Ré-ordonnancement pour l'apprentissage de transformations de documents HTML

Guillaume Wisniewski, Patrick Gallinari

LIP6 — UMPC
8 rue du capitaine Scott
75015 Paris
{prénom.nom}@lip6.fr,
http://www-connex.lip6.fr

Résumé. Notre objectif est de transformer les documents Web vers un schéma médiateur XML défini a priori. C'est une étape nécessaire pour de nombreuses tâches de recherche d'information concernant le Web Sémantique, les documents semi-structurés, le traitement de sources hétérogènes, etc. Elle permet d'associer une structure sémantiquement riche à des documents dont le formats ne contient que des informations de présentation. Nous proposons de traiter ce problème comme un problème d'apprentissage structuré en le formalisant comme une transformation d'arbre en arbre.

Notre méthode de transformation comporte deux étapes. Dans une première étape, une grammaire hors-contexte probabiliste permet de générer un ensemble de solutions candidates. Dans une deuxième étape, ces solutaions candidates sont ordonnées grâce à un algorithme de ré-ordonnancement à base de perceptron à noyau. Cette étape d'ordonnancement nous permet d'utiliser de manière efficace des caractéristiques complexes définies à partir du document d'entrée et de la solution candidate.

1 Introduction

Le Web 2.0 a pour objectif de faciliter l'accès à l'information en représentant les documents Web par une structure sémantiquement riche et non par un traditionnel « sac de mots ». Cette structure est généralement définie par la représentation des documents sous forme d'arbres : des *éléments de contenu*, identifiés par la séquence étiquetée des feuilles de l'arbre, sont organisés selon une structure prédéfinie par un ensemble de *nœuds internes* représentant les relations entre éléments. Cette structure traduit les relations sémantiques ou logiques entre éléments de contenu. Les comparateurs de prix, le Web Sémantique sont des exemples de services fournis par le Web 2.0.

La plupart des documents du web utilisent des formats semi structurés comme le HTML, le XML, le PDF ou encore le WikiText. Ces formats permettent d'enrichir le texte à l'aide de balises et une interprétation directe de celles-ci permet de décrire les documents par un arbre, l'arbre DOM. Nous appellerons *structure syntaxique* cette structure directement liée à la manière dont l'information est codée. Les applications du Web 2.0 ne peuvent toutefois pas

tirer directement profit de cette structure : elles ont toutes besoin de connaître à priori la structure utilisée et ne sont donc capables de ne traiter que les documents respectant strictement un *schéma* qui leur est spécifique. Ce schéma définit les structures que peuvent avoir les documents. Il est peu probable que les documents Web respectent un schéma donné à priori : en général, ceux-ci proviennent de plusieurs sources hétérogènes, utilisant chacune des structures différentes. De plus, dans le cas du HTML leur format ne contient que des informations de mise en page peu informatif pour ces applications. L'utilisation directe de la structure syntaxique par les application du Web 2.0 est donc impossible. Pour les rendre utilisables, il faut transformer ces sources hétérogènes en un format médiateur spécifique de l'application. C'est l'objet du travail que nous présentons. Nous nous intéressons plus spécifiquement à la conversion automatique de documents HTML vers un format XML prédéfini. Cette problématique spécifique tire son intéret de la masse d'information présente sur le web sous un format HTML.

Bien que moins riche que celle de nombreux formats semi-structurés, l'information de mise en page présente dans le HTML fournit une information qui est exploitée quotidiennement par de nombreux utilisateurs, notamment pour faciliter leur navigation ou la recherche d'information. En effet, avec le développement des sites basés sur des systèmes de gestion de contenu (blogs, site de nouvelles, ...), de plus en plus de pages sont générées automatiquement à partir de bases de données. Par leurs régularités, la mise en page des documents reflète leur structure logique et permet d'identifier des éléments (un titre, un commentaire, ...) ainsi que des relations entre ceux-ci (on peut par exemple préciser l'auteur d'une sous-partie du document). Ce nouveau type d'information, directement lié à la présentation des documents, peut, par exemple, être utilisé pour organiser les commentaires des visiteurs d'un site en *threads* (Figure 1) ou par ordre chronologique .

FIG. 1 – *Extrait d'un thread de commentaires sur Slashdot. La mise en page nous permet d'identifier facilement chaque commentaire ainsi que les relations entre commentaires. Des méta-informations, comme le nom de l'auteur ou la date du commentaire, sont également immédiatement accessibles.*

Pour exploiter cette information additionnelle nous proposons de transformer les documents Web vers un format médiateur. Cette structure sera spécifiée par un schéma cible dépendant de l'application considérée. L'écriture manuelle de convertisseurs spécifiques à chaque source de documents et à chaque application est un travail présentant un grand risque d'erreur et qui est peu adapté à la richesse et à la nature dynamique du Web. Plusieurs solutions (Doan et al., 2003; Chung et al., 2002) utilisant l'apprentissage artificiel ont été proposées pour automatiser la transformation de documents semi structurés vers une structure définie par un schéma arbitraire : leur objectif est d'*associer* à un document d'entrée d^{in} le document d^{out} contenant les mêmes informations exprimées dans le schéma cible, après avoir observé un ensemble $\left(d_i^{in}, d_i^{out}\right)_{i=1}^{N}$ de documents exprimés à la fois dans leur structure d'origine et dans la structure cible. Une des approches les plus prometteuses, (Chidlovskii et Fuselier, 2005), a formulé cette tâche comme une *généralisation de l'analyse syntaxique* : le document d'entrée est représenté par une séquence d'observations correspondant aux feuilles du document HTML ; la structure du document de sortie est reconstruite à partir de cette représentation grâce à une grammaire hors-contexte décrivant le schéma de sortie. Dans cette approche, la structure du document d'entrée est cependant ignorée, même si, comme nous le montrerons, elle fournit une information indispensable pour déterminer la bonne structure de sortie.

Dans ce travail, nous proposons une approche plus générale qui permet de considérer des caractéristiques arbitraires décrivant à la fois la structure de sortie et la structure d'entrée des documents. Nous commencerons par décrire un cadre général de transformation de documents fondé sur des techniques d'apprentissage structuré (paragraphe 2), puis nous détaillerons notre modèle (paragraphe 3). Nous présentons enfin un ensemble d'expériences prospectives sur plusieurs corpus de documents réels (paragraphe 4).

2 La problématique de restructuration

Pour transformer automatiquement un document, nous considérons la tâche générale de restructuration : étant donné un schéma arbitraire, nous souhaitons transformer un document d'entrée d^{in} en un document de sortie d^{out} conforme à ce schéma. Cette transformation d'arbres peut inclure différents types d'opérations : réorganisation d'éléments (une bibliographie peut être présentée par thématique ou par année), regroupement d'éléments (le nom et le prénom d'un auteur peuvent être stockés dans un élément ou dans deux), etc. La tâche de restructuration revient à identifier dans un document les éléments pertinents et, à déterminer récursivement les relations entre ces éléments. Sur l'exemple de la Figure 2, l'objectif est ainsi d'identifier les noms d'acteurs et les noms de personnages, puis de déterminer le rôle de chaque acteur. Les éléments et les relations à extraire sont, tous deux, définis par un schéma cible.

Automatiser cette transformation revient à apprendre une fonction f, tel que $f(d^{in}) = d^{out}$. La mise en correspondance de documents est un problème généralement sous-déterminé, puisque plusieurs documents de sortie peuvent être *compatibles* avec le document d'entrée. Ainsi, dans l'exemple du casting, n'importe quel acteur peut, à priori, jouer n'importe quel rôle. Pour déterminer l'unique restructuration correspondant à d^{in}, nous définissons une fonction paramétrée par w, $F(d, d^{in}; w)$ permettant d'évaluer la qualité d'une solution candidate d. Cette fonction nous permet d'ordonner les éléments de $\mathcal{D}(d^{in})$, l'ensemble des *restructurations potentielles* constitué par tous les documents d respectant le schéma cible et contenant les informations de d^{in}. La tâche de restructuration correspond alors à la recherche de la re-

structuration potentielle de plus grand score :

$$d^{out} = \underset{d \in \mathcal{D}(d^{in})}{\operatorname{argmax}} F(d, d^{in}; w) \tag{1}$$

Dans l'Équation 1, l'argmax traduit le parcours de l'espace de toutes les restructurations potentielles $\mathcal{D}(d^{in})$ pour rechercher la meilleure solution. Cet ensemble doit inclure l'ensemble des segmentations et des réorganisations des nœuds de contenu (notamment les permutations, fusions et séparations de nœuds) et tous les arbres compatibles avec cette nouvelle organisation. La taille de l'espace de recherche est donc exponentielle par rapport au nombre de nœuds de contenu : en se limitant aux permutations de n nœuds, l'espace de recherche contient déjà $n!$ éléments.

Pour trouver la meilleure solution en un temps *raisonnable*, plusieurs sources d'informations doivent être considérées pour élaguer l'espace de recherche : le contenu du document source, sa structure et la définition du schéma cible. Ainsi, sur l'exemple de la Figure 2, on peut dire que la restructuration est *guidée* par la structure du document d'entrée — les deux premières feuilles décrivent le même personnage, puisqu'ils ont le même parent — et *contrainte* par la structure cible — chaque personnage est composé d'un acteur et d'un nom : l'information apportée par les structures d'entrée et de sortie sont essentielles pour la tâche de restructuration.

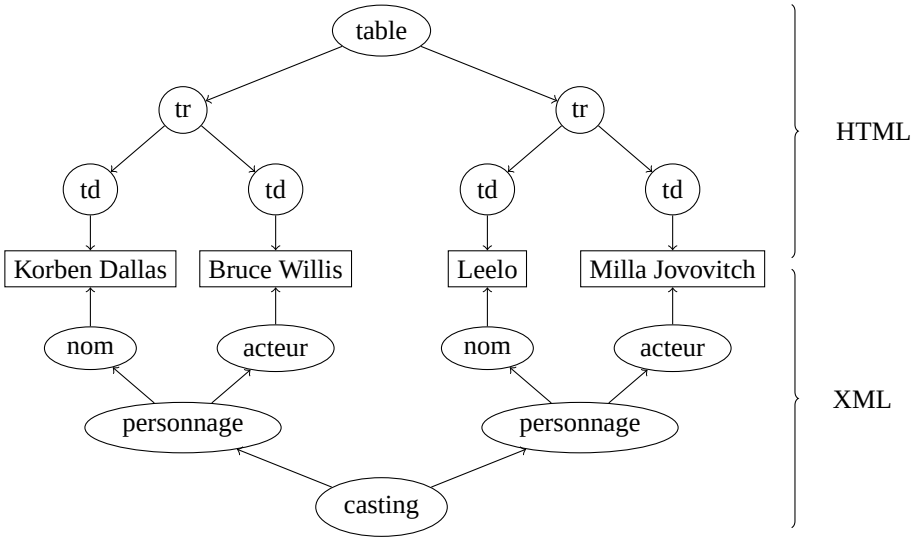

FIG. 2 – *Exemple de transformation d'un fragment de page HTML : l'objectif est d'identifier les noms d'acteur et les noms de personnages, puis de déterminer le rôle joué par chaque acteur.*

C'est pourquoi, nous choisissons de faire dépendre F d'une *représentation jointe*, $\phi(d^{in}, d)$ du document d'entrée et de sa restructuration potentielle. Cette représentation jointe comporte toutes les caractéristiques permettant de sélectionner la meilleure solution. Elle inclut notam-

ment des *caractéristiques globales* (qui ne peuvent être déterminées qu'en considérant le document dans son ensemble) comme le nombre d'occurrences d'une étiquette et des *dépendances à longue distance*.

Dans la suite, nous supposerons que F est linéaire par rapport à ϕ : $F(d, d^{in}; w) = \langle \phi(d^{in}, d), w \rangle$. Avec cette hypothèse, la tâche de restructuration peut être vue comme un problème d'apprentissage structuré (Tsochantaridis et al., 2004). L'apprentissage structuré est une généralisation de l'apprentissage multi-classe permettant de traiter des problèmes dans lesquels les entrées et les sorties peuvent être décomposées en un ensemble de sous-parties inter-dépendantes. Utiliser des techniques d'apprentissage structuré est intéressant puisque ce formalisme nous fournit plusieurs méthodes pour apprendre à partir de caractéristiques arbitraires décrivant les structures d'entrée et de sortie. Les algorithmes de l'apprentissage structuré souffrent en général d'une complexité élevée. Cela prohibe leur utilisation pour des tâches complexes comme la tâche de restructuration et pour traiter des corpus de grande taille. Nous allons maintenant présenter une méthode d'apprentissage structuré qui présente l'avantage d'avoir une complexité suffisamment faible pour pouvoir être appliquée au problème de la restructuration de corpus de grande taille.

3 Modèle d'apprentissage

La méthode que nous proposons repose sur l'observation suivante : l'apprentissage structuré, tel qu'il est décrit par l'Équation 1, nécessite de construire un ensemble très grand de structures combinatoires et de retrouver celle de plus grand score. Nous proposons de considérer ces deux étapes séquentiellement. Plus précisément, l'algorithme proposé enchaîne deux étapes :

une étape de génération qui va construire $GEN(d^{in})$ un ensemble de N solutions candidates. Ce processus repose sur des hypothèses d'indépendance fortes entre les éléments des documents et sur l'utilisation d'un ensemble restreint de caractéristiques pour permettre une construction efficace des solutions candidates à l'aide d'un algorithme basé sur la programmation dynamique.

une étape d'ordonnancement qui va trouver la meilleure restructuration parmi les solutions candidates $GEN(d^{in})$ générées à l'étape précédente en considérant des caractéristiques arbitraires aussi bien du document d'entrée que de la solution candidate. En particulier, puisqu'on travaille l'univers restreint $GEN(d^{in})$, on pourra considérer des caractéristiques globales des arbres d'entrée et de sortie, ce que ne permet pas la programmation dynamique.

Traiter ces deux étapes de manière séquentielle permet de conserver les avantages de la programmation dynamique (construction efficace d'une sortie structurée, mais qui ne considére que des caractéristiques locales) tout en considérant des caractéristiques globales lors de la seconde étape pour sélectionner la meilleure restructuration. Cette approche ne fournit qu'une solution approchée de l'Équation 1, puisque seules les solutions les plus prometteuses de la première étape sont évaluées. Toutefois, elle a déjà montré son efficacité dans plusieurs tâches de langue naturelle (Collins et Koo, 2005). Nous allons maintenant détailler ces deux étapes.

3.1 Génération des solutions candidates

La première étape de notre modèle a pour objectif de construire la structure du document de sortie à partir de la séquence des nœuds de contenu $\mathbf{c} = (c_1, ..., c_{\#\mathbf{c}})$. Cette construction peut se faire facilement avec des algorithmes d'analyse syntaxique (Jurafsky et Martin, 2000).

L'utilisation de ces algorithmes est d'autant plus intéressante que les structures arborescentes des documents semi structurés se modélisent naturellement par des grammaires hors-contexte (Chidlovskii et Fuselier, 2005). En effet, celles-ci permettent de décrire simplement des structures récursives à la fois horizontale (une section comporte plusieurs sous-sections) et verticale (il est possible de faire des listes de listes). De plus, elles permettent aussi de formaliser l'observation suivante : on peut identifier un élément soit par son contenu (une taille est définie par un nombre et une unité), soit par les éléments qui le constituent (une date est composée d'un jour, d'un mois et d'une année, même si la représentation et l'ordre de ces éléments peuvent varier).

L'utilisation d'une version probabilisée de cette grammaire (PCFG) permet, en outre, de caractériser les régularités et la variabilité du schéma cible : il est possible, par exemple, de modéliser le fait qu'une section regroupe un ensemble de sous-sections, et que, le nombre de sous sections suit une distribution donnée.

Formellement, une PCFG décrivant un document semi structuré est définie par le quintuplet $\mathcal{G} = \langle \Sigma, \Lambda, R, S, P_c \rangle$ où :

- Σ est un ensemble de non-terminaux décrivant les étiquettes des nœuds internes (qui correspondent aux relations) ;
- Λ est un ensemble de terminaux définissant les étiquettes des feuilles (qui correspondent aux nœuds de contenu) ;
- S, un élément de Σ, est le symbole initial décrivant la racine de l'arbre ;
- R est un ensemble de productions dont chaque élément est associé à une probabilité. Chaque production est une relation de $\Sigma \times (\Lambda \cup \Sigma)^*$ qui décrit les règles de composition : la règle $html \rightarrow head\ body$ indique qu'un élément $html$ regroupe (dans cet ordre) un élément $head$ et un élément $body$;
- P_c est un *modèle de contenu* qui défini une distribution de probabilité sur Λ pour chaque nœud de contenu. Intuitivement, cette distribution permet de rajouter un ensemble de productions du type $\lambda \rightarrow c_i$ pour tous les éléments λ de Λ et tous les éléments de contenu c_i. Ces productions permettent de déterminer l'étiquette des feuilles.

La Figure 3 montre un exemple de schéma cible et de la PCFG qui lui est associée.

Une PCFG permet d'associer une probabilité à chaque structure de sortie t. Cette probabilité mesure la *compatibilité* entre la structure arborescente et la séquence d'observations. Elle est définie par :

$$p(t, \mathbf{c}) = \prod_r p(r)$$

où r est l'ensemble des productions utilisées pour construire l'arbre t, $\mathbf{c}$ est la séquence des nœuds de contenu du document. L'estimation des probabilités $p(r)$ sera détaillée dans le prochain paragraphe. La Figure 4 donne un exemple du calcul de cette probabilité.

Un algorithme d'analyse syntaxique permet de reconstruire efficacement les N-meilleures restructuration associées à une séquence de nœuds de contenu. Nous avons utilisé une exten-

```
<!ELEMENT NEWS (HEADER BODY)>
<!ELEMENT HEADER (author title | author title date)>
<!ELEMENT BODY (para | para para)>
```

$NEWS \rightarrow HEADER\,BODY$	$(s_1 = 1)$
$HEADER \rightarrow author\ title$	$(s_2 = 0,8)$
$HEADER \rightarrow author\ title\ date$	$(s_3 = 0,2)$
$BODY \rightarrow para\ para$	$(s_4 = 0,3)$
$BODY \rightarrow para$	$(s_5 = 0,7)$

P_c	c_1	c_2	c_3	c_4
para	0,3	0,3	0,5	0,4
author	0,3	0,2	0,1	0,3
title	0,3	0,1	0,2	0,1
date	0,1	0,4	0,2	0,2

FIG. 3 – *Exemple d'une partie d'un schéma décrivant la structure d'une nouvelle et de la PCFG qui lui est associée. Ce schéma spécifie 4 étiquettes pour les nœuds de contenu et 3 étiquettes pour les relations. Il indique aussi qu'un élément $BODY$ est composé soit de deux éléments $para$ soit d'un seul. L'observation des probabilités associées aux productions montre que ce dernier cas est moins fréquent. La table donne un exemple de distribution de probabilité P_c pour une suite de quatre nœuds de contenu (c_1, c_2, c_3, c_4)*

sion de l'algorithme CYK[1] (Jimnez et Marzal, 2000) pour construire les N meilleures solutions avec une complexité $\mathcal{O}(n^3 + N \cdot \frac{\#R}{\#\Lambda} \cdot n \cdot \log n^3)$. Généralement, dans nos expériences $N < n$, cette complexité est donc du même ordre de grandeur que celle de l'algorithme reconstruisant la meilleure solution, qui est en $\mathcal{O}(n^3)$.

Apprentissage des paramètres L'apprentissage se fait à partir d'un ensemble de documents exprimés dans le schéma cible. Les paramètres décrivant les éléments de R sont estimés par maximum de vraisemblance (Jurafsky et Martin, 2000). La probabilité d'une production $A \rightarrow \alpha$ est donnée par :

$$p(A \rightarrow \alpha) = \frac{\#\{A \rightarrow \alpha\}}{\sum_{A \rightarrow \beta \in R} \#\{A \rightarrow \beta\}}$$

où $\#\{A \rightarrow \alpha\}$ correspond au nombre d'apparition de cette production dans le corpus d'apprentissage, A est un élément de Σ et α de $\Lambda \cup \Sigma^*$.

Les probabilités de contenu sont estimées dans notre modèle par un classifieur maximisant l'entropie. Ce type de classifieur nous permet de prendre en compte facilement toutes les caractéristiques que nous jugeons pertinentes, sans nécessiter d'hypothèses d'indépendance entre celles-ci. Ces caractéristiques peuvent dépendre à la fois du nœud de contenu c et de l'étiquette λ. Nous pouvons donc définir aussi bien des caractéristiques décrivant le contenu des nœuds (nombre de majuscules, présence de chiffre, ...), leur contexte (le nombre de frères, la profondeur dans l'arbre, ...) et le type de données spécifié par le schéma. La Table 1 détaille une partie des caractéristiques utilisées.

Plus précisément, on a :

$$p(\lambda|c) = \frac{1}{Z_\alpha(c)} \exp \sum_i \alpha_i \cdot f_i(c, \lambda)$$

[1]L'algorithme CYK permet de retrouver efficacement la meilleure solution en utilisant la programmation dynamique

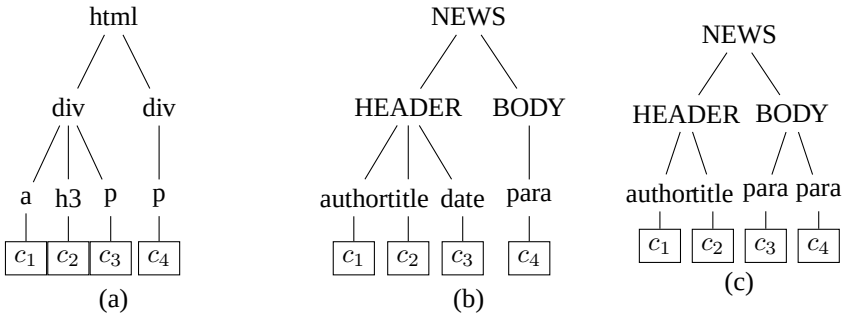

FIG. 4 – *Exemple de deux restructurations potentielles (les arbres (b) et (c)) de l'arbre (a). Avec la PCFG de la Figure 3, le score du document (b) est* $p(b|\mathbf{c}) = s_1 \cdot s_3 \cdot s_5 \cdot p(c_1|author) \cdot p(title|c_2) \cdot p(date|c_3) \cdot p(para|c_4) = 3,34 \cdot 10^{-4}$ *; le score du document (c) est* $1 \cdot 10^{-3}$.

où f est le vecteur de caractéristiques décrit au paragraphe précédent,λ une étiquette, $Z_\alpha(c)$ est un coefficient de normalisation, α le vecteur des paramètres à estimer et $\sum_i \alpha_i \cdot f_i(c, \lambda)$ le produit scalaire canonique.

Le vecteur des paramètres α a été estimé en utilisant le principe de maximisation de l'entropie : l'algorithme d'apprentissage détermine, parmi toutes les distributions compatibles avec les observations, celle qui fait le moins d'hypothèse sur les valeurs non observées.

caractéristiques de contenu	caractéristiques du contexte	caractéristiques sur le type de données
`contains-http`	`is-only-child`	`is-xs_string`
`begins-with-capitals`	`has-1-to-3-siblings`	`is-xs_duration`
`contains-number`	`is-descendant-of-title`	`is-xs_time`
`contains-1-to-5-spaces`		
...		

TAB. 1 – *Exemples de caractéristiques utilisées pour décrire les nœuds de contenu. La figure représente trois types de caractéristiques : contenu (e.g. une séquence contient "'http"'), contexte (e.g. ce nœud n'a pas de frère), type de données (e.g. type string selon la définition de la norme XML Schema).*

3.2 Étape d'ordonnancement

L'objectif de cette étape est d'apprendre la fonction $F(d, d^{in}; w)$, qui permet d'ordonner l'ensemble des solutions candidates. Ce score va notamment nous permettre de prendre en compte des caractéristiques globales sur d, et d^{in}. Le vecteur de paramètres w est estimé à partir d'un ensemble d'apprentissage constitué par :

– un ensemble de n documents, chaque document étant exprimé à la fois dans sa structure d'origine et dans la structure cible. Nous noterons $\mathcal{T} = \left(d_i^{in}, d_i^{out}\right)_{i=1}^{n}$ cet ensemble ;

– pour chaque élément de $\mathcal{T}$ les N solutions candidates : $GEN(d_i^{in}) = (d_{ij})_{j=1}^{N}$ qui sont construites par la première étape. Nous supposerons, sans perte de généralité, que $d_i 1$ est la meilleure restructuration pour d_1^{in}.

L'apprentissage est effectué par un perceptron à noyaux (Collins et Duffy, 2002; Collins et Koo, 2005). C'est une méthode simple mais efficace, qui permet, grâce à l'utilisation d'un noyau, de considérer des espaces de grandes dimensions. Le score calculé par le perceptron à noyau pour une solution candidate d associée au document d'entrée d^{in} est :

$$G(d, d^{in}) = \sum_{i,j} \alpha_{ij} \cdot k\left((d_{i1}, d_i^{in}), ((d, d^{in})\right) - k\left((d_{ij}, d_i^{in}), ((d, d^{in})\right)$$

avec k une fonction noyau qui sera explicitée dans le paragraphe suivant, α_{ij} les paramètres appris, d_i^{in} un exemple d'apprentissage, d_{i1} la meilleure restructuration associée à cet exemple parmi les n éléments de $GEN(d_i^{in})$ et $(d_{ij})_{2 \leq j \leq n}$ les $n-1$ autres solutions candidates.

3.3 Caractéristiques utilisées

Deux types de caractéristiques sont envisageables pour discriminer la meilleure solution candidate. Un premier type de caractéristiques décrit les dépendances à longue distance entre les nœuds de l'arbre, afin de décrire chaque nœud par un contexte plus riche que celui utilisé dans l'étape de génération. Un deuxième type de caractéristiques permet de mesurer une similarité entre le document d'entrée et la solution candidate. En effet, dans le cas de documents Web majoritairement textuel, certains groupes d'éléments doivent être conservés lors de la transformation, pour garder un sens.

C'est pourquoi nous allons utiliser une combinaison de deux noyaux :

$$k\left((d_{ij}, d_i^{in}), (d, d^{in})\right) = k_{tree}(d, d_{ij}) + k_{features}\left((d, d^{in}), (d_{ij}, d_{ij}^{in})\right) \quad (2)$$

L'addition de deux noyaux traduit la concaténation des deux espaces de caractéristiques.

Le premier noyau, k_{tree} est le *noyau d'arbre* de (Collins et Duffy, 2001) qui capture les dépendances à long terme entre les nœuds d'un arbre. Contrairement à une PCFG qui ne considère que les dépendances entre un nœud et ses fils, le noyau d'arbre considère l'ensemble de l'arbre pour modéliser le contexte d'un nœud : un arbre sera représenté par l'ensemble de ses sous-arbres. Le nombre de sous-arbres d'un arbre est exponentiel par rapport à la taille d'un arbre, mais, une solution basée sur la programmation dynamique permet de réaliser le comptage sans avoir à énumérer tous les sous-arbres.

Le second noyau utilisé, $k_{features}$ est un noyau RBF (Radial Basis Function). Il utilise des caractéristiques globales à la fois sur le document d'entrée et sur la solution candidate. Ces caractéristiques incluent :

– une comparaison entre le nombre de nœuds du document d'entrée et du document de sortie
– les *couvertures* communes au document d'entrée et au document de sortie. La couverture est définie, pour chaque nœud d'un arbre par la paire constituée de la position dans la séquence des feuilles de la première et de la dernière feuille du sous-arbre ayant ce nœud comme racine. De manière intuitive, les couvertures permettent de résoudre des problème comme ceux présentés Figure 4 ;

– des contraintes imposées par le schéma cible (par exemple un film ne peut avoir qu'un titre mais plusieurs réalisateurs). Ces contraintes sont déduites automatiquement du schéma cible.
– le score de la première étape

Les valeurs de ces caractéristiques sont combinées à l'aide d'un noyau puis incorporées dans le noyau global.

4 Expériences

Nous avons testé notre modèle sur deux corpus différents. Le premier corpus est un ensemble de nouvelles publiées sur un site traitant d'actualité informatique LinuxFr (`http://linuxfr.org`). Les pages du site ont été téléchargées et converties, à la main, en XML pour un schéma prédéfini. Chaque page correspond à une nouvelle et comporte un en tête regroupant les méta-informations (auteur, titre, date, ...), le corps de la nouvelle et plusieurs *threads* de commentaires des visiteurs du site. Le corps de la nouvelle comme les commentaires peuvent utiliser la plupart des tags HTML. La partie décrivant les commentaires des utilisateurs est très fortement structurée et présente un défi à la transformation du document : la structure logique des commentaires (i.e. : de quel commentaire un commentaire donné est-il la réponse) doit être reconstruite à partir de la structure du document d'entrée. Le corpus comporte 200 nouvelles, chaque nouvelle ayant, en moyenne, 70 nœuds de contenu et 50 nœuds internes. Le plus grand document a 165 nœuds de contenu et 114 nœuds internes. Le schéma cible définit 13 étiquettes possibles pour les nœuds de contenu et 11 pour les nœuds internes.

Le second corpus est basé sur les données d'IMDb (`http://imdb.com`). 710 descriptions de films ont été téléchargées et converties manuellement en XML suivant un schéma donné. Chaque description comporte, en moyenne, 35 nœuds de contenu et 35 nœuds internes ; la plus grande en a, respectivement, 212 et 211. C'est un corpus de type « base de données » : comme dans les bases de données relationnelles, les descriptions de film ont une structure $attribut–valeur$ et seuls quelques nœuds ont des données textuelles (généralement les commentaires des utilisateurs et les résumés). En conséquence, la structure des documents est plus régulières que dans le premier corpus.

Chaque corpus a été séparé aléatoirement en un ensemble d'apprentissage et un ensemble de test. Tous deux comportent les documents d'entrée en HTML et les documents cible correspondant en XML, toutefois, pour le test seuls les documents d'entrée sont utilisés. Ces deux ensembles ont la même taille. Notre modèle a alors été utilisé pour retrouver la structure cible XML des documents du corpus de test. Différentes valeurs de N (le nombre de solutions candidates générées par la première étape) ont été testées. Notre modèle de base ($N = 1$) correspond au cas où seule l'étape de génération entre en jeu : la reconstruction a alors lieu sans tenir compte de la structure du document d'entrée et est proche du modèle proposé par (Chidlovskii et Fuselier, 2005). Nous proposons d'évaluer la qualité de la restructuration en mesurant la similitude entre le document reconstruit et le document convertit la main. Comme mesure de similarité, nous avons utilisé le pourcentage de constituants correctement reconstruit. Un constituant correspond à une paire $(étiquette, couverture)$ et permet de décrire à la fois l'étiquette d'un nœud et sa position dans l'arbre reconstruit. Nous avons mesuré séparément les résultats de la reconstruction sur les feuilles de l'arbre et sur les nœuds internes

de celui-ci afin de pouvoir évaluer la capacité de notre modèle à identifier des éléments et à identifier des relations entre ces éléments.

La Table 2 rassemble les résultats de nos expériences. Dans tous les cas, l'utilisation de l'information sur le document d'entrée lors de l'étape de ré-ordonnancement améliore les résultats du modèle de base, pour une complexité globale sensiblement identique. Cette amélioration est significative lorsque l'on ne considère qu'un petit nombre de solutions candidates. Cependant, le fait d'augmenter le nombre de solutions candidates au delà d'un certain seuil lors de l'étape de ré-ordonnancement diminue les performances. Cette baisse est certainement due aux limites de l'algorithme d'apprentissage notamment par rapport au petit nombre de données fournies en apprentissage. Une analyse plus détaillée des résultats montre que les performances de l'approche sont très bonnes sur les parties les plus régulières des documents, mais que la qualité de la reconstruction chute dans les parties récursives des documents. Dans tous les cas, les résultats de l'étape de ré-ordonnancement dépendent de la qualité des solutions candidates.

	LinuxFr			Movie		
	Feuilles	Nœuds internes	Arbre	Feuilles	Nœuds internes	Arbre
$N = 1$	39,6%	24,2%	40,5%	54,3%	49,1%	52,2%
$N = 10$	56,6%	35,6%	57,9%	64,7%	59,7%	63,2%
$N = 100$	41,3%	26,1%	42,9%	59,6%	53,9%	56,9%

TAB. 2 – *Résultat de reconstruction sur les deux corpus*

5 État de l'art

Des problèmes semblables (*schema matching*, intégration de données, ...) sont traités depuis plusieurs années par la communauté base de données et sont apparus, plus récemment, pour la recherche documentaire, la conversion(Chidlovskii et Fuselier, 2005) et l'intégration de documents (Chung et al., 2002), et l'alignement d'ontologies. Plusieurs techniques d'apprentissage ont été employées : classifieur multi-classes, spectre de graphe, ... Plusieurs travaux récents abordent la problématique de la transformation de documents, notamment avec des grammaires formelles (Chidlovskii et Fuselier, 2005; Wisniewski et al., 2006).

Une comparaison des différentes approches proposées en base de données est faite dans (Doan et Halevy., 2005). (Doan et al., 2003) présente une des approches les plus abouties pour travailler sur différents types de données (SQL, XML, ontologies). La tâche y est présentée comme un problème de classification supervisée multi-étiquettes. Toutefois les corpus considérés sont très différents des corpus auxquels nous nous intéressons : l'évaluation des méthodes développées a, généralement, été faite sur des corpus de petite taille, ayant une structure très stricte présentant peu de récursion et ne contenant que très rarement des données textuelles.

6 Conclusion

Nous avons proposé un cadre général pour la transformation de document semi structuré vers un schéma arbitraire. Ce cadre nous permet de considérer des caractéristiques du document d'entrée et de la structure de sortie, améliorant ainsi plusieurs approches existantes.

Plusieurs expériences prospectives ont été réalisées. Les résultats sont encourageants, même si de nouvelles caractéristiques doivent être définies pour améliorer les performances.

Plusieurs améliorations sont envisageables. Nous pensons notamment réduire la complexité de l'étape de génération en utilisant des méthodes d'inférence approchée. Une autre piste consiste à étudier comment les informations apprises sur un corpus peuvent être utilisées pour faciliter l'apprentissage de la transformation d'un corpus similaire.

Références

Chidlovskii, B. et J. Fuselier (2005). A Probabilistic Learning Method for XML Annotation of Documents. In *IJCAI*.

Chung, C. Y., M. Gertz, et N. Sundaresan (2002). Reverse engineering for web data : From visual to semantic structures. In *ICDE*.

Collins, M. et N. Duffy (2001). Convolution kernels for natural language. In *NIPS*.

Collins, M. et N. Duffy (2002). New ranking algorithms for parsing and tagging : Kernels over discrete structures, and the voted perceptron. In *ACL*.

Collins, M. et T. Koo (2005). Discriminative reranking for natural language parsing. *Computational Linguistics*.

Doan, A., P. Domingos, et A. Halevy (2003). Learning to match the schemas of data sources : A multistrategy approach. *Mach. Learn. 50*(3), 279–301.

Doan, A. et A. Halevy. (2005). Semantic integration research in the database community : A brief survey. *AI Magazine, Special Issue on Semantic Integration*.

Jimnez, J. et A. Marzal (2000). Computation of the n best parse trees for weighted and stochastic context-free grammars. In *SSPR/SPR*.

Jurafsky, D. et J. H. Martin (2000). *Speech and Language Processing : An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*.

Tsochantaridis, I., T. Hofmann, T. Joachims, et Y. Altun (2004). Support vector machine learning for interdependent and structured output spaces. In *ICML*.

Wisniewski, G., L. Denoyer, F. Maes, et P. Gallinari (2006). Modèle probabiliste pour l'extraction de structures dans les documents semi-structurés. In *CORIA'06*.

Summary

Many documents on the web are formated in a weakly structured format. The exploitation of these documents, by systems like INEX, is made difficult by the heterogeneity of their structure and by their weak semantic. We consider here the conversion of such documents into a predefined semi-structured format which will be be more amenable to automatic processing of the document content. We develop a machine learning approach to this conversion problem, based on a parsing-like algorithm and a kernel perceptron.