialite NumsTmp,
        recoNumerosRec NumsTmp NumsOut.
recoNumerosRec Nums Nums. % Cas de base

reconnaitreProchainNumero N ::=
        recoNumero N Score, % Prend une hypothèse avec son score associé
        ADD_SCORE(Score).
```

FIGURE 5.7: Grammaire « idéalisée » pour l'analyse des numéros de séquence.

Évidemment, cette solution est infaisable : le nombre de possibilités à tester est dans le pire cas l'ordre de M^{N+1} solutions, où M est le nombre de lignes contenant des numéros et N est le nombre d'hypothèses (en plus de l'hypothèse nulle, « -1 ») par numéro rendues par le classifieur. Dans notre cas de test, nous avons $N = 10$ et $M = 616$. Même si la majeur partie de ces solutions sera élaguée par la contrainte de succession, il restera un nombre trop élevé à évaluer.

La grammaire a donc eu besoin d'être modifiée pour diminuer le nombre de solutions à explorer :

- la portée du `FIND_BEST_FIRST` a été limitée à un nombre réduit d'éléments ; il est donc appelé plusieurs fois et rend en sortie un ensemble de numéros « validés », considérés fiables et qui peuvent être utilisés lors de l'application de la contrainte de succession. Si ce n'était pas le cas, un numéro incorrect trop élevé (par exemple, 2297 au lieu de 297) forcerait l'échec des numéros qui suivent ;
- un ensemble de numéros (quelques uns déjà validés, d'autres pas encore validés) de taille fixe est utilisé comme contexte d'analyse ; cette « fenêtre glissante » permet de choisir un bon compromis entre la séquence incorrecte maximale tolérée et la quantité de solutions à explorer ; ce paramètre est appelé T_F (taille de la fenêtre) ;
- la différence maximale entre deux éléments voisins (DV_{Max}) ne doit pas dépasser un seuil fixé ; cela permet d'éviter des séquences de numéros cohérents entre eux mais incohérents par rapport au reste. Par exemple, la séquence $[295, 296, 2297, 2929]$ est rejetée si $DV_{Max} < 2001$. Dans l'absence de ce paramètre, la taille de la fenêtre glissante doit être plus grande pour éviter des erreurs locaux. Ce numéro est d'autant plus important que, dû au bruit dans l'image, il arrive que le classifieur ajoute un chiffre aux numéros reconnus (comme le « 2 »

précédent), et cela à plusieurs numéros consécutifs, ce qui augmente les chances d'une erreur « contextualisée ».

Le réglage de ces deux derniers paramètres permet d'établir les seuils minimaux pour avoir une reconnaissance maximale : celle qui, à chaque fois que le numéro correct est présent dans la liste d'hypothèses, il est rendu en résultat. Dans le meilleur des cas, cette séquence est obtenue en moins de 4 secondes avec $T_F = 3$ et $DV_{Max} = 5$; en ajoutant une marge de sécurité pour des documents plus bruités, nous pouvons considérer les valeurs $T_F = 10$ et $DV_{Max} = 15$ comme assez sûres. Avec ces valeurs, nos données de test (les 616 lignes) sont explorées en 73 secondes.

5.2.3 Résultats de l'évaluation : limitations et nouvel opérateur

Les principales différences entre cette utilisation de `FIND_BEST_FIRST` et celle de la grammaire de courriers manuscrits, et qui ont permis d'évaluer d'autres aspects du mécanisme stochastique, sont :

- les limitations mémoire imposées par l'implémentation du mécanisme de collecte de solutions : même si nous modifions la grammaire pour utiliser la fenêtre glissante, nous ne pouvons plus nous passer des considérations à propos de l'utilisation mémoire, qui sont normalement transparentes pour l'utilisateur d'un langage de programmation logique. La liste construite en sortie (`NumsOut` dans l'exemple 5.7) est recopiée à chaque solution, comme discuté précédemment (partie 4.3), en générant un surcoût mémoire qui oblige le concepteur à réfléchir aux aspects procéduraux ;
- l'appel à l'opérateur `FIND_BEST_FIRST`, qui est fait plusieurs fois en séquence, et non avec un appel unique, comme avant. Même si le retour arrière n'est jamais utilisé (car nous n'avons besoin ici que de l'ordonnancement des solutions pour récupérer la meilleure), toutes les solutions alternatives sont gardées en mémoire par le mécanisme de collecte de solutions. Il y a donc un gaspillage de mémoire : le changement des paramètres dans la partie précédente amène à une utilisation mémoire deux fois plus importante, quand en réalité aucune des solutions alternatives n'est nécessaire. Ici, une coupure juste après le `FIND_BEST_FIRST` serait suffisante car nous n'utilisons que la première solution, mais si nous avions besoin d'une parmi les N meilleures solutions ($N > 1$), alors la coupure ne pourrait pas être utilisée. Pour résoudre ce problème, nous pouvons toutefois utiliser l'opérateur `FIND_N_BEST`, décrit juste après. Son implémentation est triviale et permet d'éviter un gaspillage mémoire trop important.

Opérateur FIND_N_BEST_FIRST Cet opérateur a la signature suivante :

```
FIND_N_BEST_FIRST(+N : int, -Valeur : T) FOR (+RegleT : RegleEPF_T)
```

La seule différence par rapport à l'opérateur `FIND_BEST_FIRST` est l'ajout d'un paramètre entier `N` qui définit le nombre maximum de solutions à garder à chaque appel. Il n'est donc pas possible d'effectuer un retour arrière plus de `N` fois, même si plusieurs solutions existent.

Pour l'instant, cet opérateur ne présente un intérêt que pour l'économie de mémoire avec des appels successifs à `FIND_BEST_FIRST` ; à long terme, cependant, il peut profiter d'une amélioration en termes de temps de calcul : avec l'insertion d'heuristiques et d'une recherche intelligente des solutions, l'algorithme de recherche *N-best A** peut être appliqué pour obtenir une solution plus efficace en temps de calcul.

Bilan de l'application sur les documents d'archives Nous pouvons conclure que cette grammaire d'analyse de numéros de séquence, beaucoup plus simple conceptuellement que la gram-

maire d'analyse de courriers manuscrits, est polluée par les limitations du mécanisme d'analyse stochastique, à savoir : l'absence d'heuristiques pour guider la recherche ; l'impossibilité de ne garder qu'un nombre fixe de solutions par appel au prédicat, pour éviter le cumul d'alternatives qui ne seront jamais explorées ; et le surcoût de copie des éléments instanciés à l'intérieur du prédicat `FIND_BEST_FIRST`, qui est basé sur une implémentation incompatible avec l'unification, et qu'impose donc une vision procédurale dans la description de la grammaire.

Malgré ces limitations, la grammaire permet de profiter des contraintes de succession pour trouver la solution maximale correctement ; le réglage des paramètres permet la récupération d'une solution correcte assez rapide avec une bonne marge de sécurité. En plus, cet usage différent nous a inspiré la création d'un opérateur alternatif adapté à l'économie de mémoire.

Chapitre 6

Conclusion

L'analyse de la structure de documents à partir de méthodes syntaxiques possède plusieurs caractéristiques intéressantes, telles que la décomposition naturelle de la structure dans des sous-structures plus simples, de façon analogue à leur construction. Cependant, la présence de bruit et d'ambiguïtés dus au traitement des images et à la diversité des structures possibles (qu'un humain n'a aucune difficulté à traiter, mais qui sont plus délicates pour un système automatique) exige des considérations particulières ; les grammaires formelles ont été conçues dans un cadre plus strict et leur adaptation à la reconnaissance de documents n'est pas immédiate. Une évolution importante est l'incorporation de la bidimensionnalité des documents dans l'analyse, via des opérateurs de position, par exemple. Un autre pré-requis est l'incorporation d'attributs à la grammaire, pour incorporer les caractéristiques des éléments terminaux, telles que l'incertitude et des données numériques (taille des caractères, longueur des mots, etc.).

Pour augmenter la performance de l'analyse grammaticale, nous pouvons y ajouter de l'information de plusieurs façons :

- via l'insertion de plusieurs règles d'analyse, assez précises et capables de traiter les situations exceptionnelles ; mais l'effort demandé au concepteur peut finir par diminuer les bénéfices de l'approche grammaticale ;
- via l'insertion d'information provenant d'autres sources (telles que le contenu), qui permet au système d'effectuer une analyse plus robuste, en profitant des mécanismes d'extraction de contenu déjà existants tels que les classifieurs.

Comme nous l'avons vu, les deux approches (symbolique pour l'analyse structurelle, numérique pour l'analyse du contenu) possèdent des différences qui exigent l'implantation d'un pont entre les deux. Ce pont, établi à l'aide des grammaires stochastiques, augmente l'expressivité de la grammaire, car le concepteur dispose alors de plusieurs sources d'information pour exprimer la connaissance.

L'intégration de ce mécanisme à la méthode DMOS-P, après l'étude du fonctionnement de l'analyseur et des possibilités offertes par le langage sur lequel il est basé (λ-Prolog), a été effectuée sans nécessiter d'intervention majeure au sein du compilateur, et surtout de manière très peu invasive : les deux mécanismes cohabitent sans perturbations.

Du point de vue du concepteur de la grammaire, les opérateurs `ADD_SCORE` et `SELECT_SCORE` sont assez flexibles pour être utilisés dans plusieurs contextes ; l'utilisation des scores en tant que coûts enlève des restrictions et des soucis en termes de normalisation ; et la composition des coûts par addition facilite la conception « intuitive » (sans entraînement) d'une grammaire stochastique simple comme quelques unes rencontrées dans l'état de l'art.

En termes quantitatifs, notre mécanisme stochastique a permis d'obtenir un gain de performance sur une grammaire qui avait déjà un taux de reconnaissance élevé. En simplifiant la grammaire, nous avons une augmentation du rappel (de 81% à 91%, l'équivalent à une diminution de 50% du taux d'erreur) pour la classe ciblée, avec des répercussions positives sur l'ensemble du document : le rappel global est passé de 92,5% à 94% (20% de diminution du taux d'erreur). En outre, en conservant les règles existantes, notre mécanisme est capable d'utiliser cette information extra pour augmenter encore sa performance : 93,6% de rappel sur la classe ciblée (une diminution de 66% du taux d'erreur) et 94,47% de rappel total (taux d'erreur diminué de 25%).

L'un des inconvénients de la méthode stochastique est le coût plus important en mémoire et en temps de calcul, mais nous avons déjà une solution partielle à ce problème : le choix du point d'insertion de l'opérateur stochastique, qui fournit un mécanisme de contrôle au concepteur de la grammaire. Des solutions plus sophistiquées exigent des modifications profondes au sein du compilateur qui impacteront les grammaires déterministes, et l'absence d'heuristiques autres que le score diminuent l'intérêt de cette approche. Néanmoins, pour des situations telles que celle des documents d'archives, ces solutions peuvent s'avérer utiles.

Nous avons donc validé l'intérêt, l'applicabilité, le gain de performance et les limitations de l'analyse stochastique dans le domaine de l'analyse structurelle des documents. Nous pouvons repenser la description des grammaires d'analyse de la structure de documents d'une manière plus élargie et surtout plus naturelle. Nous avons donc une solution plus dynamique au paradoxe de Sayre [Say73] (« pour segmenter une entité, il faut la localiser, mais pour la localiser, il faut la reconnaître ») grâce à l'exécution interposée des deux mécanismes, syntaxique et numérique, de manière transparente.

Perspectives

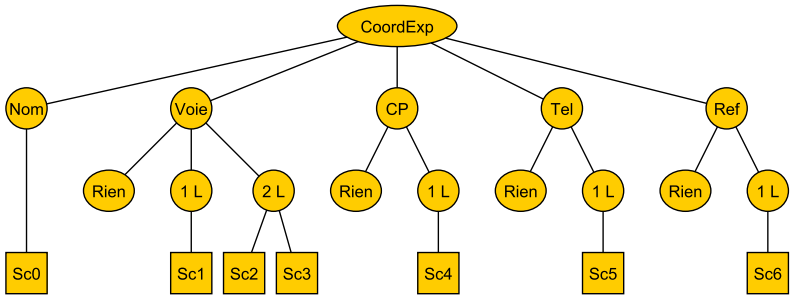
Nous citons ici deux aspects différents sur lesquels des travaux futurs peuvent porter : la définition automatique de probabilités pour les structures complexes et l'accélération de l'exploration de l'arbre d'analyse.

Normalisation des scores et apprentissage

Actuellement, l'association de scores aux feuilles de l'arbre d'analyse est relativement simple : soit nous prenons directement l'incertitude des classifieurs, soit nous définissons une métrique (par exemple, la distance moyenne entre les composantes connexes d'une ligne) qui soit convertie vers un score. Pour l'intégration de ces informations dans un niveau plus élevé (un nœud intermédiaire), les travaux futurs devront étudier les questions de normalisation et équivalence entre arbres de tailles différentes.

Pour illustrer la complexité de ce problème, prenons comme exemple l'arbre de la figure 6.1a, dans le cadre de l'analyse de courrier manuscrits. Elle correspond à un ensemble de règles assez simples et au même temps assez flexibles pour analyser les blocs de coordonnées d'expéditeur : sauf pour le nom, toutes les autres lignes sont optionnelles ; et dans le cas de la voie, elle peut apparaître sur 1 ou 2 lignes. Il y au total jusqu'à sept scores qui peuvent remonter, mais dû à la présence d'éléments optionnels, si nous ne prenons aucune précaution, la plupart de ces analyses sera toujours ignorée. La raison est que une branche qui fait appel à un classifieur aura un résultat supérieur à zéro, tandis que sa correspondante « absente » n'a aucun score par défaut, ce qui la rend toujours préférable à l'autre. Il faut donc penser à ajouter une pénalité « par défaut » à la branche sans lignes, mais la définition de cette valeur peut devenir assez arbitraire sans un mécanisme

d'apprentissage. Même si nous forçons la consommation de tous les terminaux pour contraindre les arbres acceptés, la combinaison de règles ambiguës génère un nombre exponentiellement grand d'arbres d'analyse et donc plusieurs configurations équivalentes (ou presque) en numéro de lignes, comme dans la figure 6.1b, où seul le contenu peut nous indiquer laquelle des solutions est la plus correcte. Si nous incluons des règles pour autoriser la permutation d'éléments (ce qui est nécessaire, par exemple, pour les lignes *référence* et *téléphone*), le contenu devient effectivement la seule source de désambiguïsation.



(a) Arbre d'analyse d'un bloc de coordonnées d'expéditeur.

(b) Deux blocs structurellement similaires, mais avec des contenus très différents : le deuxième contient les coordonnées du destinataire collées à celles de l'expéditeur.

FIGURE 6.1: L'arbre dans (a) peut être utilisé pour analyser les blocs dans (b), mais sa structure très ambiguë exige que les classifieurs soient assez performants pour éviter une analyse incorrecte.

Dans des cas relativement complexes (mais assez fréquents) comme celui-ci, la définition manuelle de coûts devient une tâche impossible. Un mécanisme d'apprentissage de paramètres au sein de DMOS-P permettrait l'extension de l'application de l'opérateur stochastique à des niveaux plus élevés de la grammaire, ce qui permettrait de la simplifier davantage.

Optimisation de l'exploration

Des travaux futurs peuvent aussi porter sur deux optimisations du mécanisme d'exploration de solutions : la recherche intelligente et le stockage d'arbres d'analyse partielles, pour éviter de les recalculer à chaque solution.

Pour la recherche intelligente, les scores pourraient servir en tant qu'heuristique : combinés à un mécanisme de limitation du nombre de solutions, tel que l'opérateur `FIND_N_BEST_FIRST`, ils pourraient amener à une variante de l'algorithme *N-best A**, qui éviterait l'exploration de solutions trop mauvaises. Cela exigerait des modifications au sein du compilateur EPF qui entraîneraient probablement une perte de performance pour l'analyse déterministe.

Pour le « cache » de solutions partielles, nous pourrions nous inspirer des *analyseurs tabulaires*. En effet, dans les grammaires de chaînes ou bidimensionnelles avec un ensemble fixe d'opérateurs de position (tels que *au-dessus*, *à côté de*, *entre*, etc.), il est relativement simple de définir une structure pour stocker l'état d'une analyse partielle. Cependant, la grande liberté de positionnement de DMOS-P rend cette tâche plus délicate.

Dans les deux cas, des travaux effectués sur l'optimisation de l'exploration permettraient aux concepteurs de grammaires d'oublier les aspects procéduraux du mécanisme, en simplifiant leur tâche de description des documents.

Bibliographie

[Art03] Thierry Artières. Poorly structured handwritten documents segmentation using continuous probabilistic feature grammars, 2003.

[BR92] Pascal Brisset and Olivier Ridoux. The architecture of an implementation of λ-prolog : Prolog/-mali, 1992.

[CD04] Gaurav Chanda and Frank Dellaer. Grammatical methods in computer vision : an overview. Technical report, Georgia Institute of Technology, Novembre 2004.

[Con93] A. Conway. Page grammars and page parsing. a syntactic approach to document layout recognition. In *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on*, pages 761–764, Octobre 1993.

[Coü01] Bertrand Coüasnon. DMOS : A generic document recognition method, application to an automatic generator of musical scores, mathematical formulae and table structures recognition systems. *Document Analysis and Recognition, International Conference on*, 0 :0215, 2001.

[FA77] K. S. Fu and J. E. Albus. *Syntactic pattern recognition : applications / edited by K. S. Fu ; with contributions by J. E. Albus ... [et al.].* Springer-Verlag, Berlin ; New York :, 1977.

[FGK06] John A. Fitzgerald, Franz Geiselbrechtinger, and Tahar Kechadi. Structural analysis of handwritten mathematical expressions through fuzzy parsing. In *ACST'06 : Proceedings of the 2nd IASTED international conference on Advances in computer science and technology*, pages 151–156, Anaheim, CA, États-Unis, 2006. ACTA Press.

[GCG+06] E. Grosicki, M. Carré, E. Geoffrois, E. Augustin, and F. Prêteux. La campagne d'évaluation rimes pour la reconnaissance de courriers manuscrits. In *Actes Colloque International Francophone sur l'Écrit et le Document (CIFED'06), Fribourg, Suisse*, pages 61–66, September 2006.

[GHTC10] Laurent Guichard, Alejandro Héctor Toselli, and Bertrand Coüasnon. A novel verification system for handwritten words recognition. In *International Conference on Pattern Recognition (ICPR 2010)*, 2010.

[Hav08] Christian Theil Have. Stochastic definite clause grammars. Master's thesis, University of Copenhagen, Août 2008.

[HNZ05] John C. Handley, Anoop M. Namboodiri, and Richard Zanibbi. Document understanding system using stochastic context-free grammars. In *ICDAR '05 : Proceedings of the Eighth International Conference on Document Analysis and Recognition*, pages 511–515, Washington, DC, USA, 2005. IEEE Computer Society.

[HRB] Serge Le Huitouze, Olivier Ridoux, and Pascal Brisset. Prolog/mali reference manual.

[IB00] Yuri A. Ivanov and Aaron F. Bobick. Recognition of visual activities and interactions by stochastic parsing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8) :852–872, 2000.

[JM00] Daniel Jurafsky and James H Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice-Hall, Upper Saddle River, NJ, 2000.

[JWS+95] Daniel Jurafsky, Chuck Wooters, Jonathan Segal, Andreas Stolcke, Eric Fosler, Gary Tajchman, and Nelson Morgan. Using a stochastic context-free grammar as a language model for speech recognition, 1995.

[LC08] Aurélie Lemaitre and Jean Camillerapp. *Introduction de la vision perceptive pour la reconnaissance de la structure de documents*. PhD thesis, Institut National des Sciences Appliquées de Rennes, 2008.

[LCC10] Aurélie Lemaitre, Jean Camillerapp, and Bertrand Coüasnon. Interest of perceptive vision for document structure analysis. In Bernice E. Rogowitz and Thrasyvoulos N. Pappas, editors, *Proceedings of SPIE*, volume 7527, page 752714. SPIE, 2010.

[LCL06] Aurélie Lemaitre, Bertrand Coüasnon, and Ivan Leplumey. *Using a Neighbourhood Graph Based on Voronoï Tessellation with DMOS, a Generic Method for Structured Document Recognition*, volume 3926 of *Lecture Notes in Computer Science*, pages 267–278. Springer Berlin / Heidelberg, October 2006.

[LHLR93] Serge Le Huitouze, Pascale Louvet, and Olivier Ridoux. Logic grammars and lambda-prolog. In *ICLP*, pages 64–79, 1993.

[Min98] Wolfgang Minker. Stochastic versus rule-based speech understanding for information retrieval. *Speech Commun.*, 25(4) :223–247, 1998.

[MRK03] Song Mao, Azriel Rosenfeld, and Tapas Kanungo. Document structure analysis algorithms : a literature survey. In Tapas Kanungo, Elisa H. Barney Smith, Jianying Hu, and Paul B. Kantor, editors, *Proceedings of SPIE*, volume 5010, pages 197–207. SPIE, 2003.

[Nag00] George Nagy. Twenty years of document image analysis in pami. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(1) :38–62, 2000.

[Nai86] Lee Naish. *Negation and control in Prolog*. Springer-Verlag New York, Inc., New York, NY, USA, 1986.

[NPH04] Stephane Nicolas, Thierry Paquet, and Laurent Heutte. Un panorama des méthodes syntaxiques pour la segmentation d'images de documents manuscrits. In *CIFED ex CNED - Colloque International Francophone sur l'Ecrit et le Document*, pages 237–242, 2004.

[Say73] K. Sayre. Machine recognition of handwritten words : A project report. *Pattern Recognition*, 5(3) :213–228, September 1973.

[SJ07] Noah A. Smith and Mark Johnson. Weighted and probabilistic context-free grammars are equally expressive. *Comput. Linguist.*, 33(4) :477–491, 2007.

[TI94] Y. Tateisi and N. Itoh. Using stochastic syntactic analysis for extracting a logical structure from a document image. In *Pattern Recognition, 1994. Vol. 2 - Conference B : Computer Vision and Image Processing., Proceedings of the 12th IAPR International. Conference on*, volume 2, pages 391–394 vol.2, Octobre 1994.

[Tru05] Éric Trupin. La reconnaissance d'images de documents : Un panorama. *Traitement du signal*, 22(3), 2005.