Recherche des objets complexes dans le Web structuré

Nora Derouiche

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2012-ENST-011

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Informatique et Réseaux »

présentée et soutenue publiquement par

Nora DEROUICHE

le 20 Mars 2012

Titre

Recherche des Objets Complexes dans le Web Structuré

Directeur de thèse : **Talel ABDESSALEM**
Co-encadrement de la thèse : **Bogdan CAUTIS**

Jury

M. Mohand-Said HACID, Professeur, Université Claude Bernard Lyon 1	Rapporteur
Mme Amélie MARIAN, Maitre de Conférence, Université de Rutgers, New Jersey	Rapporteur
M. Bernd AMANN, Professeur, UPMC (Université de Paris 6)	Examinateur
M. Cédric DU MOUZA, Maitre de Conférence, CNAM (Université de Paris 1)	Examinateur
M. Dan VODISLAV, Professeur, Université de Cergy-Pontoise	Examinateur

TELECOM ParisTech
école de l'Institut Télécom - membre de ParisTech

T H È S E

Les seules choses qui sont impossibles à finir sont celles que l'on ne commence pas.

Lynn Johnston

Remerciements

Je tiens tout particulièrement à remercier tous les membres du jury de m'avoir fait l'honneur de participer à ma soutenance. Madame Amélie MARIAN et Monsieur Mohand-Said HACID pour avoir accepté d'être les rapporteurs de ce travail. Monsieur Bernd AMANN, Monsieur Cédric DU MOUZA et Monsieur Dan VODISLAV pour avoir examiné ce travail. Je vous remercie pour l'intérêt que vous avez porté à ce travail et pour vos remarques pertinentes.

Je tiens à exprimer mes profondes gratitudes et reconnaissances à toutes les personnes qui ont permis la réalisation et l'aboutissement de cette thèse :

J'adresse mes plus sincères remerciements à mes directeurs de thèse Monsieur Talel ABDESSALEM et Monsieur Bogdan CAUTIS pour m'avoir donné l'opportunité de réaliser cette thèse au sein de Télécom ParisTech. Un grand merci pour la confiance qu'ils m'ont accordé, pour leur aide, leur soutien, leur disponibilté ainsi que leurs conseils et la qualité de leur encadrement, qu'ils m'ont apporté tout au long de cette thèse. Leur apport scientifique, leur expertise et leur regard critique m'ont été de précieux éléments à l'aboutissement de mes travaux de recherche décrits dans ce manuscrit.

Je tiens également à remercier chaleureusement tous les membres de l'équipe DBWeb et mes collègues de Télécom ParisTech pour l'appui qu'ils m'ont témoigné durant ces années de travail, particulièrement Fabian SUCHANEK pour son écoute et ses précieux conseils.

Enfin j'adresse mes remerciements singuliers à toutes les personnes qui ont eu une place majeure dans ma vie personnelle, et dans le cadre des travaux de cette thèse. Je pense particulièrement à Farid KHIMECHE pour son soutien et sa patience, ma famille, mes amis pour vos encouragements durant toute cette période.

À la mémoire de mon très cher père CHERIF.

Résumé

Nous assistons aujourd'hui à un développement continu et rapide du Web Structuré, dans lequel les documents (les pages Web) ne sont plus composés que du texte non structuré mais sont centrés sur les données, présentant des contenus structurés et des objets complexes. Ces pages Web sont générées le plus souvent de façon dynamique à partir d'une base de données accessible via des formulaires (Web caché), et sont organisées selon une structure régulière et prédéfinie. Les plates-formes de recherche actuelles ne permettent d'obtenir que des pages en utilisant des méthodes traditionnelles de recherche par des mots-clés, qui sont inadaptées pour interroger le *Web structuré*. En effet, la recherche par mots-clés est sémantiquement pauvre et ignore les liens structurels existant entre les différents contenus des objets complexes (ex. dans une page Web d'un site commercial, constituée d'une liste de livres, les entités élémentaires "titre" et "auteur" composant chaque "livre" sont présentées selon une disposition qui illustre leurs relations. De nouveaux moyens de recherche sur le Web sont donc nécessaires, pour permettre à l'utilisateur de cibler des données complexes avec une sémantique précise.

L'objectif de cette thèse est de fournir des algorithmes efficaces pour l'extraction et la recherche des objets structurées (un livre, un concert de musique, etc.) de façon automatique, à l'aide de méthodes adaptées allant au-delà de la recherche par mots-clés. Nous avons proposé une approche d'interrogation du Web en deux étapes, qui permet à l'utilisateur de décrire le schéma des objets ciblés, de façon souple et précise. Les deux problématiques principales adressées sont : (1) la sélection de sources Web structurées les plus pertinentes pour un schéma fourni par l'utilisateur (c-à-d, contenant les objets, instances de ce schéma), et (2) la construction de wrappers (extracteurs) pour l'extraction des objets complexes ciblés à partir des sources sélectionnées, en exploitant la régularité des structures des pages et la sémantique des données.

Notre approche est générique, dans le sens où elle nŠest pas spécifique à des sources ou des objets d'un domaine particulier. Elle a été implantée (système `ObjectRunner`) et testée sur des sources Web appartenant à des domaines variés. Les résultats obtenus montrent, en particulier, une pertinente élevée au niveau de la sélection de sources et un gain significatif au niveau de la qualité de l'extraction par rapport aux approches existantes.

Abstract

We are witnessing in recent years a steady growth of the so-called structured Web, in which documents (Web pages) are no longer quasi-textual, but are data-centric, presenting structured content, complex objects. Such schematized pages are often generated dynamically by means of formatting templates over a database, possibly using user input via forms (hidden Web). The current Web search platforms allow only to retrieve Web pages by traditional keyword search methods, which are not adapted to query the *structured Web*. Indeed, keyword search is semantically poor and ignores the existing structural links between various components of complex objects (e.g., in a commercial Web site page, providing book lists, the atomic entities "title" and "author" forming each "book" are displayed in a way that illustrates their relationship. New ways of searching the Web are thus required, in order to enable users to target complex data, with a clear semantics.

The main aim of this thesis is to provide effective algorithms for extracting and retrieving structured objects (e.g., a book, a music concert, etc.) automatically, using adapted methods rather going beyond the keyword search ones. We propose a two-phase querying approach of the Web, which allows users to first describe the schema of the targeted objects, in a flexible, lightweight and precise manner. The two main problems we address are : (1) the selection of the most relevant structured Web sources with respect to the schema provided by the user (i.e., containing objects, instances of this schema), and (2) the construction of wrappers for extracting the targeted complex objects from the selected sources, leveraging both the regularity of the pages and the semantics of the data.

Our approach is generic, in the sense that it can be applied to any domain and schema for complex objects. It has been implemented in the `ObjectRunner` system, and tested extensively. The experimental results show high source-selection relevance and significant improvements over existing techniques in terms of extraction precision.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Nous assistons aujourd'hui à un développement continu et rapide du *Web Structuré*, dans lequel les documents ne sont plus composés que du texte non structuré mais sont centrés sur les données *(data-centric)*, présentant des contenus structurés et des *objets complexes*. Les plates-formes de recherche d'information (IR) actuelles s'inspirent des procédures et des techniques utilisées dans les systèmes de recherche documentaire. Le passage à des contenus structurés, avec des schémas prédéfinis, nécessite des techniques d'interrogation plus précises et plus riches, et soulève de nouveaux défis auxquels nous essayons de fournir des réponses. En effet, la recherche par *mots-clés* n'est pas adaptée pour interroger le Web structuré. De nouveaux moyens de recherche sur le Web sont donc nécessaires, pour permettre à l'utilisateur de cibler des données complexes avec une sémantique précise.

Nous étudions dans cette thèse les défis théoriques et pratiques qui sont soulevés dans la recherche d'objets complexes dans le Web. Nous proposons `ObjectRunner`, un système pour l'extraction et l'interrogation de données du Web structuré, qui exploite la redondance du Web et la régularité des structures des pages pour mieux déterminer les données à extraire. Ce système fournit un résultat le plus complet possible à des requêtes plus riches que celles constituées de simples mots-clés. Nous proposons une approche d'interrogation en deux étapes, qui permet à l'utilisateur de décrire de façon souple et précise le schéma des objets recherchés. Ensuite, pour chaque source Web (ensemble de pages `HTML`), le schéma cible et la structure des pages sont analysés pour : (1) sélectionner les sources répondant à la requête de l'utilisateur, (2) retrouver les données pouvant répondre à sa requête, et (3) les extraire. La solution proposée par notre système est générique, dans le sens où elle n'est pas spécifique à un domaine d'application ou un type d'objet en particulier. Les résultats expérimentaux que nous avons obtenus montrent une amélioration significative de la qualité et la précision des données extraites, par rapport aux approches existantes dans la littérature.

1.1 Contexte et Problématique

La popularité, le développement du Web et sa large utilisation dans différents domaines font que de nouveaux besoins apparaissent, tels que la recherche d'entités complexes *(ou objets complexes)*. Dans ce contexte, une requête utilisateur peut cibler des entités complexes telles qu'un événement sportif ou culturel, un produit, une œuvre d'art, une vente d'appartement, etc. L'utilisateur s'attend à avoir des résultats bien précis, qui répondent à ses attentes en terme de structure et de sémantique des objets recherchés. Cependant, les moteurs de recherche actuels ne permettent d'obtenir que des pages en utilisant des méthodes traditionnelles de recherche par des *mots-clés*. Ces méthodes sont sémantiquement pauvres et limitées, elles ne permettent pas une recherche précise des objets et ne tiennent pas compte de la sémantique associée à la requête de l'utilisateur. Prenons les exemples de requêtes suivantes qui concernent la recherche d'entités complexes :

- **Scénario 1.** L'utilisateur recherche un événement culturel : *Concert de Coldplay à Paris le 4 février 2012*. La requête fait référence à un objet précis constitué de trois entités élémentaires : une `date` (4 février 2012), un `lieu` (Paris) et un `nom de l'artiste` (Coldplay). Une recherche par mots-clés produirait une liste de pages contenant tout (ou une partie) des mots-clés : concert, Coldplay, Paris, 4 février 2012.
- **Scénario 2.** L'utilisateur recherche le livre *The Alchemist* de l'auteur *Paolo Coelho*. Cette requête fait référence à un objet constitué de deux entités élémentaires : un `titre` du livre et un `nom de l'auteur`. Une recherche par mots-clés produirait une liste de pages contenant les entités : The Alchemist et Paolo Coelho.

De façon générale, l'objectif de ces requêtes n'est plus de rechercher des pages par des mots-clés mais de rechercher des objets structurés – un concert, un livre – de façon automatique et à l'aide de méthodes adaptées.

Vu la forte redondance du Web, les mêmes objets peuvent se trouver dans différentes pages et sous différents formats : dans des pages Web textuelles (non structurées), mais aussi dans des pages Web structurées. Considérons l'exemple de recherche de l'objet `livre` du `Scénario 2`. La Figure 1.1 représente un segment d'une telle page textuelle extraite du site biographique `"biography.com"`, contenant des entités répondant à la requête : *The Alchemist* et *Paolo Coelho*. Reconnaitre ces deux entités dans le texte et les extraire séparément est une tâche relativement facile à réaliser. Cependant, il est difficile de déterminer si elles sont liées et si le texte décrit bien le livre *The Alchemist* de

Paolo Coelho, du fait qu'elles se retrouvent dans deux phrases différentes. Cela nécessite généralement des techniques de traitement automatique du language naturel (TALN) adaptées pour détecter les relations existantes entre les entités.

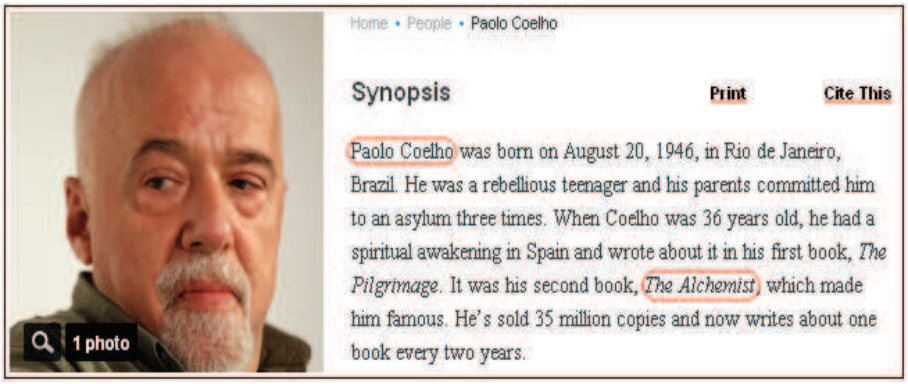


FIGURE 1.1 – Segment d'une page du site `biography.com`

Le même objet `livre(titre` *`The Alchemist`*`, auteur` *`Paolo Coelho`*`)` peut se trouver également dans des pages Web structurées (voir la Figure 1.2). Le Web structuré est de plus en plus répondu et le nombre de sources structurées augmente très rapidement. Une page du Web structuré présente des structures plus ou moins régulières. Ces pages sont typiquement composées de blocs *(segments)*, organisés hiérarchiquement et présentés selon une disposition qui indique des liens possibles entre leurs contenus. Les blocs sémantiquement liés sont disposés de manière à illustrer leurs relations. De telles propriétés peuvent fournir des informations essentielles pour analyser les pages Web et extraire des données complexes. Dans la page de la Figure 1.2, les livres sont décrits de façon structuré. Chaque livre est décrit dans un segment (bloc) séparé et la page est constituée d'une liste de segments. D'un bloc à l'autre, les entités élémentaires (titre et auteur) se retrouvent à la même position dans chaque segment, pour tous les livres disponibles sur le site.

L'exploitation de telles sources reste difficile pour répondre à des *requêtes complexes*, sans connaissances sur le schéma des données. De plus, d'une source à l'autre les mêmes objets peuvent être structurés différemment. Par conséquent, une étape préliminaire d'*extraction d'information* est nécessaire pour transformer ces sources en des collections de données (objets) structurées, avant de passer à l'étape d'interrogation. À cet effet, nous avons besoin de techniques spécifiques pour capter les structures des données contenues dans chaque source et les extraire.

La tâche d'extraction de données à partir d'une source Web structurée est réalisée par ce qu'on appelle dans la littérature un wrapper *(extracteur)*. Chaque wrapper applique un ensemble de règles d'extraction qui s'appuient sur la structure des pages HTML pour

FIGURE 1.2 – Segment d'une page du site `amazon.com`

repérer les données et construire un modèle de pages commun *(template)*. Ce modèle va permettre d'extraire les informations contenues dans ces pages.

C'est dans ce contexte (extraction au préalable de l'interrogation) que s'inscrit cette thèse. L'objectif était d'étudier les problèmes fondamentaux soulevés par l'extraction d'objets à partir du Web et de proposer des solutions pratiques, qui peuvent passer à l'échelle, et qui permettent d'améliorer la qualité des résultats de recherche que l'utilisateur peut obtenir.

1.2 Contributions de la thèse

Nous proposons dans cette thèse, avec le système `ObjecRunner`, une approche qui permet aux utilisateurs d'interroger les données structurées du Web en deux étapes, comme suit :

- **Première étape d'interrogation.** L'utilisateur définit une structure des données – appelée dans la suite *Structured Object Description* (SOD) – pour décrire le type d'information structurée qu'il cherche et qui doit être identifiée et extraite à partir du Web structuré. Par exemple, pour des objets `concerts` *(Scénario 1)*,

l'utilisateur fournit un SOD qui peut être décrit par un tuple regroupant les entités atomiques `artiste`, `date` et `adresse`. Le système que nous proposons dans cette thèse s'appuie sur le SOD pour sélectionner les sources pertinentes, inférer leurs templates et extraire leurs contenus. Pour ce scénario, `ObjectRunner` extrait la liste des concerts de sources structurées qui publient des événements de musique comme *zvents.com*, *upcoming.yahoo.com*, etc.

– **Deuxième étape d'interrogation.** Une fois la tâche d'extraction réalisée, l'utilisateur peut interroger les données extraites pour trouver des objets recherchés. Par exemple, recherche de l'objet *Concert de Coldplay à Paris le 4 février 2012*.

Les deux étapes précédentes décrivent l'interaction de l'utilisateur avec notre système. Cependant, une analyse structurelle et sémantique de sources Web est réalisée selon le processus suivant :

– **La découverte de sources structurées :** consiste à déterminer (de façon non supervisée) si une source contient des données structurées ou non. Autrement dit, si la source publient des objets structurés.
– **Pré-indexation de sources :** consiste à déterminer si une source est susceptible de contenir des instances de SODs, de manière générique. Ceci nécessite une technique d'indexation adéquate qui permettra de trouver les sources structurées pertinentes pour tout SOD spécifié ensuite par l'utilisateur.
– **La sélection de sources pertinentes :** sélectionner les sources les plus pertinentes, celles répondant le plus à la description recherchée (SOD) par l'utilisateur *(Première étape d'interrogation)*.
– **L'extraction d'objets structurés :** détecter dans les pages le schéma (SOD) recherché et extraire automatiquement les informations (les objets) contenues dans ces pages.
– **L'intégration / De-duplication :** détecter les occurrences multiples d'un même objet, provenant de plusieurs sources Web et correspondant au même objet du monde réel. Les occurrences identifiées sont fusionnées pour former des données plus complètes.
– **L'interrogation des données structurées :** interroger les données extraites de la même manière que pour une base de données relationnelle *(Deuxième étape d'interrogation)*.

Dans cette thèse, nous nous sommes intéressés tout particulièrement aux deux problématiques principales qui sont : (1) la selection de sources structurées pertinentes, pour un SOD donné en se basant sur l'indexation, et (2) la reconnaissance et l'extraction automatique d'objets à partir de ces sources. Notre objectif principal étant de mettre

en pratique et de développer une méthode *non supervisée* pour la recherche d'objets complexes à partir du Web structuré.

Une architecture globale du système `ObjectRunner` est donnée dans la Figure 1.3. Nous supposons l'existence d'un ensemble de sources Web (pas forcément structurées) collectées à partir du Web *(crawled)* et indexées pour la sélection et l'extraction. Le système utilise la structure des objets fournie par l'utilisateur pour : (1) identifier les sources structurées qui sont pertinentes pour l'extraction (sources structurées, du même domaine considéré), (2) extraire *(wrapping)* de ces sources les données pouvant correspondre aux objets recherchés. Ensuite, le résultat d'extraction est stocké dans une base de données structurée, que l'utilisateur pourra interroger.

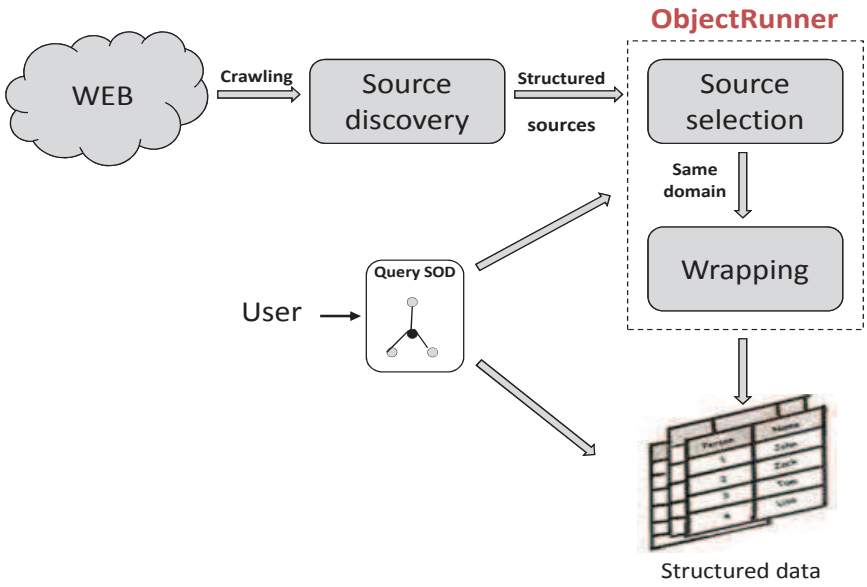


FIGURE 1.3 – Architecture globale

Plus précisément, les contributions principales de ce travail sont les suivantes :

1. **Extraction d'information structurée.** Nous proposons une méthode d'extraction ciblée qui exploite au préalable les connaissances sémantiques sur les objets recherchés (SOD), afin de guider le processus d'extraction. À partir d'un corpus de sources Web, où chaque source est composée d'un ensemble de pages partageant un schema commun et une structure commune (implicite), `ObjectRunner` construit un template d'extraction (wrapper) unique à chaque source, en utilisant : (i) la description intentionnelle des données ciblées (SODs) fournie par l'utilisateur, (ii) la structure des pages de la source, et (iii) le contenu sémantique de ces pages, en exploitant notamment des ontologies basées sur Wikipedia ou des dictionnaires

construits automatiquement à partir de corpus Web. Ensuite, il utilise le template inféré pour la source pour récupérer les objets, c-à-d, des données structurées correspondant aux instances possibles du SOD spécifié.

2. **Sélection de sources structurées.** En complément à l'extraction, nous proposons une technique de selection de sources. Les techniques d'extraction existantes supposent toutes que les sources Web sont explicitement fournies en entrée. Cependant, la découverte et la sélection de ces sources n'est pas une tâche facile, c'est même un défi majeur pour un accès plus facile aux données structurées du Web, si ce travail de découverte et d'indexation est réalisé de façon non supervisé. Notre approche – la première à notre connaissance – exploite (i) la structure hiérarchique et les caractéristiques visuelles des pages pour identifier les blocs sémantiquement riches en données, et (ii) des domaines de connaissances (ontologies) pour la reconnaissance des entités composant le SOD spécifié. Cette approche est construite en se basant sur des techniques de recherche d'information basées sur des ontologies, d'indexation de données structurées, et de recherche top-k sur des données semi-structurées (structures d'arbres).

1.3 Organisation du manuscrit

La suite de ce manuscrit est organisée comme suit.

- Le chapitre 2 définit plus précisément le contexte scientifique de cette thèse. Nous présentons l'état de l'art sur les travaux connexes aux problématiques de recherche d'information structurée à partir du Web, en détaillant les avantages et les limites de chaque approche.
- Le chapitre 3 introduit les concepts utilisés tout au long de ce manuscrit. Nous définissons les deux problématiques traitées de manière détaillée.
- Le chapitre 4 détaille l'approche proposée pour l'extraction d'objets à partir du Web structuré. Nous présentons une architecture générale de l'approche proposée. Ensuite, nous décrivons la procédure d'extraction d'information ciblée pour les différents modules qui la composent, et les techniques d'implantation utilisées pour chaque module. Nous illustrons également son fonctionnement à travers plusieurs exemples d'application et nous présentons les résultats expérimentaux obtenus.
- Le chapitre 5 présente la solution proposée pour la sélection de sources structurées à partir du Web. Nous présentons l'architecture générale de l'approche proposée. Ensuite, nous détaillons les techniques utilisées pour l'annotation, l'indexation et le ranking de sources pertinentes pour la requête de l'utilisateur. Enfin, nous présentons les résultats expérimentaux.

– Le chapitre 6 conclut ce rapport en rappelant les problématiques traitées, les solutions proposées et les résultats obtenus. Il présente également quelques perspectives de recherche pour adresser la totalité des problèmes liés à la recherche des informations complexes sur le Web structuré avec `ObjectRunner`.

Chapitre 2

Contexte et travaux connexes

Nous avons identifié trois problèmes majeurs liés à la recherche d'information à partir du Web structuré : (1) la découverte automatique de sources structurées à partir du Web, (2) la sélection des sources pertinentes pour la tâche d'extraction, et (3) l'inférence et la génération du template pour l'extraction d'information. Notre travail touche aux deux dernières problématiques : la sélection de sources et l'extraction d'information. Nous présentons dans ce chapitre les travaux existants pour les trois domaines. Tout d'abord, nous résumons les travaux de recherche sur l'extraction d'information, en détaillant les techniques d'extraction les plus étroitement liées (Section 2.1), et nous commentons leur connexion à notre méthode d'extraction basée sur l'extraction ciblée du Chapitre 4. Ensuite, nous présentons les travaux existants sur la découverte et la sélection de sources Web (Section 2.2).

2.1 Extraction d'information

Afin que les applications puissent exploiter les diverses et nombreuses informations disponibles sur le Web, ces informations doivent être extraites et transformées aux formats de représentation appropriés. Cette tâche est appelée *Extraction d'Information*. Elle consiste à repérer et à déduire une information structurée et détaillée à partir de sources Web riches, hétérogènes, non structurées, ou faiblement structurées. Dans le Web on peut trouver les informations à extraire dans différentes sources et sous différents formats : le Web non structuré, le Web structuré et le Web caché.

– **Pages Web non structurées :** aussi connu sous le nom du *Web textuel*, les informations peuvent être décrites en langage naturel : le texte dans les pages, les blogs, les dépêches, etc. Afin de les extraire, plusieurs tâches doivent être effectuées selon [CM04] : (1) la *segmentation* et le découpage du texte en segments pour la reconnaissance des entités nommées à extraire, (2) la *catégorisation (classification)*

de ces entités et l'identification des noms, dates, organisations, etc., (3) l'*association* et l'identification des relations entres les différentes entités et les intégrer dans un seul enregistrement *(record)*, et (4) la *classification (clustering)* des enregistrements qui référencent le même enregistrement dans le monde réel, i.e., ceux qui ont la même sémantique. Finalement, ces informations sont représentées dans une forme structurée.

- **Pages Web structurées :** les informations peuvent être présentées sous forme de listes HTML, des tableaux ou des données XML, etc. Contrairement au Web textuel, dans le Web structuré les pages suivent la même mise en page (structure). Cette structure peut être exploitée pour l'identification des informations structurées (objets). Cependant, elle est unique à chaque site et pratiquement il n'existe pas de description (grammaire) générale qui peut décrire tous les schémas de pages possibles. En conséquence, chaque structure peut nécessiter un extracteur spécifique, ce qui rend la programmation manuelle des extracteurs très coûteuse.
- **Web caché :** aussi connu sous le non du *Web caché*, les informations peuvent aussi être décrites dans des pages non indexées par les moteurs de recherche (dans des bases de données accessibles par le biais de formulaires Web). Il y a deux problématiques principales à traiter pour l'extraction à partir du Web caché : (1) comment parvenir à comprendre le formulaire et trouver la requête qui permet de remplir les champs de ce formulaire, (2) une fois le formulaire rempli, on se retrouve confronté aux mêmes problématiques liées à l'extraction d'information à partir du Web structuré.

2.1.1 Entités, relations, type d'extraction

On rappelle que l'extraction d'information est le processus de récupération des informations structurées à partir des textes, ces informations pouvant être des entités simples ou relations binaires, n-aires ou imbriquées *(nested structure)*.

- **Entités.** Elles représentent une seule unité d'information. Elle sont généralement des phrases nominales comprenant un ou plusieurs mots. Les plus populaires sont les *entités nommées* comme les noms de personnes, les noms de pays, etc. L'identification des entités nommées inclut traditionnellement trois types d'expressions : les noms propres (personne, organisation, lieu), les expressions temporelles (date, période, durée, etc.) et les expressions numériques (poids, nombre, température, etc.).
- **Relations binaires.** Elles sont définies par l'intermédiaire de deux entités liées d'une manière prédéfinie. Par exemple, la relation *"is written by"* entre un auteur et un titre de livre, *"is price of"* entre un prix et un produit, etc.

– **Relations n-aires ou imbriquées.** Elles relient plusieurs entités, ces relations peuvent être *plates* ou *imbriquées*. Par exemple, un object `concert` peut avoir des relations entre le nom de l'artist, la date et l'adresse. L'adresse peut être une relation imbriquée contenant d'autres relations entre le nom de la ville, le nom de la salle de spectacle, la rue, etc.

2.1.2 Annotation sémantique

Une annotation désigne une information "sémantique" associée à un ou plusieurs mots existants dans le texte. Les documents à annoter sont découpés en mots *(tokenized)*[1], et la séquence de mots qui correspond au maximum à une entité dans un domaine de connaissance est identifiée. Ce domaine peut être spécifié à l'aide de listes d'instances ou d'expressions régulières (ensemble de patterns). Plus précisément, la tâche d'annotation est effectuée par l'identification des candidats dans le texte, en exploitant un domaine de connaissance. Par la suite, une étiquette *(label)* est choisie pour chaque candidat. L'annotation est utilisée pour plusieurs applications, dont l'extraction d'information.

Habituellement, les annotations sont réalisées en utilisant une ontologie*, qui décrit formellement un domaine de connaissance. Elle consiste à relier une instance existante dans le texte au concept approprié.*

Ontologies, Taxonomies

De nos jours, plusieurs représentations normalisées de connaissances du monde réel sont formalisées sous forme d'un schéma hiérarchisé appelé *ontologie*. Autrement dit, une ontologie désigne toute collection structurée de connaissances du monde réel attachée à des règles sémantiques. Souvent, ces bases de connaissances ont été construites automatiquement par l'extraction des entités-relations à partir de sources Web. Parmi ces bases de données, celles construite à partir de Wikipédia (comme, KnowItAll [CDSE05], *DBPedia* [ABK+07] et *YAGO* [SKW07]), des ontologies commerciales (comme, *FreeBase.com* et *OpenCyc.org*), ou des ontologies plus spécialisées contenant des données spécifiques à un domaine (comme, *MusicBrainz.org* pour les données du domaine de musique, *Geonames.org* pour les données du domaine géographique).

Beaucoup de ces ontologies contiennent des données complémentaires. Cependant, puisqu'elles utilisent généralement différents identifiants pour les entités, cette information ne peut pas être connectée facilement. Partant de cette direction, un graphe globale de données regroupant ces ontologies connu sous le nom de *Linked Open Data*[2] a

1. Le *tokeniser* permet de découper le texte en unités élémentaires ou de base appelés *tokens*.
2. http ://linkeddata.org/

été créé, qui permet de connecter et de découvrir de nouvelles sources de données [BHIBL08, Biz10, BHBLH11]. Les entités dans ce graphe sont connectées par des liens, en utilisant les mêmes `URIs` pour les entités similaires (ce graphe contient environ 26 billion de tuples et 400 millions de liens entre les sources de données, en Septembre 2010).

Annotation guidée par une ontologie

Plusieurs travaux exploitent un ensemble déjà prédéfini de concepts, de propriétés et de relations (i.e., des ontologies) pour la reconnaissance et la découverte des entités et leurs relations dans le texte. Ceci rend l'annotation plus précise d'un point de vue de la structure et de la sémantique. Cette idée de guider le processus d'annotation en utilisant des ontologies a été utilisée dans diverses applications : (i) rechercher des tables HTML dans des pages Web et trouver la sémantique de chaque colonne [LSC10], (ii) pour supporter la recherche sémantique et améliorer les méthodes de recherches traditionnelles par mots-clés [FCL+11], (iii) trouver des candidats instances-relations à partir du Web [MC08b], ou (iv) assister la tâche d'extraction d'information en construisant une ontologie de domaine [SWL09].

2.1.3 Extraction d'information à partir du Web textuel

De nombreux travaux ont porté sur l'identification des *entités nommées* dans le texte, notamment dans les années 90. Ces travaux ont été conduits par une série de sept campagnes d'évaluations dans le traitement de la langue naturelle (TLN), organisés par le DARPA (MUCs, Message Understanding Conferences) [MUC-6, 1995] [MUC-7, 1998].

La plupart des travaux sur l'extraction d'information à partir du Web non structuré (textuel) se focalisent sur l'extraction des *relations* sémantiques entres les *entités* pour un ensemble de motifs *(patterns)* qui les intéressent. Ces patterns peuvent être définis manuellement pour chaque tâche, prédéfinis au préalable par le système, ou découverts d'une manière automatique.

Un des défis majeurs dans l'extraction d'information est l'implication de l'utilisateur dans la création d'un nouvel échantillon (ensemble de patterns), ou le temps de programmation consacré pour chaque nouvelle tâche (nouvelle entité). Ce défi a été abordé de différentes manières. Une des approches consiste à construire une interface utilisateur efficace et intuitive en utilisant des procédures d'apprentissage. Un tel système peut être rapidement adapté pour chaque nouvelle tâche.

Proteus [YG98] est un exemple d'un tel système, il permet à l'utilisateur (généralement un expert du domaine) de créer, généraliser et tester les patterns d'extraction en se basant sur l'analyse d'un ensemble de documents textuels. L'utilisateur interagit avec le

système, fournissant des exemples de phrases ou des segments de texte. Le système utilise ces exemples pour la génération de nouveaux patterns d'extraction ou pour l'expansion de patterns déjà existants, avec la validation de l'utilisateur tout au long du processus.

Un autre système GATE [CMBT02], fournit des composants génériques qui permettent facilement de concevoir, de développer et d'évaluer les patterns d'extraction pour une tâche d'extraction donnée. GATE est un système open source qui permet d'étendre les composants de chaque étape du processus d'extraction. Il est très flexible, néanmoins, c'est au développeur d'adapter les composants pour chaque nouvelles tâche d'extraction.

Brin a proposé DIPRE [Bri99], un système pour l'extraction de relations à partir d'une collection de pages Web. DIPRE se focalise principalement sur l'exploitation de la *dualité entre les patterns et les relations*. Plus précisément, il se base sur l'idée suivante : avoir suffisamment de patterns pour une relation permet de trouver automatiquement d'autres patterns, et repérer les patterns fiables permet de trouver automatiquement plus de relations.

DIPRE commence par un petit échantillon de relations prédéfinies afin d'identifier de nouveaux patterns, ce processus est connu par le nom du *bootstrapping*. Par exemple, pour extraire la relation `books(author,title)`, un ensemble d'instances de la paire `(author,title)` est fourni en entrée. Il trouve toutes les occurrences de ces livres dans le Web, puis à partir de ces occurrences, de nouveaux patterns sont repérés pour cette relation. Ces nouveaux patterns sont utilisés pour trouver de nouveaux livres. Par la suite, ces livres sont utilisés à leur tour pour trouver de nouvelles occurrences et générer de nouveaux patterns (comme illustré dans le graphe ci-après).

$$\begin{array}{ccc} \{\text{instances de relations}\} & \longrightarrow & \{\text{occurrences dans le Web}\} \\ \uparrow & & \downarrow \\ \{\text{nouvelles instances}\} & \longleftarrow & \{\text{expressions régulières (patterns)}\} \end{array}$$

Néanmoins, l'inconvénient majeur de cette approche est que les patterns générés ne sont pas toujours corrects. Une série de travaux ont tenté de développer cette idée et de l'améliorer.

Snowball [AG00] a introduit de nouvelles stratégies pour : (i) générer les patterns et extraire des tuples de relations, et (ii) évaluer la qualité des patterns et des tuples extraits. La principale amélioration est l'introduction de l'`annotation` des entités nommées dans le texte, au lieu de chercher des instances (strings), on cherche plutôt des entités. Par exemple, chercher le pattern `<Title>` *written by* `<Author>` au lieu de `<String>` *written by* `<String>`. La limite de ces deux approches (DIPRE et Snawball) est qu'elles sont spécifiques à un domaine particulier (Livres pour cette exemple). De plus, ça nécessite une intervention humaine pour la création de l'échantillon.

KnowItAll [CDSE05] est le premier système d'extraction indépendant du domaine. Il n'utilise pas d'échantillon annoté manuellement mais annote automatiquement le texte en utilisant les patterns de Hearst [Hea92]. Ces derniers sont de simples patterns textuels, tels que `"X such as Y"` and `"X, including Y"`. KnowItAll les utilise pour extraire les relations à partir du texte et découvrir de nouvelles relations-instances de façon entièrement automatisée.

TextRunner [BCS+07] n'utilise pas de relations prédéfinies, il les découvre automatiquement durant le processus d'extraction. Il découvre les relations ainsi que leurs instances à la volée. Pour plus de literature sur la gestion des systèmes d'extraction d'information à partir du Web textuel, voir le survey [Sar08].

D'autres systèmes dans ce domaine, comme YAGO [SKW07] and DBpedia [ABK+07] extraient les relations en utilisant des règles d'extraction de faits à partir de sources structurées comme Wikipedia. YAGO est une base de connaissance structurée *(ontology)* construite à partir du contenu de Wikipedia et de la hiérarchie de WordNet [Fel98]. Elle contient plus de 2 millions d'entités et 20 millions de faits. Les entités et les relations sont extraites principalement à partir des catégories et des infoboxes de Wikipedia, alors que la hiérarchie des classes est déduite de WordNet. Pour plus de détail sur les différentes techniques de création de bases de connaissances à partir du Web ou du texte, voir le tutoriel [LSS+10].

2.1.4 Extraction d'information à partir du Web structuré

D'après les travaux existants, la méthodologie d'extraction de données structurées à partir du Web structuré peut être résumée en trois tâches dépendantes les unes des autres : (a) l'identification de la partie la plus riche en données dans la page, (b) l'extraction d'information, et (c) l'interprétation sémantique des données extraites.

La première tâche consiste à la localisation des régions les plus riches en données dans la page. Les pages Web contiennent souvent des informations qui ne sont pas intéressantes pour l'extraction, telles que les publicités, les liens de navigations, etc. De plus, ces informations ont tendance à rendre le processus plus long, et peuvent parfois entrainer des erreurs dans les résultats d'extraction. D'où la nécessité de supprimer ces informations avant le processus d'extraction. La deuxième tâche est le cœur de cette thèse, elle consiste à repérer les données importantes dans la page et les extraire. La dernière tâche consiste à définir la sémantique des données extraites. Comme mentionné précédemment, certains systèmes ne s'intéresse pas à la sémantique des données extraites. Comme nous le verrons, nous voulons utiliser cette tâche avant la deuxième tâche (tâche d'extraction) afin d'extraire uniquement les données ciblées.

2.1.4.1 Sources Web structurées (Encodage)

Dans le Web structuré, les pages HTML sont souvent générées d'une manière dynamique par des scripts qui récupèrent les informations en interrogeant une base de données. Les données sont ensuite `encodées` dans les pages en sortie en utilisant un template prédéfini (voir la Figure 2.1).

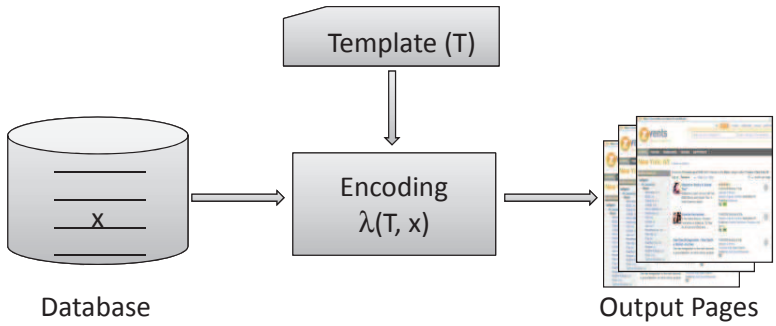


FIGURE 2.1 – Génération de pages Web structurées

2.1.4.2 Extracteur Web (Décodage)

L'extracteur, aussi connu sous le nom de *wrapper*, est un script ou un ensemble de règles d'extraction qui permettent d'extraire des données structurées à partir d'une collection de pages Web générées à partir d'une base de données en se basant sur le même template. Le wrapper s'appuie sur la structure de la page pour repérer les données et construit un template commun pour toutes las pages. Cela représente une sorte de processus inverse à la génération de pages, où le template est `décodé` à partir de pages afin de restaurer les données et déduire le template (voir la Figure 2.2). Comme il n'existe pas de schéma global pour toutes les pages, cela nécessite la construction d'extracteurs appropriés à chaque site.

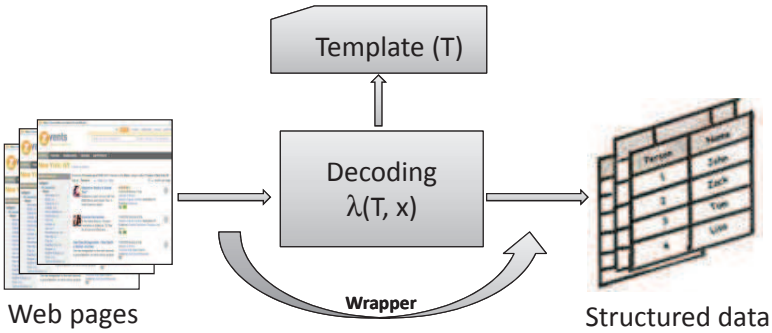


FIGURE 2.2 – Extraction à partir de pages Web structurées

2.1.4.3 Types de pages Web structurées

Pour la tâche d'extraction, il est utile de distinguer entre deux types de pages riches en données : les pages listes *(list pages)* et les pages détaillées *(detail pages)*.

– **Les pages listes :** encodent une ou plusieurs listes de *records* ou *(objets)*, sachant qu'un groupe de records partageant des descriptions similaires sont présentés dans la même liste, aussi appelée *région de données* spécifique de la page. Une page peut contenir plusieurs régions de données structurées et au sein de chaque région les contenus des records sont encodés en utilisant le même schéma. Un exemple d'une page liste est donné dans la Figure 2.3, pour une région d'une page de `amazon.com` contenant trois livres encodés en utilisant la même séquence de balises HTML.

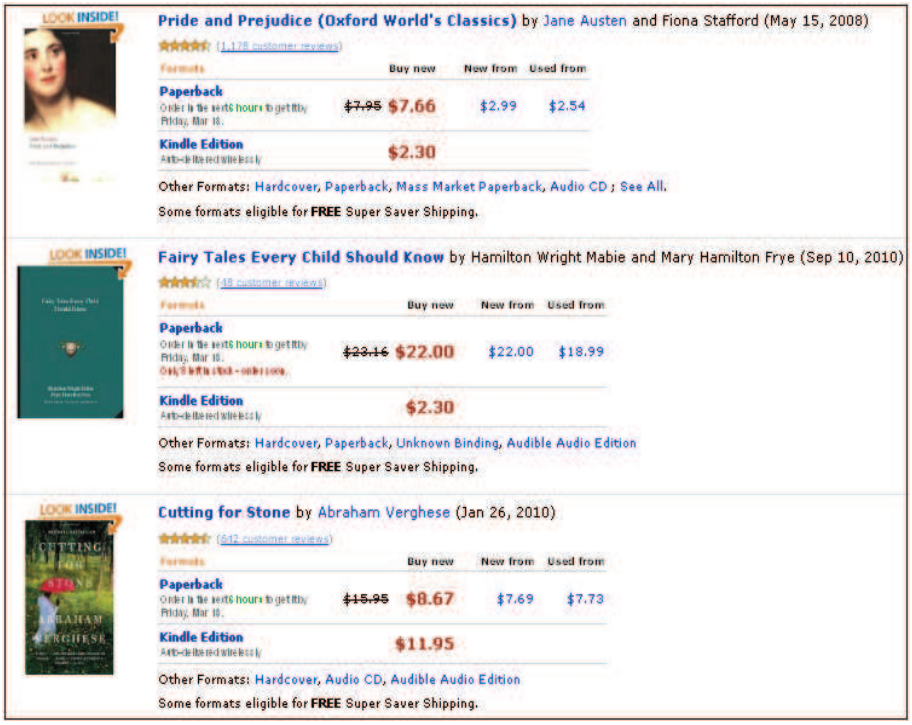


FIGURE 2.3 – Un segment d'une page liste avec trois records

– **Les pages détaillées :** se focalisent sur un seul objet (voir la page de la Figure 2.4), tout en donnant une description plus détaillée des données.

Il assez fréquent d'avoir les deux types de pages qui figurent dans le même site Web. Le premier type sert à donner une idée générale de l'objet (par exemple, affiche les informations principales comme titre, auteur, etc.). Le deuxième type est complémentaire aux pages listes en donnant plus de détails sur les objets (par exemple, les détails d'expéditions).

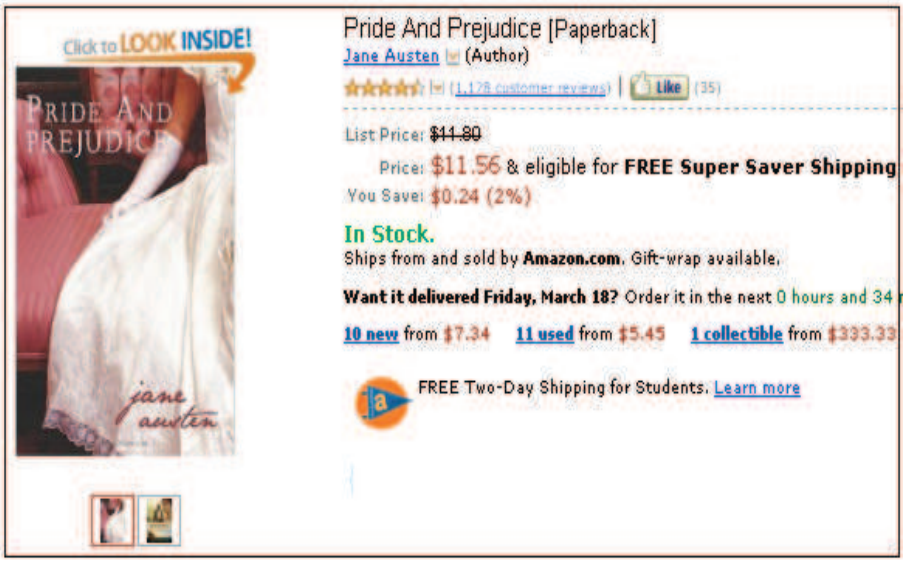


FIGURE 2.4 – Un segment d'une page détaillée

2.1.5 Systèmes de génération de wrappers

Une analyse détaillée des différents travaux sur l'extraction d'information à partir de sources semi-structurées est proposée dans [KS06]. Ces travaux peuvent être classés par rapport au degré d'automatisation, comme suit : manuels, induction de wrappers (supervisés et semi-supervisés) et automatiques (non supervisés). Nous nous référerons à l'architecture proposée par les auteurs de [KS06], représentée dans la Figure 2.5.

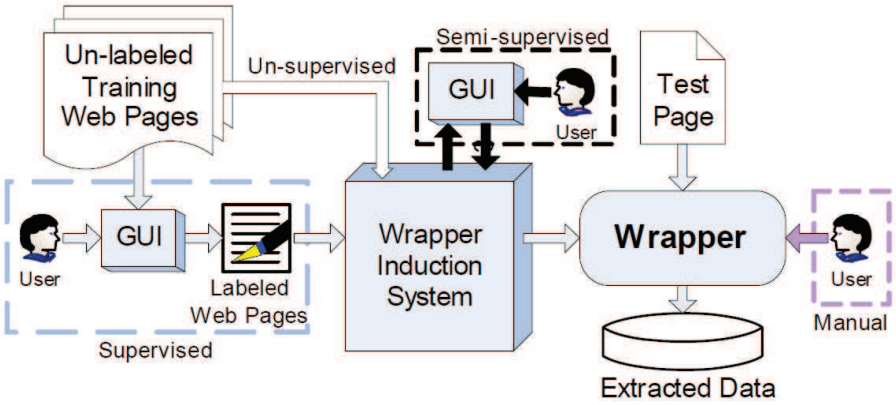


FIGURE 2.5 – Architecture générale des systèmes de génération de wrappers [KS06]

Les approches *manuelles* prennent en entrée des pages annotées manuellement par l'utilisateur pour déterminer les données à extraire, construisent le wrapper en utilisant soit des langages de programmation [HB02, Sod99] ou des plateformes visuelles

(WICCAP [LN04], Wargo [RPA+02], Lixto [GKB+04]).

Les approches *supervisées* intègrent des techniques d'apprentissage appelées *"induction de wrappers"* qui permettent de générer un ensemble de règles d'extraction en utilisant un ensemble de pages annotées manuellement (WIEN [Kus97], Softmealy [HD98], Stalker [MMK99]).

Les approches semi-supervisées (OLERA [CK04], Thresher [HK05]) arrivent à réduire le taux d'intervention humaine par l'acquisition d'un échantillon de pages annotées par l'utilisateur, ou en utilisant des approches semi-automatiques qui n'utilisent pas d'échantillon annoté mais qui interagissent avec l'utilisateur pour choisir les données à extraire (IEPAD [CL01]).

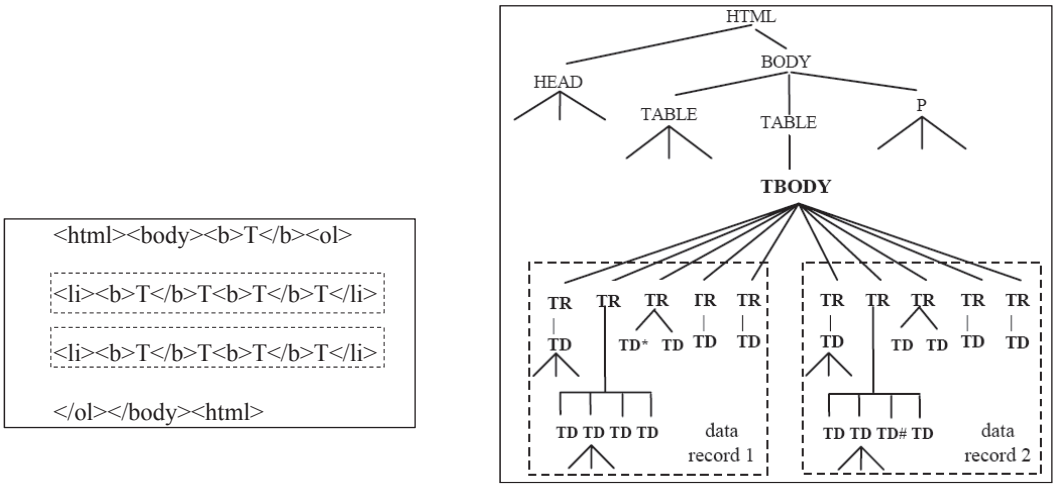
Les approches *non supervisées* (DeLa [WL03], DEPTA [ZL05], RoadRunner [CMM01], ExAlg [AGM03], G-STM [JL10]) sont des approches automatiques, qui permettent l'extraction des données sans intervention humaine durant le processus d'extraction et sans utilisation des échantillons annotés. Les annotations sont faites manuellement après l'extraction pour la reconnaissance des données extraites.

Le principal avantage des techniques d'induction de wrappers est qu'elles extraient uniquement les données annotées par l'utilisateur. Cependant, ces approches sont difficilement applicable à grande échelle, c-à-d, pour un nombre important de sites, car ça nécessite un effort important pour l'annotation manuelle des pages. Contrairement, les techniques d'extraction automatiques sont applicables à grande échelle, néanmoins, elles peuvent extraire une grande quantité de données inutiles.

Dans la partie qui suit, nous allons détailler le fonctionnement des systèmes non supervisés les plus importants dans la littérature. Nous les avons classés en fonction de la technique qu'ils utilisent pour l'inférence de wrapper : (1) alignement des records, qui consiste à examiner plusieurs records en parallèle afin d'identifier les patterns répétitifs, (2) grammaire d'inférence, qui consiste à inférer un ensemble d'expressions régulières pour le code HTML, (3) fréquence d'apparition de strings, qui consiste à identifier les strings répétitifs dans les pages, et (4) caractéristiques visuelles, qui consiste à utiliser les informations visuelles des pages afin de détecter les segments riches en données.

2.1.5.1 Alignement de plusieurs records

Un des défis majeurs de l'extraction d'information à partir du Web consiste à trouver une approche efficace pour la reconnaissance des patterns répétitifs dans les pages HTML. Certains travaux s'appuient sur l'alignement des records existants dans la page, en se basent principalement sur l'alignement de séquences de strings ou d'arbres de ces différents records. La Figure 2.6(a) présente un exemple de comparaison de strings *(string matching)*. La Figure 2.6(b) présente un exemple de comparaison d'arbres *(tree matching)*.



(a) Exemple d'alignement de strings

(b) Exemple d'alignement d'arbres

FIGURE 2.6 – Le matching par l'alignement de plusieurs records d'un segment de page

Comparaison de strings

Le *string edit distance*, ou *Levenshtein distance* est la technique la plus utilisée pour la comparaison de string (string matching). La distance de Levenshtein mesure la similarité entre deux chaînes de caractères, en calculant le nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour passer d'une chaine à l'autre.

IEPAD [CL01]

IEPAD, acronyme d'***I**nformation **E**xtraction based on **PA**ttern **D**iscovery*, comme son nom l'indique il permet l'extraction d'information par la découverte de patterns. IEPAD est l'un des premiers systèmes d'extraction d'information qui génère des patterns à partir de pages Web non annotées. Le système peut identifier automatiquement les records par les patterns répétitifs et l'alignement de plusieurs séquences de strings (voir la Figure 2.6(a)). La découverte de patterns répétitifs est réalisée en utilisant une structure de données appelée *arbres PAT* (connue sous le nom de *Patricia trees*, Practical Algorithm to Retrieve Information Coded in Alphanumeric) [Mor68], construite à partir des suffixes de strings. IEPAD comprend trois composantes principales :

- *Découverte de patterns :* prend une page en entrée et découvre les patterns qui contiennent les données ciblées à extraire.
- *Génération de règles :* l'utilisateur intervient pour sélectionner des patterns qui l'intéresse. Puis, le générateur de règles enregistre ces patterns et les sauvegarde

comme étant des règles d'extraction.

– *Extraction de données :* extrait les données souhaitées à partir des pages similaires en fonction des règles d'extraction choisies par l'utilisateur.

Cependant, cette approche reste semi-automatique vu que c'est l'utilisateur qui choisit les règles d'extraction. De plus, elle s'applique uniquement pour les pages listes, car l'inférence de règles se fait sur une seule page en entrée.

Comparaison d'arbres

Plusieurs travaux se sont inspirés du système IEPAD [CL01], en utilisant l'idée de la découverte de patterns répétitifs par l'alignement des strings similaires, comme DeLa [WL03], ou par l'alignement d'arbre, comme MDR [LGZ03] et DEPTA [ZL05], que nous allons détailler ci-dessous.

MDR [LGZ03], DEPTA [ZL05]

Chaque page HTML peut être convertie en un arbre DOM HTML en utilisant sa structure hiérarchique des balises. MDR [LGZ03] (**M**ining **D**ata **R**ecords) utilise cette structure pour l'identification des records. Il permet de trouver les records dans la page, en trois étapes principales : (1) construire la structure arborescente des balises de la page en utilisant le code source HTML, (2) identifier les régions de la page qui contiennent des records similaires, et (3) détecter les records.

La méthode consiste à identifier les *nœuds généralisés*, qui sont une séquence de nœuds adjacents ayant le même parent (on note que le *nœud généralisé* peut être une région de donnée). MDR compare les nœuds d'une manière récursive en parcourant l'arbre en profondeur, commençant par la racine. Sachant que toutes les combinaisons des nœuds fils sont comparées afin de trouver quels sont les nœuds qui peuvent être regroupés. La comparison de nœuds est effectuée en utilisant soit le *string edit distance* ou soit le *tree matching*. En outre, chaque nœud est examiné de manière récursive pour découvrir des structures de balises imbriquées.

Toutefois, il est important de souligner que dans certaines pages le code source HTML peut être mal formé (des balises erronées dans le code source HTML, telles que des balises manquantes, des balises non fermées, etc.). Ceci rend difficile : (i) la construction d'arbres DOM corrects, (ii) la détection de records corrects. L'utilisation des informations visuelles de la page pour la construction des arbres dans DEPTA [ZL05] résout ce problème. DEPTA a été proposé afin d'améliorer le système MDR [LGZ03]. C'est un système non supervisé qui permet l'extraction d'information en deux étapes : (1) l'identification des différents records dans la page en utilisant les informations visuelles de la page, et (2)

l'alignement et l'extraction des attributs (valeurs) à partir des records identifiés.

Toutes ces techniques (IEPAD [CL01], MDR [LGZ03], DEPTA [ZL05]) s'appliquent pour une seule page liste, ainsi les patterns découverts sont utilisés pour l'extraction à partir des autres pages listes de la même source Web. Cependant, l'utilisation d'une seule page pour l'inférence de wrapper peut causer certaines limites, telles que :

- Elles permettent de découvrir que les patterns existants dans la page en entrée. L'utilisation de plusieurs pages peut améliorer l'extraction, comme la découverte des parties optionnelles ou des informations complémentaires qui n'existent pas dans une seule page.
- Elles s'appliquent que pour les pages listes, et pas pour les pages détaillées. De plus, la page liste doit avoir au minimum trois records pour que l'algorithme fonctionne. Une alternative possible pour l'extraction à partir de pages détaillées est de considérer chaque page détaillée comme un record et ainsi construire un arbre DOM avec un nœud racine artificiel, chaque page comme étant un sous arbre de cette racine.

Les techniques que nous allons cités ci-après (RoadRunner [CMM01], ExAlg [AGM03], G-STM [JL10]) exploitent plusieurs pages en entrée pour l'extraction.

2.1.5.2 Inférence de grammaires

RoadRunner [CMM01]

La génération de wrappers dans RoadRunner consiste à inférer une grammaire (ensemble d'expressions régulières) pour le code HTML en utilisant un ensemble de pages HTML. RoadRunner infère le wrapper en fonction des similarités et des différences dans les pages. Il compare deux pages à la fois (un wrapper et une page), en alignant les tokens et en capturant les différences (mismatches). Dans la première itération, une simple page est utilisée comme wrapper (Figure 2.7). Ce sont ces mismatches qui vont guider l'algorithme, afin d'inférer le schéma de données (template). Il y a deux types possibles :

- *String mismatches :* ils sont utilisés pour découvrir les attributs (les données à extraire). Cela se produit lorsque la même position au sein du wrapper et la page contient des chaînes de caractères différentes.
- *Tag mismatches :* ils sont utilisés pour découvrir les parties itératives (+) et optionnelles (?). Cela se produit lorsque la même position au sein du wrapper et de la page contient soit différentes balises HTML ou l'un a une balise et l'autre a une chaîne de caractères.

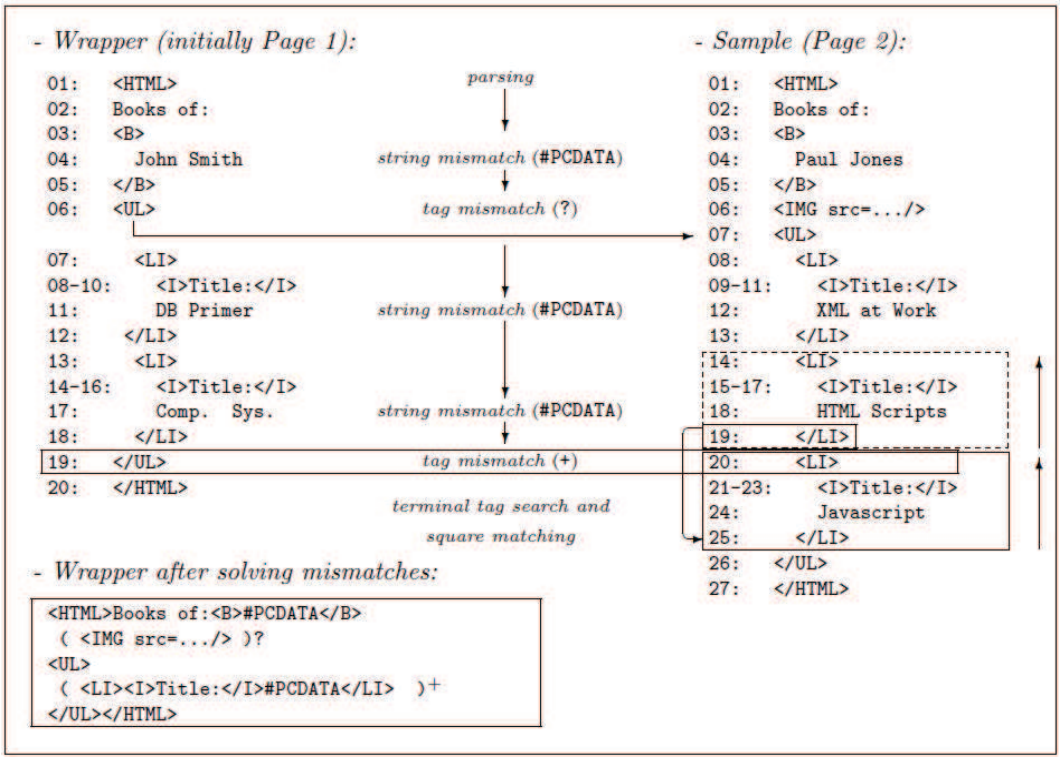


FIGURE 2.7 – Exemple d'exécution de l'algorithme RoadRunner

Un exemple d'exécution de l'algorithme est illustré dans la Figure 2.7. Durant l'analyse des deux pages, le wrapper et les données à extraire sont inférés. Plus précisément,

1. *les strings mismatches* sont découverts afin de détecter les données à extraire, par exemple, la chaine de caractère "John Smith" et "Paul Jones" (ligne *4*). Le wrapper est généré en marquant les nouveaux champs découverts, en remplaçant la chaine "John Smith" par `#PCDATA` (vu que c'est la première page qui est utilisée comme wrapper dans cette exemple). Le même scénario se produit pour les chaines "DB Primer" et "XML at Work". En ce qui concerne les chaines similaires, comme "Books of :" (ligne *2*), le wrapper ne génère pas de nouveaux champs. Ces chaines sont plutôt considérées comme étant des informations qui feront partie du template. Sachant que dans chaque étape le wrapper est généralisé avant de reprendre l'analyse des pages.

2. *les tags mismaches* sont découverts à leur tour, afin de déterminer les parties optionnelles et itératives. Dans cette exemple, le premier tag mismatch apparait sur les balises `<UL>` et `<IMG.../>` (ligne *6*). Dans ce cas, le wrapper est généralisé en introduisant un pattern de la forme `(src=.../> <IMG)?`. En ce qui concerne

les parties itératives, dans cet exemple, les deux pages HTML contiennent chacune nom d'auteur et une liste des titres de livres. Durant l'analyse, le tag mismatch apparait sur les balises `<UL>` et `<LI>` (ligne *19* et *20* respectivement). Ce mismatch provient de la différence des cardinalités pour les listes de livres (deux livres pour le wrapper et trois livres pour la page). Ainsi le pattern répétitif est `<LI><I>Title:</I>#PCDATA</LI>`. Dans ce cas, le wrapper est généralisé en introduisant un pattern de la forme `(<LI><I>Title:</I>#PCDATA</LI>)+`. Le wrapper final dans cet exemple contient à la fois une partie itérative et optionnelle.

G-STM [JL10]

Les algorithmes de tree matching cités précédemment échouent dans le traitement des listes imbriquées ayant des patterns répétitifs. Pour pallier ce problème G-STM (**G**eneralized-**STM**), inspiré par l'algorithme STM [Yan91](**S**imple **T**ree **M**atching), intègre de nouvelles heuristiques qui permettent la détection de listes, en se basant sur la génération de grammaires. GST-M est l'un des systèmes les plus récents, avec des performances qui dépassent celles des systèmes précédents.

D'une manière récursive, à chaque niveau de l'arbre DOM, G-STM attribue un symbole pour les nœuds en fonction de leurs mesures de similarité. Les nœuds similaires recevront le même symbole, respectivement, les nœuds différents recevront des symboles différents. Par conséquent, deux strings de symboles sont créés, un pour chaque arbre. Ces strings sont ensuite utilisés pour la génération de grammaires et permettent la détection de listes. Théoriquement, dans cette étape les listes sont détectées, et sont représentées par une partie itérative (+) dans la grammaire. Puis, les parties itératives dans les deux grammaires sont comparées. Si elles sont équivalentes, leurs éléments sont transformés et réunis donnant une mesure de similarité plus précise. En outre, chaque symbole est un *mapping* des nœuds structurels équivalents.

Un pseudo-code est fourni dans Algorithme 1. Un *mapping* M, entre un arbre A de taille k et un arbre B de taille n est un ensemble de paires (i, j), un pour chaque arbre, satisfont les conditions suivantes pour chaque (i_1, j_1) et $(i_2, j_2) \in M$:

- $i_1 = i_2$ ssi $j_1 = j_2$
- si i_1 est sur la gauche de i_2, alors j_1 est sur la gauche de j_2
- si i_1 est l'ancêtre de i_2, alors j_1 est l'ancêtre de j_2

La correspondance maximale entre les deux arbres est le nombre maximum de paires (i, j) équivalentes. Afin de détecter les listes, pour chaque paire de nœuds de A et B, l'algorithme retourne le tuple $(score, nodes_A, nodes_B)$, où le *score* est leurs scores de matching, $nodes_A$ et $nodes_B$ est le nombre de nœuds (la taille) de A et B respectivement.

Algorithm 1 Algorithme G-STM

if les racines des deux arbres A et B contiennent des symboles différents (nom des balises) **then**
  return $(0, nodes_A, nodes_B)$
else
  Initialisation : $m[i,0] \leftarrow 0$ for $i = 0, \ldots k$; $m[i,0] \leftarrow 0$ for $j = 0, \ldots n$;
  for $i = 1$ to k **do**
    for $i = 1$ to n **do**
      $W[i][j] = G - STM(A_i, B_j)$
    end for
  end for
  $(W, nodes_A, nodes_B) \leftarrow DetectLists(W, A, B)$
  for $i = 1$ to k **do**
    for $i = 1$ to n **do**
      $m[i][j] = max(m[i-1][j], m[i][j-1], m[i-1][j-1] + W[i][j].score)$
    end for
  end for
  return $(m[k][n] + 1, nodes_A, nodes_B)$
end if

L'algorithme sauvegarde le score et le nombre de nœuds dans la matrice W. Chaque cellule de la matrice contient trois valeurs : le score, la taille de l'arbre au $i^{ième}$ fils du sous-arbre A, respectivement, la taille de l'arbre au $j^{ième}$ fils du sous-arbre B.

2.1.5.3 Fréquence d'apparition des tokens

ExAlg [AGM03]

ExAlg infère le template en identifiant les parties statiques à partir d'une collection de pages en entrée. Ces parties statiques seront considérées comme faisant partie du template. L'algorithme d'extraction considère les pages comme étant un ensemble de tokens, texte ou balise HTML, et calcule leurs fréquences d'apparition dans ces pages.

ExAlg se base sur le calcule des *vecteurs d'occurrences* contenant le nombre d'occurrences d'un token, qui sont utilisés par la suite pour créer des *classes d'équivalence*. Ces classes sont un groupe maximal de tokens partageant les mêmes vecteurs d'occurrence.

Dans Algorithme 2, on montre les étapes principales utilisées par ExAlg pour extraire les données et inférer le template. L'idée principale est de trouver les classes d'équivalence qui contiennent les tokens les plus fréquents afin de découvrir le schéma de pages. Pour

Algorithm 2 Algorithme ExAlg

1: **input :** collection de pages
2: **output :** template, données structurées
3: différencier les rôles de tokens en utilisant le schéma `HTML`
4: **repeat**
5: trouver les classes d'équivalences (Equivalence classes EQs)
6: traiter les EQs invalides
7: différentier les rôles de tokens en utilisant les EQs
8: **until** fixpoint
9: construction du template et extraction de données

cela, déterminer et différencier les rôles des tokens est une tâche cruciale pour inférer le schéma de pages. Les rôles des tokens sont déterminés en utilisant :

- `le schéma HTML:` les tokens ayant la même valeur et le même path dans l'arbre DOM de la page ont le même rôle (ligne *3*),
- `les classes existantes:` les autres rôles de tokens sont attribués en boucle, en se basant sur leurs positions d'apparition dans les classes d'équivalence (lignes *4-8*).

L'idée générale de l'algorithme est de garder que les classes fréquentes et larges pour la construction du template. Cependant, pour éviter que des tokens forment accidentellement une classe d'équivalence non valide, les EQs avec un *support* (le nombre de pages contenant le token) et *size* (le nombre de tokens dans une EQ) insuffisants sont filtrées. Ces deux paramètres sont attribués par l'utilisateur. En outre, pour se conformer à la structure hiérarchique du schéma de données, dans chaque itération, les classes invalides sont écartées (voir Algorithme 2).

Les EQs sont dites valides si elles sont : (i) mutuellement *imbriquées*, les occurrences d'une paire de classes d'équivalence ne se chevauchent pas, ou toutes les occurrences de la première classe se retrouvent dans la même position par rapport aux occurrences de la deuxième classe (ou vice versa), et (ii) *ordonnées*, les tokens dans chaque EQ doivent être ordonnés, selon le même ordre d'apparition dans la page. Seules les EQs valides sont utilisées pour la construction du template.

2.1.5.4 Caractéristiques visuelles

Plusieurs travaux utilisent les caractéristiques visuelles de la page pour l'identification des sections contenant les records (ViNTs [ZMW+05], MSE [ZMY06], ViPER [SL05]) ou l'extraction d'information à partir de ces records (ViDE [LMM10]). Ces techniques sont entièrement automatiques. Elles supposent que la page Web contient au moins deux records spatialement consécutives d'une région de données, affiche des données

structurellement et visuellement similaires.

Ces travaux se sont inspirés de VIPS [CYWM03] (**VI**sion-based **P**age **S**egmentation), qui a démontré dans ces expérimentations que l'utilisation des informations visuelles peut reconstruire facilement la structure de la page. VIPS extrait la structure sémantique d'une page Web en combinant la structure DOM et la représentation visuelle de la page. Chaque page est représentée sous forme d'une *structure hiérarchique (arbre)*, ou chaque nœud correspond à un bloc. Chaque bloc dans l'arbre correspond à un segment de la page, et il lui sera attribué une valeur qui indique le *degré de cohérence* du contenue du bloc, en se basant sur les informations visuelles. Ces blocs sont déterminés en se basant sur : (i) l'arbre DOM HTML de la page, et (ii) les séparateurs, en utilisant les lignes horizontales et verticales de la page Web. Finalement, la structure sémantique de la page est construite en utilisant ces séparateurs. Dans le navigateur Web chaque élément HTML (balises ouvrantes, fermantes, attributs, valeurs) est traduit comme étant un rectangle. Les informations visuelles peuvent êtres obtenues par le navigateur après le chargement du code HTML de la page. De plus, même si le code HTML est mal formé (tags manquants, tags non fermés, etc.), les rectangles peuvent être détectés correctement par le navigateur en utilisant les coordonnés des bornes de chaque élément `HTML`. La structure hiérarchique peut être ainsi construite, en se basant sur les différents rectangles imbriquées résultants des balises imbriquées.

ViNTs [ZMW+05], acronyme de **V**isual **i**nformation a**N**d **T**ag **s**tructure, génère des règles d'extraction en utilisant les informations visuelles de la page et la structure des balises HTML. Il utilise en premier lieu le contenu visuel pour identifier les différentes régularités dans la page. Ensuite, il les combine avec les régularités de la structure des balises. MSE [ZMY06] (**M**ultiple **S**ection **E**xtraction) permet d'identifier et d'extraire toutes les sections dynamiques dans une page Web.

ViPER [SL05] (**Vi**sual **P**erception-based **E**xtraction of **R**ecords) identifie les différents records en intégrant également les informations visuelles de la page. L'alignement de ces records utilise une technique d'alignement de plusieurs séquences.

Cependant, ces travaux se focalisent sur l'extraction des records, sans considérer l'extraction des attributs (données), contrairement à ViDE [LMM10] (**Vi**sion-based **D**ata **E**xtractor), qui extrait les informations à partir de ces records.

2.2 Découverte et sélection de sources

Le problème de la découverte de sources que nous traitons dans cette thèse soulève deux principaux défis :

1. *Découverte de sources structurées :* consiste à déterminer (de façon non supervisée) si une source contient des données structurées ou non. Autrement dit, au moment de la collecte des sources à partir du Web (crawl), arriver à découvrir qu'une source (ensemble de pages) publient des contenus structurés, par un même template de présentation. Elles pourraient être générées automatiquement à l'aide ce template (la majorité des cas en pratique) ou remplies manuellement.
2. *Indexation de sources :* consiste à déterminer si une sources est susceptible de contenir des instances d'une description donnée. Ceci nécessite une technique d'indexation adéquate, qui permet de trouver les sources structurées pertinentes pour toute description spécifiée par l'utilisateur.

Certains travaux ont tenté de résoudre le problème de la découverte de sources contenant des pages partageant la même structure. Dans [CMM05, BDM11], les auteurs s'appuient principalement sur le clustering de pages Web, en se basant sur la structure des pages et leurs contenus. Le processus de découverte commence d'une URL comme point de départ, visite un nombre représentatifs de pages Web et détermine par le bais du clustering les parties des pages qui présentent de fortes similarités dans leurs structures. Toutefois, ces techniques supposent que l'ensemble de pages est soit donné directement ou soit obtenu par un point d'accès à la source. Découvrir ces sources d'une manière automatique reste donc un défi en soi.

Le processus de découverte de sources Web peut être guidé en utilisant une description des données ciblées *(focused crawling)*. Le problème de se focaliser à un domaine particulier (collecteurs ciblés) à été abordé dans plusieurs travaux, qui visent à rechercher et récupérer à partir du Web un sous-ensemble concernant à un domaine spécifique, en se basant sur des techniques de classification [CvdBD99, CGMP98] ou sur des techniques d'analyse des liens des documents qui sont étroitement liés aux pages ciblées [DCL+00]. Pour plus de détail sur les différents algorithmes de *focused crawling*, voir le survey [MPS04].

Le problème d'indexation et de sélection de sources pertinentes pour la tâche d'extraction est lié aux travaux existants dans la littérature sur la recherche de documents à partir du Web textuel, Web caché ou le Web structuré.

Web textuel. Le problème de la sélection de documents à partir du Web textuel a été principalement traité pour l'extraction de relations simples (binaires) en se basant sur la découverte de patterns qui apparaissent dans le texte (comme KnowItAll [CDSE05],

KnowItNow [CDSE05], Snowball [AG00], TextRunner [BCS+07]).
KnowItAll [CDSE05] récupère une liste d'URLs contenant les termes de la requête. Par exemple, pour la requête `"cities such as properNoun(head(<Noun phrase>))"`, il récupère un ensemble de pages Web contenant les termes "cities such as". Ensuite, pour chaque page Web dont l'URL est sélectionnée, il extrait les instances en utilisant la règle associée à la requête, i.e., les noms de villes pour cette requête.
KnowItNow [CDSE05] sauvegarde les documents en local et construit des index inversés (Bindings Engine) afin de récupérer un ensemble de relations répondant à la requête.
TextRunner [BCS+07] sélectionne les pages Web qui contiennent les relations indépendamment du domaine. QXtract [AG03, IAJG06] se base sur une technique de génération de requêtes pour analyser et filter des documents dans le Web, afin de sélectionner ceux qui sont prometteurs pour l'extraction d'une relation donnée. Ensuite, l'extraction est réalisée en utilisant un système d'extraction d'information (Snowball [AG00]).

Toutefois, notre objectif est différent de ces lignes de recherche, comme nous voulons récupérer des sources structurées qui publient des instances d'un certain schéma, et nous avons un accès complet à leur contenu (par exemple, pendant le crawl) pour les indexer.

Web caché. Aussi connu sous le nom du *Deep Web*, contenant des données qui ne peuvent être accessible que par l'intermédiaire de formulaires Web. Certains travaux ont considéré la problématique de sélection de pages à partir du Web caché, en essayant de découvrir la structure et le domaine du formulaire [RGM01]. Les auteurs de [SMM+08] tentent de comprendre ces formulaires en reliant leurs champs aux concepts appropriés. D'autres travaux [HPZC07, MAAH09] ont proposé des techniques qui permettent d'accéder aux données du Web caché, souvent par la formulation automatique de requêtes appropriées pour remplir les formulaires.
SourceRank [BK11] sélectionne les sources les plus pertinentes et les plus fiables à partir du Web caché, en se basant sur une mesure globale qui calcule la pertinence et la fiabilité d'une source. Cette mesure s'appuie sur la concordance des réponses apportées par différentes sources. En outre, les méthodes de sélection de base de données pour les bases de données relationnelles (tel que [IG04]) évaluent la pertinence des sources en se basant sur les similarités entre la requête et les réponses potentielles.

Web structuré. Dans le projet WebTables [CHW+08, CHZ+08] de Google, les tables HTML sont collectées automatiquement du Web afin de construire un grand corpus de base de données. À notre connaissance, Flint [BCMP08] est le seul système qui a essayé de résoudre la problématique de trouver des pages Web contenant des instances d'objets

structurés. Flint exploite les informations présentées dans les pages pour découvrir, collecter et indexer des pages Web qui publient des données représentant une instance d'un certain schéma. Cependant, l'inconvénient de ce système est qu'il fonctionne uniquement pour les sites qui représentent des objets dont les composantes sont données dans le style de paires (*attribut* : *valeur*). Par exemple, pour un objet de type `"Book"`, les informations dans la page doivent être présentées dans le style (book title : valeur, author : valeur, etc.), comme dans Figure 2.8.

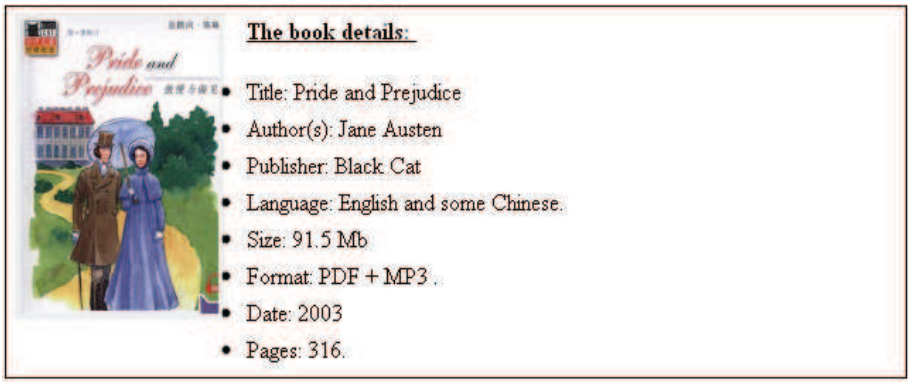


FIGURE 2.8 – Segment d'une page Web présentant une instance d'un objet `Book`

Or la majorité des sites structurés sur le Web ne présentent pas les instances de structures d'objets de cette façon. Nous voulons combiner la représentation structurelle des objets dans les pages et l'annotation automatique des entités pour la sélection des sources pertinentes.

2.3 Synthèse et positionnement

Dans ce chapitre, nous avons passé en revue les travaux connexes aux problématiques liées à la recherche d'information dans le Web : extraction d'information et sélection de sources.

Le processus d'extraction automatique à partir du Web textuel et du Web structuré peut se résumer (selon les travaux exposés précédemment) comme suit :

- *Extraction à partir du* ***Web textuel.*** Documents (pages Web) → annotation → pattern matching → données structurées.
- *Extraction à partir du* ***Web structuré.*** Documents (pages Web) → découverte de patterns → données structurées → annotation.

Nous voulons tirer profit des travaux existants dans ces deux domaines de recherche et d'exploiter l'annotation sémantique pendant le processus d'extraction. Plus précisément, de guider l'extraction en utilisant un domaine de connaissances (les annotations sémantiques) qui vont permettre de reconnaitre et d'extraire de manière plus précise uniquement les informations qui intéressent l'utilisateur.

Nous avons vu précédemment que le principe des approches *non supervisées* consiste à exploiter les régularités dans les pages pour la génération de wrappers et à se baser principalement sur les parties variables dans une page pour extraire les données. Une importante problématique consiste à distinguer le rôle de chaque `token` dans une page, qui peut être soit une unité de donnée (mot) ou une unité du template (tag). Certains travaux simplifie le processus et considère que chaque tag `HTML` est généré par le template (comme DeLa [WL03], DEPTA [ZL05]), ce qui n'est pas souvent le cas en pratique. RoadRunner [CMM01] et G-STM [JL10] utilisent une approche basée sur l'inférence grammaticale (grammar inference) et le matching de schémas, assumant eux aussi que chaque tag `HTML` est généré par le template, et que de plus les tokens de types strings peuvent également être considérés comme une partie du template. Cependant, G-STM [JL10] intègre une méthode plus robuste pour détecter les listes et il a la capacité de traiter les listes imbriquées. En comparaison, ExAlg [AGM03] a des hypothèses plus flexibles, où les tokens du template sont ceux correspondant aux classes d'équivalences les plus fréquentes. En outre, il a l'approche la plus générale, comme il peut manipuler des parties optionnelles et alternatives des pages.

TurboSyncer [CCZ07] est un système d'intégration qui peut intégrer de nombreuses sources et utiliser les résultats des systèmes d'extraction existants afin de mieux calibrer les extractions futures. D'autres travaux explorent le bénéfice mutuel de l'annotation et l'extraction, pour l'induction de wrappers en se basant sur des pages annotées [ZNW+06, SMM+08], ou pour assister l'extraction en utilisant une ontologie de domaine construite par la correspondance des interfaces d'interrogation de Web sites (formulaires du Web caché) et les records des pages résultants des requêtes [SWL09]. Pour plus de détail sur la gestion du contenu du Web, voir [Liu06].

Toutes ces approches ne sont pas adaptées pour l'extraction ciblée d'objets complexes. D'une part, elles ne s'intéressent qu'à la structure de la page et ne tiennent pas compte de la sémantique des données contenues dans ces pages. D'autre part, elles extraient toutes les informations contenues dans une page. Par la suite, la tâche de reconnaissance et d'annotation sémantique est faite souvent manuellement, pour filtrer et annoter les données désirées. Par ailleurs, vu que les données extraites sont uniquement les données variables des pages Web, il peut y avoir une perte d'information dans le cas où un bout de texte utile est compris dans le modèle de pages. En plus, comme les données extraites

ne portent aucune sémantique, cela peut conduire à extraire des informations de types différents (ex, date et adresse) dans le même attribut du résultat, ce qui rend en suite l'annotation impossible. Notre approche vise à obtenir le meilleur des deux directions, en exploitant à la fois la structure et la sémantique fournie par l'utilisateur dans le processus de génération automatique de wrapper.

Nous proposons dans le Chapitre 4 une nouvelle approche qui permet la reconnaissance et l'extraction d'objets complexes à partir du Web structuré. Notre approche prend en entrée un ensemble de pages non annotées d'un site structuré donné, et une structure décrivant les objets que l'utilisateur souhaite extraire. Elle retourne en sortie des données structurés qui contiennent uniquement les objets désirés. L'originalité de notre approche est qu'elle permet une extraction ciblée en exploitant la structure de la page et une sémantique prédéfinie. L'idée principale n'est pas d'extraire uniquement des données mais des d'objets structurés, qui répondent au mieux aux besoins de l'utilisateur.

Nous proposons dans le Chapitre 5 une nouvelle approche d'indexation qui permet la sélection de sources structurées pertinentes pour une requête utilisateur, dans laquelle il décrit les objets recherchés sous forme d'un schéma, à l'aide de types d'entités simples. Nous exploitons à la fois la structure visuelle des pages pour identifier les différentes régions sémantiques et un domaine de connaissances pour la reconnaissance des entités.

Chapitre 3

Préliminaires et définitions des problèmes

Ce chapitre présente les problématiques abordées de manière détaillée. Nous présentons quelques préliminaires et les différentes définissions que nous allons utiliser tout au long de ce manuscrit. En ce qui concerne le problème de la sélection de sources, nous supposons l'existence de : (1) un ensemble large de sources Web (pas forcément structurées) en entrée (elles pourraient être le résultat d'un crawl), où chaque source représente un ensemble de pages `HTML` décrivant des objets du monde réel, par exemple, des concerts, des annonces immobilières, des livres, etc., (2) une description de la structure des objets recherchés (la requête SOD). Le système recherche automatiquement les sources pertinentes, i.e., celles qui semblent les plus pertinentes pour une extraction. En ce qui concerne le problème d'extraction d'objets structurés, nous supposons que l'utilisateur fournit la description recherchée (SOD) et que la source à partir de laquelle l'extraction doit être faite est choisie, parmi les sources retournées dans l'étape précédente (étape de sélection automatique de sources). Le système extrait automatiquement les objets correspondant à la description recherchée à partir de chaque source sélectionnée.

3.1 Définitions, notations

Nous définissons dans ce qui suit un formalisme par lequel tout utilisateur peut spécifier quelles données doivent être ciblées et extraites à partir de pages `HTML`. Notre exemple d'application se réfère à des objets `concerts`, qui peuvent être considérés comme étant des tuples relationnels formés par trois types d'entités simples (atomiques) : artiste, date et adresse. Nous illustrons dans la Figure 3.1 trois fragments de modèles de pages décrivant ce type d'information.

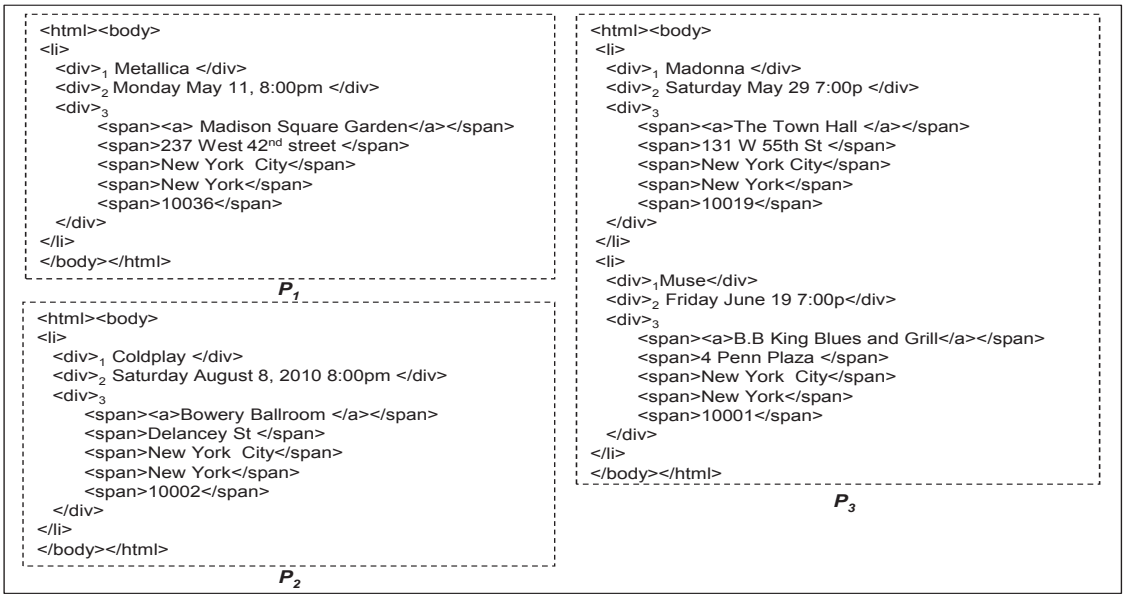


FIGURE 3.1 – Échantillon de pages

3.1.1 Description d'information structurée

Intuitivement, la description d'objets structurés permet aux utilisateurs de décrire des données relationnelles imbriquées avec des contraintes de multiplicités [AHV95]. Plus précisément, un objet est obtenu par la combinaison de différentes composantes apparaissant dans une structure englobante. Ces composantes sont généralement disposées à des distances voisines dans la page (proximité spatiale et proximité au niveau du code source de la page) pour illustrer les relations entre elles.

3.1.1.1 Types, SODs

Un **type d'entité** représente une unité d'information *atomique*, qui est représenté comme une chaine de tokens (mots). Chaque type d'entité t_i est supposé avoir un *reconnaisseur* r_i associé, qui peut être simplement considéré comme une expression régulière ou un dictionnaire de valeurs (thesaurus).

Les reconnaisseurs. En pratique, nous distinguons trois types de reconnaisseurs :

1. Les reconnaisseurs définis par l'utilisateur, où l'utilisateur définit l'expression régulière qu'il souhaite extraire.
2. Les reconnaisseurs prédéfinis par le système, par exemple les adresses, les dates, les numéros de téléphone, etc., pour lesquels des reconnaisseurs existent.

3. Les reconnaisseurs basés sur des dictionnaires, c'est un ensemble de valeurs (instances) associées à un type donné, que nous appelons ci-après les reconnaisseurs *isInstanceOf*.

Les **types complexes** peuvent être définis d'une manière récursive, en se basant sur les types d'entités. Un `type set` est une paire $t = [\{t_i\}, m_i]$ où $\{t_i\}$ désigne un ensemble d'instances du type t_i (pas forcément atomique), m_i désigne la contrainte de multiplicité qui spécifie des restrictions sur le nombre de instances t_i, comme suit :

- $n - m$ pour au moins n et au plus m
- $*$ pour zéro ou plus
- $+$ pour un ou plus
- ? pour zéro ou un
- 1 pour exactement un

Un `type tuple` désigne une collection d'éléments, qui peuvent être de *type set* ou de *type tuple*. Un `type disjonctif` désigne une paire de types mutuellement exclusifs.

Un **SOD** *(Structured Object Description)* désigne tout type complexe, contenant un ou plusieurs *type set* ou *type tuple*. La Figure 3.2 montre un exemple d'un type complexe s composé de : (i) deux types tuples $[t_1, \{t_2\}, t_3]$ et $[t_{31}, t_{32}]$, et (ii) un type set $\{t_2\}$ avec des contraintes de multiplicité arbitraires.

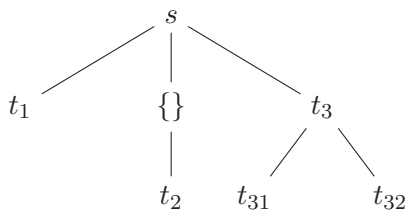
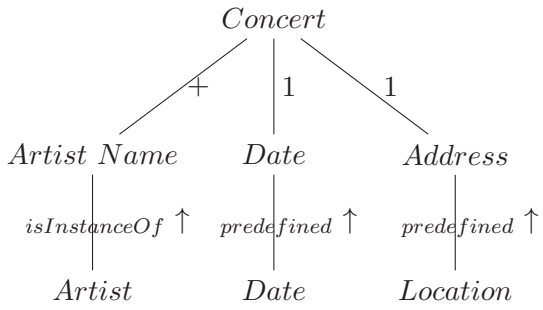


FIGURE 3.2 – Exemple d'un SOD

En pratique, ces SODs pourraient être complétés par des restrictions supplémentaires sous forme de valeur, texte ou règles de désambiguïsation. Ces restrictions peuvent permettre par exemple de spécifier qu'un type d'entité donné peut couvrir la totalité du contenu textuel d'un élément `HTML`, ou d'exiger que deux types *date* doivent être dans un ordre spécifié ou qu'une *adresse* particulière doit être dans une certaine région géographique. Ces contraintes sont omises dans le modèle décrit dans cette thèse et dans nos expérimentations.

3.1.1.2 Instances d'objets

Une *instance* d'un type d'entité t_i est une chaine de caractères associée à un reconnaisseur r_i. Une instance d'un SOD est définie de manière bottom-up, et peut être considérée comme un arbre fini dont les nœuds internes représentent des types complexes. Par exemple, le schéma (SOD) ci-dessous représente une structure possible pour les objets `concerts` : elle est composée d'un type tuple contenant trois types d'entités (un pour la *date*, un pour l'*adresse* et un pour le *nom d'artiste*). Les deux premiers types d'entités (date et adresse) sont associés à des *reconnaisseurs prédéfinis*, puisque ce genre d'information est facile à détecter par des patterns appropriés (dans cet exemple, le reconnaisseur *Date* est attribué pour la reconnaissance des dates, respectivement, *Location* pour les adresses). Le troisième type (nom d'artiste) est associé au reconnaisseur *isInstanceOf* (une liste d'instances de la classe *Artist*). La multiplicité "+" est attribuée pour le nom d'artiste, "1" pour l'adresse et la date.



3.2 Pré-traitement de pages Web

Nous avons défini dans la Section 2.1.4.1 les pages Web structurées et les différents types de pages existants (listes ou détaillées). Souvent, de nombreux segments dans ces pages ne contiennent pas d'informations utiles. Ces informations peuvent rendre le processus d'extraction plus lent, et peuvent même affecter parfois les résultats finaux. Les techniques d'inférences de wrappers suggèrent qu'il est généralement nécessaire de procéder à une étape de pré-traitement de ces pages HTML avant le processus d'extraction.

Dans cette section, nous décrivons les pricipales étapes de pré-traitement effectuées dans `ObjectRunner` – avant le processus de selection de sources et d'extraction d'information – sur les pages HTML. Ces étapes sont : (1) l'extraction du segment de la page le plus riche en données, (2) le nettoyage du code HTML, et (3) la transformation en XML du code source de la page.

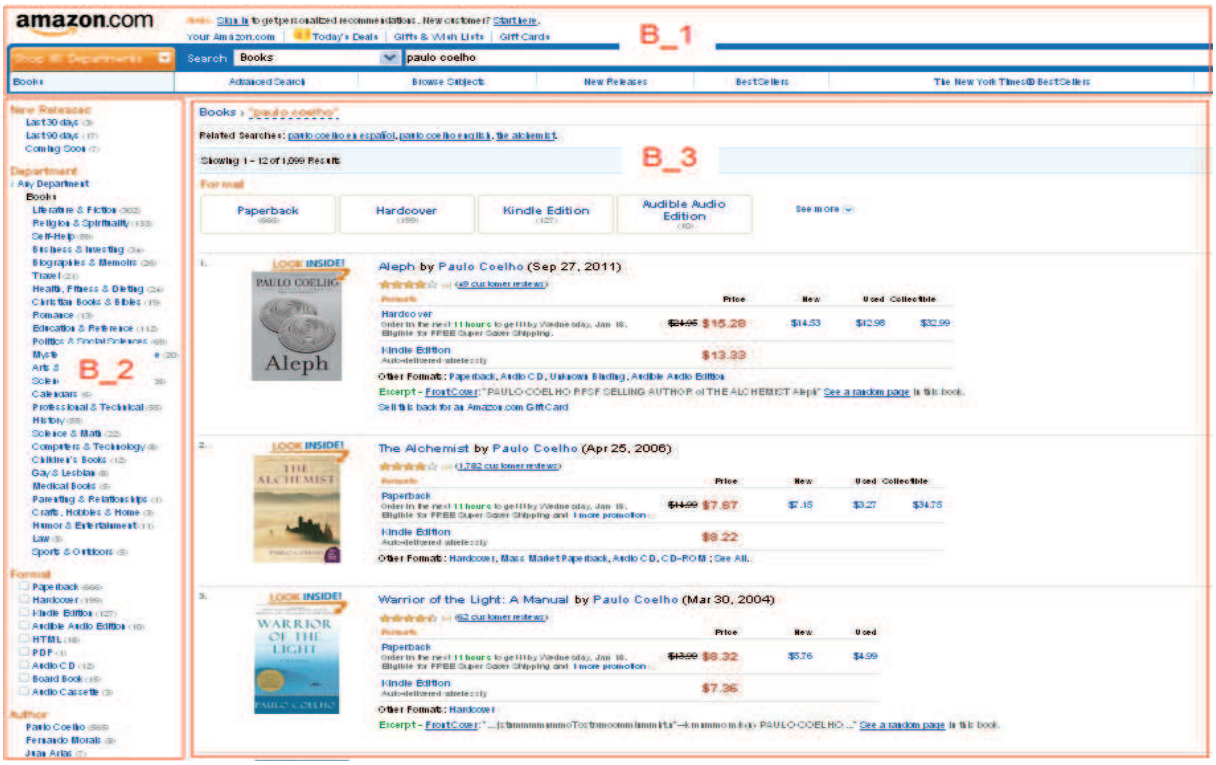


FIGURE 3.3 – Segmentation d'une page Web en blocs

Extraction du segment riche en données. Au delà du nettoyage de pages, la première étape de pré-traitement consiste à l'identification du segment de la page le plus riche en données. Nous appliquons sur la collection de pages une simplification radicale de leur segment "central", celui qui affiche habituellement le contenu principal de la page. Pour cela, nous exploitons les informations visuelles (rendering) de la page. Nous nous appuyons sur un algorithme qui permet la segmentation de la page, à la manière de ce que fait VIPS [CYWM03] et ViNTs [ZMW+05] (voir la Section 2.1.5.4). Dans VIPS, chaque page est vue comme "une structure arborescente" de plusieurs blocs (segments). Ces blocs sont délimités en se basant sur : (i) l'arbre DOM de la page, (ii) les séparateurs de blocs, en utilisant les lignes horizontales et verticales des pages Web. La Figure 3.3 montre un exemple de segmentation visuelle d'une page de amazon (la page est séparée en trois blocs B_1, B_2 et B_3).

Pour toutes nos sources, nous avons appliqué une heuristique simple qui sélectionne comme meilleur segment candidat, le rectangle *le plus grand et le plus central* dans la page. Par exemple, dans la Figure 3.3, le bloc B_3 représente le segment qui sera choisi pour l'extraction.

Cependant, les tailles des blocs, notamment leurs structures peuvent varier d'une page à une autre. Pour cela, nous avons choisi d'identifier le meilleur bloc candidat par son nom de balise, son chemin (position) dans l'arbre DOM (path) et ses noms et valeurs d'attributs, en utilisant toutes les pages existantes dans la source.

Pour minimiser le coût de pré-traitement, cette étape est effectuée sur un échantillon de pages provenant de la même source. Ensuite, les résultats sont généralisés à l'ensemble des pages.

Nettoyage du code HTML. Le nettoyage des pages HTML est nécessaire afin d'éliminer toutes les informations inutiles, telles que les scripts (<script...</script>), les styles (<style...</style>), les commentaires (<!–...–>), les images (<img...</img>), les contenus cachées <input type="hidden"...>), les balises vides
, les espaces blancs, les propriétés des balises, etc.

Transformation en XML. Vu que les documents `HTML` ne sont pas souvent bien formé, nous avons utilisé le logiciel open source JTidy [JTi] (d'autres logiciels existent comme `HTMLCleaner` [Htm]) pour les transformer en documents `XML`. Par exemple, les pages simplifiées dans notre exemple ont subi de telles étapes de pré-traitement après leur récupération du site `http://upcoming.yahoo.com/`.

3.3 Définition des problèmes

Nous formalisons dans ce qui suit notre approche d'interrogation à deux étapes. Premièrement, l'utilisateur fournit une description (SOD) afin d'extraire des objets correspondant à cette dernière à partir de sources structurées. Deuxièmement, il interroge directement les données extraites pour récupérer les objets recherchés.

De manière générale, nous définissons le problème de recherche d'information structurée dans le Web comme suit. Étant donné un SOD s fourni par l'utilisateur et un ensemble large et varié (correspondant à différents domaines) de sources Web $\{S_1, \ldots, S_i\}$, `ObjectRunner` :

- **sélectionne** les meilleures sources pertinentes, qui correspondent au SOD fourni, ces sources peuvent ne pas être structurées et ni contenir les objets recherchés,
- **extrait** les instances du SOD à partir de la source choisie, parmi la liste des sources sélectionnées auparavant.

Une fois les objets extraits, l'utilisateur interroge le système pour récupérer l'objet recherché.

3.3.1 Le problème de la selection de sources

Un défi clé pour permettre l'extraction d'information ciblée fait l'objet d'une des deux contributions majeurs de cette thèse. Il consiste à identifier d'une manière complètement non-supervisée des sources (ensemble de pages) qui peuvent être utilisées pour l'inférence de wrappers, étant donné un schéma d'information en entrée (un SOD).

Nous avons identifier plusieurs critères qui devraient jouer un rôle important dans la sélection de sources structurées, et qui peuvent être prometteuses pour l'extraction. Plus précisément, une source Web est considérée pertinente, si elle vérifie les conditions suivantes :

1. elle contient des instances des types d'entités du SOD,
2. ces instances de types d'entités doivent apparaitre dans le même bloc de la page (généralement les entités regroupées dans le même bloc partagent le même contexte) à n'importe quel niveau dans la hiérarchie des blocs sur l'ensemble des pages de la source,
3. les instances de chaque type doivent apparaitre dans la même position dans l'arbre DOM (même *path*), vu que les données partagent la même structure et les mêmes schémas.

Dans notre système nous nous sommes limités à des SODs ayant des structures plates, mais il est possible d'étendre cette approche pour des structures imbriquées. Le problème de sélection de sources que nous considérons est comme suit :

- étant donné : (i) un seuil k, (ii) un répertoire de sources Web $\{S_1, \ldots, S_n\}$, se rapportant à divers domaines, et (iii) un SOD s, contenant un ensemble de types d'entités simples $t_1, \ldots, t_m$ correspondant à des concepts de l'ontologie,
- en se basant sur : (i) une base de connaissances sémantique organisée comme une hiérarchie de concepts et d'instances de ces concepts [1], et (ii) les caractéristiques visuelles des pages Web,
- trouver les k meilleures sources pertinentes pour l'extraction d'instances de s par l'inférence de wrappers.

Il est important que la sélection de sources se fasse à la volée *(online)* et de la manière la plus efficace possible. Cela signifie qu'une fois les sources collectées à partir du Web *(crawled)*, elles doivent être analysées et indexées – par rapport à la base de connaissances – afin de faciliter leur sélection. Nous voudrions que l'étape de sélection soit la plus rapide et la plus précise que possible. Pour cela, nous projetons d'effectuer tous les traitements

1. Dans nos expérimentations, nous avons utilisé l'ontologie FreeBase [BEP+08].

coûteux à l'avance *(offline)*.

Nous avons identifié les **sous-problèmes** suivants pour réduire les coûts de traitement et accroître la précision :

- *Construction de la structure hiérarchique d'une source.* Nous avons besoin de modéliser chaque source par une seule (unique) structure arborescente des blocs visuels contenus dans les pages de cette source, afin d'identifier le bloc qui contient les objets recherchés. Cette structure pourrait être inférée à partir d'un échantillon de pages et généralisée par la suite à toutes les pages de la source.
- *Détection du domaine de référence.* Comme les sources Web structurées sont généralement concentrées sur un domaine particulier, il est important de le découvrir avant l'annotation sémantique des pages Web, afin de simplifier cette étape d'annotation. À cet effet, un petit nombre de domaines (un ou deux en général) doivent être sélectionnés dans la base de connaissances comme étant les plus pertinents pour décrire la source.
- *Annotation et indexation adaptées.* Parmi les domaines détectés, cette étape consiste à trouver et annoter dans les pages les instances des concepts (classes) de l'ontologie. L'information dans les pages Web peut être localisée à divers niveaux de granularité : chaque source présente un ensemble de pages, et chaque page peut être vu comme une structure arborescente des blocs (voir la Figure 3.4). Détecter les instances des concepts de l'ontologie sur la représentation visuelle (structure) des pages semble être une solution intéressante, car cette structure suit souvent une logique liée à la sémantique des données présentées dans la page. L'indexation de sources a besoin de capter l'information sur : (i) les URIs des sources $\{S_1, \ldots, S_n\}$, (ii) la collection de pages $\{p_1^i, \ldots, p_j^i\}$ pour chaque S_i, et (iii) les identifiants de blocs $b_1^i, \ldots, b_k^i$ pour chaque source S_i. Ces différents éléments (sources, pages, blocs) seront associés aux instances de concepts de l'ontologie. Ainsi, les sources doivent être analysées et indexées d'une sorte à capter la structure arborescente (blocs visuels) et les annotations. Par exemple, la Figure 3.4 montre que le type t_1 apparait dans les triplés bloc-page-source $\{b_1^1, p_1^1, S_1\}$, $\{b_1^1, p_2^1, S_1\}$, $\{b_1^3, p_1^3, S_3\}$, etc.
- *Sélection flexible guidée par l'ontologie.* La représentation des sources annotées dans une structure arborescente devrait être exploitée de façon flexible, c-à-d, l'interrogation de sources doit être faite de façon souple et flexible lorsqu'il n'y a pas de correspondance directe entre les annotations dans les pages et les types spécifiés par l'utilisateur. Une recherche dans les concepts voisins peut être efficace.

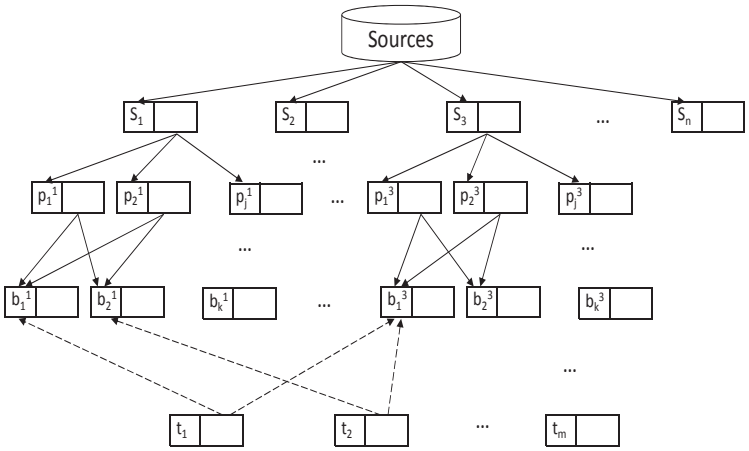


FIGURE 3.4 – Divers niveaux de granularité dans les sources Web

3.3.2 Le problème d'extraction d'objets

Le problème d'extraction d'objets dans le Web structuré peut être formalisé comme suit. Étant donné un SOD s et une source S_i, un *template* τ vis-à-vis de s et S_i décrit comment des instances de s peuvent être extraites à partir des pages de S_i. Plus précisément,

- pour chaque `type set` $t = [\{t_i\}, m_i]$ apparaissant dans s, τ définit un *séparateur (chaine de caractères) sep*^{t} ; il indique que les instances consécutives de t_i seront séparées par cette chaine.
- pour chaque `type tuple` $t = [t_1, \ldots, t_k]$, τ définit un ordre total sur la collection de types et une séquence de $k+1$ *separateur* de chaines de caractères $sep_1^t, \ldots, sep_{k+1}^t$; ce qui indique que les k instances des k types formant t, dans l'ordre spécifié, seront délimitées par ces séparateurs.

Le problème d'extraction que nous considérons est comme suit. Pour un SOD s et un ensemble de sources $\{S_1, \ldots, S_n\}$ sélectionnées dans l'étape précédente :

1. **relier** les reconnaisseurs de types aux types d'entités dans s,
2. pour chaque source S_i,
   (a) **trouver** et **annoter** les instances de chaque type d'entité dans les pages Web,
   (b) **sélectionner** un échantillon de pages sur la base du résultat des annotations,
   (c) **déduire** le template $\tau_i(s, S_i)$ en se basant sur l'échantillon,
   (d) utiliser τ_i pour **extraire** toutes les instances de s à partir de S_i,

La Figure 3.5 montre le résultat souhaité pour l'extraction des objets `concerts` à partir des pages de l'exemple de la Figure 3.1, en supposant que le SOD fourni est composé de trois types d'entités simples : artiste, date et adresse. Nous observons que le template obtenu contient les types d'entités recherchées (*artiste*, *date* et *adresse*) et les valeurs de ces entités sont extraites séparément pour les quatre objets (x_1, x_2, x_3 et x_4) découverts dans les pages (p_1, p_2 et p_3).

	Artist	Date	Address
X_1	Metallica	Monday May 11, 8:00pm	Madison Square Garden 237 West 42nd street New York City New York 10036
X_2	Coldplay	Saturday August 8, 2010 8:00pm	Bowery Ballroom Delancey St New York City New York 10002
X_3	Madonna	Saturday May 29 7:00p	The Town Hall 131 W 55th St New York City New York 10019
X_4	Muse	Friday June 19 7:00p	B.B King Blues Club and Grill 4 Penn Plaza New York City New York 10001

```
<html><body>
  {<
    <li>
    <div type="Artist"> * </div>
    <div type="Date"> * </div>
    <div type="Address">
      <span><a> * </a></span>
      <span> * </span>
      <span> * </span>
      <span> * </span>
      <span> * </span>
    </div>
    </li>
  >}
</body></html>
```

FIGURE 3.5 – La solution correcte (objets et wrapper) sur l'exemple 3.1

3.4 Les avantages d'une interrogation à deux étapes

Nous avons vu dans le Chapitre 2 que les approches antérieures d'extraction non-supervisées présentent plusieurs limites, vu qu'elles exploitent uniquement les régularités dans les pages pour récupérer les données. Nous pensons que l'approche alternative d'une interrogation à deux étapes peut être plus appropriée pour des scénarios du monde réel, offrant plusieurs avantages tels que :

- *Éviter de mélanger différent types d'information.* En s'appuyant sur les reconnaisseurs de types pour annoter les pages Web en entrée, en exploitant des informations sémantiques en plus des caractéristiques structurelles des données, nous pouvons améliorer le processus d'extraction.
- *Extraire uniquement les données utiles.* La description des objets ciblés nous permet d'éviter l'extraction de données inutiles, aussi que d'éventuels traitements après l'extraction (filtrage/annotation de données).
- *Arrêt anticipé du processus d'extraction.* Durant le processus de construction du

template, si la collection de pages n'est plus considérée comme pertinente pour l'extraction, l'inférence du wrapper peut être arrêtée.

- *Activer un processus d'auto-validation.* En utilisant des annotations sémantiques sur le contenu des pages, nous pouvons estimer automatiquement la qualité de l'extraction (et du wrapper correspondant) en vérifiant si d'éventuelles annotations contradictoires apparaissent sur des valeurs d'un même attribut.
- *Éviter la perte d'informations utiles.* Certaines données qui devraient être sélectionnées dans les résultats des objets extraits peuvent paraitre "trop régulières" par le système, c-à-d, comme faisant partie du template de la page. Ceci se produit lorsque les techniques utilisées ne prennent pas en considération la sémantique des données. Dans ce cas, des données utiles feront partie du template et ne seront pas extraites. Par exemple, dans l'échantillon de page de la Figure 3.1, le texte "New York" apparait souvent et dans la même position dans les pages, simplement du fait que de nombreux concerts ont lieu dans cette ville. Si le texte est reconnu comme une adresse, faisant partie de la composition des objets concerts, il sera reconnu comme une donnée à extraite.
- *Utiliser les annotations sémantiques dans les pages pour découvrir de nouvelles annotations potentielles.* Inévitablement, l'utilisation de dictionnaires (gazetteers) pour annoter les données produit souvent des annotations incomplètes. Cependant, nous pouvons utiliser les annotations existantes sur les pages pour en déduire d'autres. Ceci peut se faire par l'enrichissement de dictionnaires après chaque extraction, en se basant sur les données extraites. On peut aussi déduire des mesures de confiance à associer aux annotations, à partir de la qualité de résultats obtenu par le wrapper.

3.5 Conclusion

Dans ce chapitre, nous avons présenté les pré-requis sur lesquels s'appuient les différents concepts que nous avons utilisé pour décrire le modèle proposé. Nous avons ensuite précisé les problématiques traitées dans cette thèse et les solutions possibles au vu de techniques (existantes) proposées dans la littérature.

Nous détaillons dans les chapitres suivants les deux principales contributions abordées de cette thèse. En premier lieu, l'approche proposée pour l'extraction d'objets structurés sera présentée dans le chapitre 4. En second lieu, l'approche proposée pour la sélection de sources pertinentes sera présentée dans le chapitre 5.

Chapitre 4

Extraction d'Information Ciblée

Dans ce chapitre, nous détaillons le processus d'extraction de notre système `ObjectRunner`. Nous décrivons les tâches principales de notre approche d'extraction, en termes de modules qui la composent et des choix d'implantation. Nous illustrons également son fonctionnement à travers un exemple d'application.

Le processus d'extraction est effectué en deux étapes : (1) l'annotation automatique, qui consiste à reconnaitre dans le contenu de la page les instances des types d'entités qui composent le SOD spécifié par l'utilisateur, et (2) la construction du template d'extraction, en utilisant les annotations sémantiques de la phase précédente et la régularité des pages.

4.1 Architecture générale

Une vue globale du système `ObjectRunner` est donnée dans la Figure 4.1. Le système est composé d'une interface d'interrogation qui permet aux utilisateurs de spécifier par l'intermédiaire des SODs ce qui doit être extrait à partir de pages Web. En particulier, les types atomiques (i.e., entités simples), qui composent les objets recherchés et leur structuration (par exemple, les contraintes d'occurrences, imbrication, valeurs de jointures).

À partir d'un corpus de sources Web, où chaque source est composée d'un ensemble de pages partageant un schema commun et une structure commune (implicite). `ObjectRunner` construit un template d'extraction (wrapper) et récupère les objets – données structurées correspondant aux instances possibles du SOD spécifié – à partir de ces pages. Les données extraites et les informations textuelles liées à ces données pourraient être en suite indexées et stockées dans le système.

Après la phase d'extraction, les utilisateurs peuvent interroger le système, en sélectionnant un ou plusieurs SODs et des sources d'intérêt. Ceci se fait par l'intermédiaire d'une interface d'interrogation du type QBE (Query By Example) dans laquelle l'interrogation

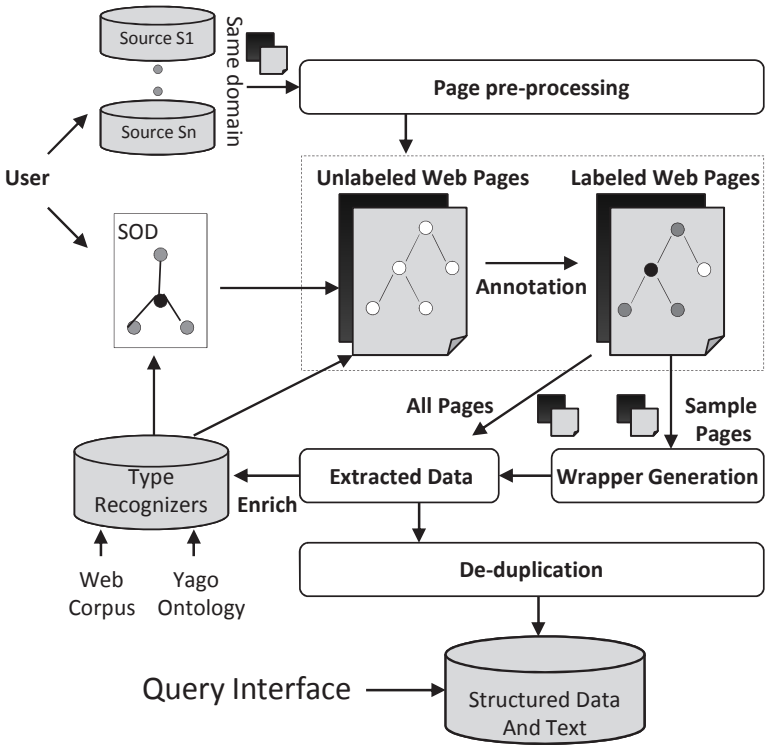


FIGURE 4.1 – Architecture du système ObjectRunner

par mots-clés est également possible. Les résultats de la requête sont ordonnés en se basant sur les scores de confiance calculés dans la phase d'extraction.

4.2 Algorithme d'extraction

La phase d'extraction constitue la partie la plus importante d'ObjectRunner. Dans cette section nous détaillerons les axes principaux du système.

4.2.1 Reconnaisseurs de types

Dans cette section, nous décrivons seulement comment les types *IsInstanceOf* (les types basés sur les dictionnaires et les ontologies que nous utilisons) sont gérés, puisque les types prédéfinis (Date, Adresse, etc.) et les expressions régulières sont relativement faciles à détecter dans les pages.

Le problème de la reconnaissance des types $isInstanceOf$ consiste à associer à chaque type atomique t du SOD s ($t \in s$) un dictionnaire (ensemble d'instances possibles). Ces dictionnaires sont déterminés en utilisant des ontologies existantes ou en s'appuyant sur des techniques d'instanciation de classes à partir du contenu du Web. Dans notre système, nous avons utilisé l'ontologie YAGO et les patterns de Hearst.

Intuitivement, pour alléger la tâche de spécification pour l'utilisateur, ce sont des types pour lesquels seulement un *nom de classe* est fourni. Ceci pourrait être le cas pour le type d'entité `Artist`. Quand un tel type est fourni par l'utilisateur, le système construit et associe automatiquement un dictionnaire pour le reconnaitre.

Des sources d'information riches (des ontologies basées sur Wikipedia ou de grands ensembles de données textuelles basées sur le Web tel que ClueWeb [1]) sont disponibles et peuvent être facilement exploitées pour la construction de dictionnaires associés aux types *isInstanceOf*. Cette solution s'est avérée simple et performante, nous décrivons dans la suite comment ces dictionnaires sont obtenus.

4.2.1.1 Ontologie YAGO

L'ontology YAGO [SKW07], est une base de connaissances large construite à partir de Wikipedia et WordNet. Actuellement, YAGO2 [HSB+11], extension de YAGO, contient plus de 10 million d'entités (des noms de personnes, des lieux, etc.) et 80 million de faits (relations isA, $wasBornIn$, etc.). Les instances dans YAGO sont associées à des valeurs de confiance. Ces valeurs sont calculées par des estimations de probabilités que les relations associées aux instances soient correctes dans Wikipedia.

Nous avons exploité les relations isA dans YAGO, pour associer un dictionnaire à tout type t spécifié par l'utilisateur. Plus précisément, si le type *Artist* fait partie de la composition du SOD fourni, le système récupère toutes les instances de la classe (*Artist*) en utilisant la relation isA de YAGO. Néanmoins, des instances pertinentes peuvent ne pas être trouvées en exploitant uniquement les relations isA associées au type fourni par l'utilisateur. Par exemple, le groupe `Metallica` n'est pas une instance de la classe `Artist`. Dans ce genre de situation, nous regardons s'il existe dans l'ontologie une classe ayant une sémantique voisine de celle du type fourni : ex, le groupe `Metallica` est une instance de la classe `Band`, qui est sémantiquement proche de la classe `Artist`. Nous avons donc adapté YAGO à nos objectifs afin d'accéder à ce genre de données.

4.2.1.2 Patterns de Hearst

En complément à YAGO, c-à-d, rechercher des instances d'un type d'entité donné, nous avons appliqué les patterns de Hearst [Hea92] (définis dans le Tableau 4.1) sur un corpus de pages Web qui est pré-traité à cet effet. Les patterns de Hearst sont de simples patterns textuels, tels que *Artist such as X*, ou *X is an Artist*, par lesquels on veut trouver les valeurs du paramètre X dans le texte.

Cette technique a permis d'associer des instances aux types, par exemple pour ceux qui

1. http ://www.lemurproject.org/clueweb09.php.

Patterns
concept such as GP | $((GP, ?)^+$ and $GP)$
concept,? including GP | $((GP, ?)^+$ and $GP)$
concept,? especially GP | $((GP, ?)^+$ and $GP)$
GP is a *concept* | $((concept, ?)^+$ and *concept*)
GP and other *concept* | $((concept, ?)^+$ and *concept*)
GP or other *concept* | $((concept, ?)^+$ and *concept*)

TABLE 4.1 – Patterns de Hearst (GP représente le groupe nominal)

ne sont pas disponibles dans YAGO. Pour parvenir à attribuer des valeurs de confiance aux instances candidates, nous nous somme appuyés sur la métrique proposée par McDowell et al. [MC08a], qui calcule le score de confiance pour chaque paire (i, t), comme suit :

$$score(i,t) = \frac{\sum_{p \in P} count(i,t,p)}{max(count(i), count_{25}) \times count(t)} \tag{4.1}$$

Dans cette formule, $count(i, t, p)$ est le nombre de résultats pour la paire (i, t) dans le corpus par le pattern p, $count(i)$ est le nombre de résultats pour le terme i dans le corpus, et $count_{25}$ est le résultat obtenu au pourcentage 25. Intuitivement, si une paire (i, t) est extraite plusieurs fois dans le corpus, cette redondance renforce le fait que la paire est correcte (son score de confiance sera de ce fait élevé).

4.2.2 Annotation de pages Web

Le problème de reconnaissance d'entités consiste à correspondre un ensemble de types d'entités $t_1, \ldots, t_k$ (en général petit) avec une collection de pages Web (que nous appelons une source).

L'annotation est souvent réalisée en associant un attribut au nœud contenant l'instance correspondant au type donné, dans l'arbre DOM représentant la page traitée. Plusieurs annotations peuvent être attribuées à un même nœud. Le résultat final est un arbre `DOM` annoté avec les différents types. Par exemple, dans la page p_1 de notre exemple de la Figure 3.1, nous obtiendrons les annotations montrées dans la Figure 4.2. La première balise $<$ `div` $>_1$ contient un nom d'artiste, ainsi elle est annotée comme suit : `<div type= "Artist"> Metallica </div>`. Les annotations sont propagées dans l'arbre DOM aux parents lorsque ceux-ci n'ont qu'un seul nœud fils (c-à-d, une hiérarchie linéaire sans bifurcation *(linear path)*). Elles sont également propagées aux nœuds parents lorsque tous les fils partagent la même annotation. Par exemple, la balise $<$ `div` $>_3$ hérite l'annotation

`Address` vu que tous les nœuds fils ont la même annotation. Intuitivement, ces deux choix renforcent l'annotation, en l'associant à une partie plus importante d'une page.

```
<html><body>
<li>
  <div type="Artist">1 Metallica </div>
  <div type="Date">2 Monday May 11, 8:00pm </div>
  <div type="Address">3
        <span><a> Madison Square Garden</a></span>
        <span>237 West 42nd street </span>
        <span>New York  City</span>
        <span>New York</span>
        <span>10036</span>
  </div>
</li>
</body></html>
```

FIGURE 4.2 – Page p_1 après l'annotation

Dans notre système nous avons choisi de ne considérer que les pages contenant suffisamment d'annotations. Pour une source donnée, étant donné plusieurs dictionnaires en entrée, chacun avec des scores de confiance associés. Les pages contenant le plus grand nombre d'annotations sont sélectionnées et utilisées comme échantillon pour la construction du template d'extraction.

Si cette étape d'annotation est réalisée de façon exhaustive, ça pourrait rendre l'extraction plus coûteuse, puisque nous utilisons une base de données d'instances de types d'entités, qui peut être grande. Dans notre approche, seulement un sous ensemble de pages est annoté, et à partir de ces pages, seulement quelques unes (approximativement 20 pages) seront utilisées pour la construction du template. Notez que nous sélectionnons d'abord les pages pour les types $isInstanceOf$ (un type à la fois), puis les types prédéfinis et les expressions régulières sont traités à leur tour.

Sélectivité des types

Pour accélérer le processus d'annotation, nous utilisons *des estimations de sélectivité*, à la fois au niveau des types et des instances de types. La sélectivité d'un type t donne une indication sur le nombre de fois où il pourrait apparaitre dans l'échantillon de pages. Plus un type est sélectif et moins il risque d'apparaitre dans les pages. L'Algorithme 3 décrit le processus d'annotation.

Dans cette étape, les pages les plus annotées sont sélectionnées pour l'inférence du wrapper. Nous associons pour chaque type *isInstanceOf* t une estimation de sélectivité,

Algorithm 3 Annotation de pages

1: **input :** taille de l'échantillon résultat k, source S_i, SOD s
2: échantillon $S := S_i$
3: ordonner les types d'entités dans s par ordre décroissant de sélectivité (Équation 4.2)
4: **for each** type t dans s **do**
5: chercher les correspondances de t dans S et annoter
6: calculer le score pour chaque page (Équation 4.3)
7: pour $S' \sqsubseteq S$ top pages annotées, faire $S := S'$
8: **end for**
9: **return** l'échantillon de k pages les plus annotées dans S

calculée comme suit :

$$score(t) = \sum_i score(i,t)/tf(i), \tag{4.2}$$

où i désigne chaque instance dans le dictionnaire associée au type t ($i\ isInstanceOf\ t$), décrite par son score de confiance $score(i,t)$ et la fréquence du terme $tf(i)$ dans le corpus Web ou dans l'ontologie utilisée, en fonction du choix de l'utilisateur.

Avant d'annoter, les types du SOD sont ordonnés (ligne 3), pour que l'annotation commence par les types les plus sélectifs. À chaque itération (lignes $4-8$), les pages sont annotées pour un type t, puis l'échantillon qui servira à l'étape suivante est sélectionné. Cette sélection s'appuie sur le nombre d'annotations contenues dans chaque page. Un score indiquant le degré d'annotation est calculé pour chaque page selon l'équation suivante :

$$score(page/t) = \sum_{i' \in t} score(i',t)/tf(i') \tag{4.3}$$

où i' désigne chaque instance de t dans la page et $tf(i')$ est la fréquence de i' dans cette page.

Seules les pages les plus annotées ("les plus riches") sont retenues après chaque étape d'annotation. Pour cela, nous ordonnons les pages par leur score minimum par rapport aux types qui ont déjà été traités, comme suit :

$$min(score(page/t_1), \ldots, score(page/t_n))$$

Ce processus de sélection, nous permet de réduire le nombre de pages qui doivent être annotées aux étapes suivantes. D'ailleurs, une source peut être éliminée pendant l'annotation si les niveaux d'annotations obtenus sont insuffisants (les détails sont donnés dans la Section 4.2.5).

4.2.3 Génération de wrappers

Ce module est le noyau de notre algorithme d'extraction ciblée. Pour chaque source S_i, il retourne un template d'extraction τ_i correspondant au SOD s fourni. Nous adaptons dans `ObjectRunner` une approche qui est similaire dans le style à l'algorithme `ExAlg` [AGM03] (voir la Section 2.1.5.3 pour plus de détails sur `ExAlg`).

Le template est inféré en identifiant les parties statiques, à partir d'un échantillon de pages de la source traitée. Ces parties statiques sont considérées comme faisant partie du template. L'algorithme d'extraction considère les pages comme étant un ensemble de `tokens`, texte ou balise HTML, et calcule leurs fréquences d'apparition dans ces pages. Ces fréquences servent à calculer des *vecteurs d'occurrence* de tokens - des vecteurs contenant le nombre d'occurrences d'un token - qui sont utilisés par la suite pour créer des *classes d'équivalence* définies par ces tokens. Dans notre exemple d'application (Figure 3.1), le token `<div>` a trois occurrences dans les deux premières pages et six occurrences dans la troisième page, ce qui nous donne le vecteur d'occurrence suivant : $< 3, 3, 6 >$.

Une *classe d'équivalence* EQ (Equivalence Class), désigne un ensemble de tokens partageant le même *vecteur d'occurrence* dans les pages sélectionnées. Pour que les EQs soient `valides`, elles doivent être *ordonnées*, c-à-d, leurs tokens sont ordonnés selon l'ordre d'apparition dans les pages. De plus, les EQs doivent être mutuellement *imbriquées* ou ne se chevauchant pas. Seules les EQs valides seront utilisées pour la construction du template.

Les EQs servent à déterminer le template d'extraction inféré à partir des pages Web. L'identification et la distinction des différents rôles des tokens est cruciale pour l'inférence implicite du schéma. Cette inférence de rôles dépend de deux critères : (i) la position dans l'arbre DOM de la page, et (ii) la position par rapport à chaque EQ trouvée dans l'itération précédente (les EQs en cours). Nous ajoutons dans `ObjectRunner` un troisième critère : les annotations qui permettent d'affiner les rôles et d'obtenir des résultats d'extraction plus précis.

Les itérations consécutives raffinent les EQs jusqu'à ce qu'un point fixe soit atteint. Dans chaque étape, les EQs non valides sont éliminées, si elles ne sont pas correctement ordonnées et imbriquées.

4.2.3.1 Identification des rôles de tokens

L'algorithme 4 montre comment les rôles des tokens sont différenciés. Premièrement, les rôles des tokens sont déterminés en utilisant la structure `HTML` de la page (ligne *1*) : les tokens ayant la même valeur et la même position (i.e., même chemin (path) dans l'arbre DOM), auront le même rôle. Ensuite, les rôles des tokens sont raffinés en boucle,

en se basant sur leurs positions d'apparition dans les classes d'équivalence EQs (lignes *3-10*). Durant cette phase, toutes les EQs non valides sont éliminées et le processus peut être arrêté si certaines conditions ne sont pas vérifiées (nous allons les détailler dans la Section 4.2.5).

Algorithm 4 Différencier les rôles de tokens

```
1: différencier les rôles en utilisant la position des tokens dans l'arbre DOM de la page
2: repeat
3:    repeat
4:       recherche des classes d'équivalence (EQs)
5:       gestion des classes non valides EQs
6:       if les conditions d'arrêt sont vérifiées then
7:          arrêter le processus
8:       end if
9:       différencier les rôles en utilisant les EQs + les annotations non conflictuelles
10:   until fixpoint
11:   différencier les rôles en utilisant les EQs + les annotations conflictuelles
12: until fixpoint
```

Le calcul des rôles des tokens et des classes d'équivalence se fait dans notre approche de façon plus précise de celle proposée dans [AGM03]. Premièrement, nous utilisons les annotations comme critère complémentaire pour distinguer de nouveaux rôles de tokens. Deuxièmement, le SOD accomplit un double rôle pendant l'étape de génération du wrapper, car il nous permet de : (i) arrêter le processus aussitôt que nous pouvons conclure que le SOD ciblé ne peut pas être satisfait, et (ii) accepter des classes d'équivalence approximatives en plus de celles qui pourraient représenter les instances à extraire (i.e., accepter quelques classes non valides si elles contiennent les annotations recherchées).

4.2.3.2 Utilisation des annotations

Dans notre approche, nous utilisons les annotations comme critère additionnel pour la découverte de nouveaux rôles des tokens. Cependant, ces annotations ne sont pas toujours complètes (les dictionnaires ne couvrent pas toutes les instances contenues dans les pages). Notamment, les annotations peuvent être conflictuelles sur l'ensemble des pages (une instance peut être associée à différents types sur différentes pages).

Nous avons distingué deux types d'annotations :

– **Les annotations non conflictuelles.** Un token, identifié par sa position dans l'arbre DOM d'une page et ses coordonnés (positions) dans les classes d'équivalence existantes, a des annotations non conflictuelles si toutes ses occurrences ont le

même type d'annotation (unique) ou n'en ont aucune.
- **Les annotations conflictuelles.** Un token a des annotations conflictuelles si différents types d'annotations ont été attribués à ses occurrences.

Les annotations sont appliquées rigoureusement dans la différenciation des rôles des tokens. En premier lieu, les tokens sans annotations conflictuelles sont traités dans la boucle interne (ligne *9* de l'algorithme 4) avec les autres critères (utilisation des positions dans les EQs). Une fois que toutes les EQs sont calculées de cette manière, nous procédons à une itération supplémentaire (lignes *2-12*) pour différencier les rôles des tokens pour lesquels nous avons des annotations conflictuelles (ligne *11*). Cela va générer de nouveaux vecteurs d'occurrence et éventuellement de nouvelles classes d'équivalence EQs. L'ensemble du processus est répété jusqu'à ce qu'un point fixe soit atteint.

Exemple d'application. Revenons à notre exemple (Figure 3.1). Si les annotations sont prises en compte, on peut détecter que les occurrences de la balise `<div>`, notées $<$ `div` $>_1$, $<$ `div` $>_2$ et $<$ `div` $>_3$ ont des rôles différents (voir la Figure 4.2). Ainsi, nous pouvons déterminer correctement comment extraire ces trois composantes. Ceci ne serait pas possible si seulement les positions dans l'arbre `HTML` et dans les EQs sont prises en compte, car les trois occurrences de `<div>` auraient dans ce cas le même rôle. Cette façon de faire mènerait, dans le cas de la page p_1, au résultat d'extraction suivant : (1) Metallica Monday May 11, 8 :00pm, (2) Madison Square Garden, (3) 237 West 42nd street, and (4) 10036.
Or cee résultat n'est pas satisfaisant, puisque le nom de l'artiste et la date ne sont pas séparés et sont extraits dans le même attribut (champs) du résultat (d'un enregistrement extrait). Le même résultat d'extraction sera obtenu pour les pages p_2 et p_3.

Exemple illustrant l'importance des annotations. Pour mettre en évidence encore plus l'intérêt des annotations dans l'inférence du wrapper, considérons l'exemple de la Figure 2.3. Le fragment de la page liste représenté contient trois livres. Un livre peut avoir un ou plusieurs auteurs, qui peuvent être représentés différemment dans le code `HTML`. Nous détaillons ci-dessous le code `HTML` correspondant aux auteurs de ces livres :

livre 1 : `<span>`by `<a>`Jane Austen`</a>` and Fiona Stafford`</span>`
livre 2 : `<span>`by Hamilton Wright Mabie, Mary Hamilton Frey`</span>`
livre 3 : `<span>`by `<a>`Abraham Verghese`</a><span>`

Le fait de ne prendre en compte que les positions des tokens dans les EQs peut nous mener à considérer les noms d'auteurs comme étant des valeurs de champs distincts. Dans

notre système, comme les annotations sont propagées vers les nœuds parents dans l'arbre DOM (voir Section 5.3), le nœud `<span>` héritera donc d'une annotation qui indique que ce champ représente des noms d'auteurs et son contenu sera extrait correctement.

Quelques choix considérés durant l'utilisation des annotations

Pendant la phase d'utilisation des annotations pour différencier les rôles des tokens, nous avons considéré certains choix pour mieux adapter notre approche aux différentes situations qui peuvent se présenter en pratique. Parmi ces choix, nous citons :

- **Fixer le nombre d'occurrences consécutives d'un token.** Le nombre d'occurrences consécutives d'un token peut dans certains cas varier d'une page à une autre et éventuellement au sein de la même page (dans les enregistrements des pages listes). Ceci n'est pas le cas dans notre exemple d'application (Figure 3.1), où le token `<div>` a le même nombre d'occurrences dans tous les enregistrements des pages (3 occurrences). Dans le cas où ce nombre est variable, nous avons choisi de différencier les rôles dans un périmètre fixé en fonction du nombre minimum d'occurrences consécutives trouvées sur toutes les pages. Par exemple, si le token `<div>` avait le vecteur d'occurrence suivant : $< 2, 3, 6 >$, la différenciation des rôles se ferait seulement pour les deux premières occurrences de chaque enregistrement.
- **Généralisation des annotations incomplètes.** Aucune hypothèse n'est faite durant la phase d'annotation sur la complétude des dictionnaires utilisés. Ces dictionnaires peuvent bien évidemment ne pas couvrir toutes les instances existantes dans les pages. Dans notre approche, le traitement des annotations incomplètes se fait en généralisant les plus fréquentes au-delà d'un seuil donné (0.7 dans nos expérimentations). Notez que, dans le cas où les occurrences consécutives d'un token sont variables, la généralisation se fait dans le périmètre fixé (nombre minimum d'occurrences).

4.2.4 Construction du template

Dans notre approche, la construction du template se fait en se basant sur : (1) la structure du schéma spécifié en entrée (SOD), et (2) la hiérarchie des classes d'équivalences valides $\{EQ_1, \ldots, EQ_n\}$, découvertes durant la construction du wrapper (Algorithme 4).

Comme expliqué dans la Section 3.3.2, le problème d'extraction d'objets dans le Web structuré peut être formalisé comme suit. Étant donné un SOD s et une source S_i, un *template* τ vis-à-vis de s et S_i décrit comment des instances de s peuvent être extraites à partir des pages de S_i. Plus précisément,

- pour chaque `type set` $t = [\{t_i\}, m_i]$ apparaissant dans s, τ définit un *séparateur (chaine de caractères)* sep^t ; il indique que les instances consécutives de t_i seront séparées par cette chaine.
- pour chaque `type tuple` $t = [t_1, \ldots, t_k]$, τ définit un ordre total sur la collection de types et une séquence de $k+1$ *séparateurs* (chaines de caractères) $sep^t_1, \ldots, sep^t_{k+1}$; ce qui indique que les k instances des k types formant t, dans l'ordre spécifié, seront délimitées par ces séparateurs.

Les tokens contenus dans les EQs forment les séparateurs du template recherché τ.

Rappelons qu'une classe d'équivalence valide EQ est ordonnée et deux EQs quelconques sont soit imbriquées, soit ne se chevauchant pas. Cette hiérarchie des EQs est maintenue au niveau du template τ – i.e., les séparateurs dans les EQs – et sera appliquée au template τ construit en sortie. Ainsi, τ suivra une structure arborescente semblable à celle des EQs et les annotations des types existantes dans les EQs vont permettre d'indiquer où se trouve l'information à extraire. Nous appelons cette structure *l'arbre annoté du template*.

La Figure 4.4 représente un arbre du template qui devrait correspondre au SOD s donné dans la partie gauche de la Figure 4.3. Ce SOD se compose de : (i) deux `types tuples` $[t_1, \{t_2\}, t_3]$ et $[t_{31}, t_{32}]$, et (ii) un `type set` $\{t_2\}$ ayant des contraintes de multiplicité arbitraires. La racine de l'arbre du template (Figure 4.4) couvre une page entière, le premier niveau de l'arbre correspond à une classe d'équivalence EQ_i, décrite par des séparateurs et des annotations pour les types atomiques t_1, t_{31}, et t_{32}. Le sous-arbre du deuxième niveau correspond à la classe EQ_j, annotée par le type atomique t_2. Chaque niveau de l'arbre du template représente une séquence de tokens d'une EQ. L'imbrication est représentée par les `types set`.

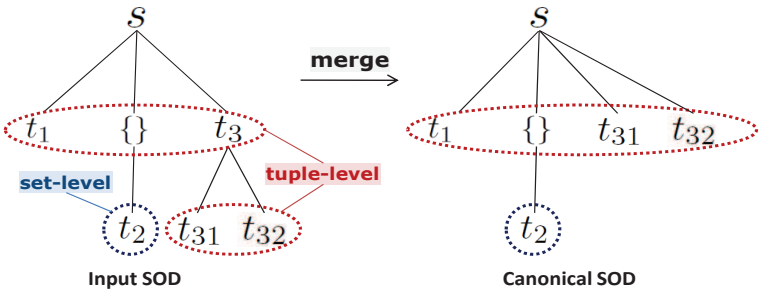


FIGURE 4.3 – Correspondance ascendante (bottom-up) du SOD.

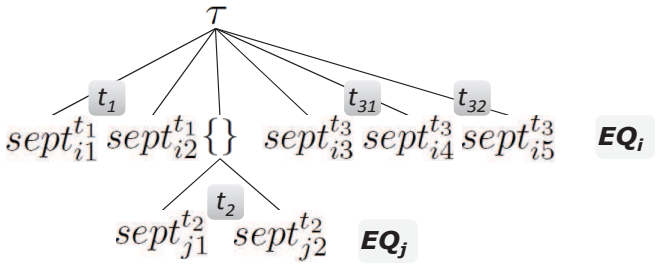


FIGURE 4.4 – Arbre annoté du template.

Correspondance entre le SOD et le template

Cette étape permet de déterminer, pour un SOD donné, les parties des pages concernées par l'extraction. Elle consiste à identifier dans l'arbre du template les régions (sous-arbres) qui correspondent au SOD ciblé. Seules ces régions seront extraites. Pour cela, une étape préliminaire transforme le SOD en entrée dans une forme simplifiée (canonique), qui désimbrique les tuples et les regroupe. Rappelons qu'un SOD peut être formé par des `types tuple` et des `types set`, et que les feuilles représentent des `types atomiques`.

Afin de mettre le SOD dans sa forme canonique, chaque nœud tuple recevra comme fils directs tous les nœuds de types atomiques qui lui sont attachés uniquement par des nœuds fils de type tuple (aucun fils de type set). La deuxième partie de la Figure 4.3 montre cette mise sous la forme canonique, des tuples : $[t_{31}, t_{32}]$ sera combiné avec le tuple $[t_1, \{t_2\}, t_3]$ et remplacé par un nouveau tuple $[t_1, \{t_2\}, t_{31}, t_{32}]$.

Une fois le SOD transformé dans la forme canonique, nous procédons à une correspondance ascendante (bottom-up) entre *le SOD canonique et l'arbre annoté du template*, en commençant par les EQs qui se retrouvent au niveau feuille et dont les tokens portent des annotations correspondant à des types d'entités qui apparaissent au niveau inférieur du SOD.

Le processus commence par les nœuds parents directs des feuilles du SOD (niveau feuille - 1), qui peuvent être soit de *type tuple* ou de *type set*.

Dans le cas le plus simple, si le SOD a *une structure plate* (un seul tuple), son arbre aura deux niveaux : un nœud racine de type tuple et des nœuds feuilles de types entités simples (types atomiques). Ces types d'entités simples sont associés aux séparateurs de types qui composent les classes d'équivalence suivant deux critères : (i) les séparateurs doivent appartenir à la même classe d'équivalence, et (ii) ils doivent avoir des annotations qui correspondent aux types considérés.

Pour l'exemple du SOD de la Figure 4.3, nous devons trouver une classe d'équivalence Q_j au niveau feuille de la hiérarchie contenant des séparateurs pour le `type set` $\{t_2\}$, et

cette classe d'équivalence doit avoir une classe d'équivalence parente Q_i contenant des séparateurs pour les types $\{t_1, t_{31}, t_{32}\}$.

4.2.5 Arrêt du processus de génération du wrapper

Avoir une description intentionnelle des données à extraire nous permet également d'arrêter le processus de génération du wrapper, lorsqu'on peut juger qu'il ne rapportera pas de résultats satisfaisants. Nous décrirons dans cette section comment cette tâche est réalisée dans notre système. Nous pouvons distinguer deux phases où le processus de génération peut être arrêté.

4.2.5.1 Pendant la phase d'annotation

Dans notre système, lorsque le niveau d'annotations obtenu pour une source est faible, elle est éliminée du jeu de données. La condition d'arrêt repose sur un seuil et s'applique aux niveaux de granularité les plus bas des blocs visuels composant les pages.

Les différents blocs représentent habituellement différents contextes dans la page, il est donc judicieux de distinguer les annotations par bloc. Pour chaque bloc, nous nous assurons que la condition suivante est vérifiée :

$$\sum_{i=1}^{k} \frac{no.\ of\ annotations\ in\ block}{k} > \alpha$$

où k est la taille de l'échantillon (nombre de pages) et α est le seuil (50% dans nos expérimentations).

Après chaque annotation pour un type $t \in s$, si nous obtenons au moins un bloc qui satisfait le seuil d'annotation, nous continuons le processus d'annotation pour les types restants. Autrement le processus est arrêté.

4.2.5.2 Pendant la phase de génération du wrapper

Rappelons que l'inférence du wrapper est principalement basée sur la découverte de classes d'équivalence et sur leur correspondance avec le SOD spécifié. Nous avons vu dans la Section 4.2.3 que le processus de génération du wrapper est itératif et s'appuie sur la découverte des EQs, en traitant les EQs non valides (non ordonnées et non imbriquées) et en différenciant les rôles des tokens. À chaque itération (voir Algorithm 4), le processus peut être arrêté si nous trouvons que les classes d'équivalence obtenues ne peuvent pas aboutir à une hiérarchie de classes avec laquelle il peut y avoir une correspondance avec le SOD.

Après l'étape de construction des classes d'équivalence et de traitement de celles qui sont non valides (lignes *4-5* de l'algo 4), nous utilisons l'arbre du template correspondant

à la hiérarchie de classes en cours (ces classes ne sont pas nécessairement finales). Nous cherchons à déterminer s'il existe une correspondance partielle entre cet arbre et le SOD. Si aucune correspondance n'est trouvée le processus est arrêté.

La correspondance avec le SOD est réalisée comme décrit dans la Section 4.2.4. En cas de correspondance partielle, seule une partie du SOD correspond à une partie du template. Pour les parties manquantes, il peut rester des tokens annotés qui ne sont pas encore traités, et qui peuvent contenir des types atomiques qui mèneraient plus tard à une correspondance complète. Ceci est vérifié de manière incrémentale, en gardant la trace des correspondances partielles obtenues et leurs évolution progressive à une correspondance complète.

4.2.6 Enrichissement des dictionnaires

La découverte de nouvelles instances à partir de pages Web pendant la phase d'extraction, permet aussi d'enrichir les dictionnaires. Nous associons aussi aux nouvelles instances un score de confiance avant de les ajouter dans les dictionnaires. Ces scores sont calculés selon la formule suivante :

$$score(i) = f(wrapper\_score(t), \frac{\sum_{D \cap I} score(i,t)}{\mid I \mid}) \tag{4.4}$$

où i désigne l'instance découverte dans la page Web, D désigne l'ensemble d'instances existantes dans les dictionnaires, I désigne l'ensemble d'instances extraites de la source considérée, et $wrapper\_score(t)$ est un poids supplémentaire attribué au wrapper. Ce poids varie en fonction du nombre d'annotations conflictuelles constatées lors de la construction du template. Intuitivement, cette formule donne plus de poids aux instances obtenues par un wrapper correct construit sans ou avec très peu d'annotations conflictuelles (voir Section 4.2.3.2) ou à celles associées à un type ayant déjà un nombre significatifs d'instances dans les dictionnaires.

De la même manière, nous mettons à jour les scores des valeurs existantes dans les dictionnaires après le traitement de chaque source.

4.3 Expérimentations

Nous présentons dans cette section les résultats expérimentaux de notre algorithme d'extraction ciblée. Nous nous focalisons principalement sur la précision de l'extraction, en comparant nos résultats à ceux des deux prototypes publiquement disponibles `ExAlg` [AGM03] et `RoadRunner` [CMM01]. Concernant le temps d'execution, nos tests ont montré qu'une fois les reconnaisseurs nécessaires sont mis en place, le temps de

construction du wrapper varie de 4 à 9 secondes. Ensuite, le temps requis pour extraire les données est négligeables pour toutes les sources traitées. Les expérimentations ont été réalisées sur un ordinateur équipé d'une CPU intel pentium 2.8GHz et de 3Go de RAM.

4.3.1 Jeu de données

Nos tests ont été réalisés sur cinq domaines différents : `concerts`, `albums`, `livres`, `publications` et `voitures`.

Pour ne pas biaiser la qualité de l'extraction par le choix des sources, nous avons essayé d'adapter une démarche objective pour la sélection de sources pertinentes pour chaque domaine. Pour ce faire, nous avons utilisé la plateforme *Mechanical Turk de Amazon* [2], pour simuler l'étape de recherche de sources pertinentes. Pour chaque domaine, excepté celui des publications, nous avons demandé à dix utilisateurs de nous fournir une liste ordonnée de dix sources Web structurées contenant des listes enregistrements de ce domaine. Nous avons sélectionné les dix meilleures sources apparaissant dans les listes fournies par les utilisateurs. Une exception a été faite pour les concerts, pour lesquels nous avons eu moins de réponses (les utilisateurs n'avaient pas pu donné plus que cinq sources) [3]. Pour ce qui est des publications scientifiques, pour lesquelles nous avons considéré que l'approche Mechanical Turk était moins adaptée, nous avons utilisé la liste complète des sources citées dans les expérimentations de [SMM+08].

Nous avons pris en compte également les sources utilisées dans le système de construction de wrappers TurboSyncer [CCZ07], pour les livres, les albums et les voitures [4].

Ensuite, le jeu de données pour lequel nous rapportons nos résultats d'extraction a subi une étape de simplification (telle qu'elle est décrite dans la Section 3.2). Sans donner d'avantages de détail sur cette étape préliminaire, nous précisons que dans plus de 80% des cas, notre heuristique permet de réduire le nombre de segments non significatifs existants dans les pages.

Pour chaque source, nous avons sélectionné aléatoirement une *cinquantaine* de pages, à l'exception des sites comme *autoweb* et *iowastate* qui ne contiennent pas autant de pages. Nous nous sommes principalement focalisés sur les sources qui proposent des pages de types listes. Afin de montrer que la technique proposée fonctionne bien pour tout type de pages structurées, nous avons collecté également des pages détaillées (dont le contenu n'est pas constitué de listes) pour chaque source du domaine concert.

2. https ://www.mturk.com/mturk/welcome

3. Les réponses détaillées de Mechanical Turk sont disponibles sur la page de notre projet http ://perso.telecom-paristech.fr/∼derouich/datasets/

4. Nous avons pas pu obtenir des auteurs l'implantation du prototype ou les résultats détaillés par source [CCZ07].

Pour chaque domaine, les sources choisies sont testées avec les SODs décrits ci-dessous :

1. **Concerts.** Un objet concert est décrit par une structure arborescente. Le premier niveau est composé de (i) deux types d'entités élémentaires (feuilles), *artiste* et *date*, (ii) un `type tuple` qui précise la *localisation* du concert. Ce tuple est composé de deux types d'entités, *adresse* et *nom de la salle de spectacle*.
2. **Albums.** Un objet album est décrit par un tuple composé de quatre types d'entités élémentaires : *titre*, *artiste*, *prix* et *date*. La date étant optionnelle.
3. **Livres.** Un object livre est décrit par une structure arborescente à deux niveau. Le premier niveau est composé d'un tuple regroupant trois types d'entités : *titre*, *prix*, *date* et un `type set` d'*auteurs*. Ce dernier désigne un ensemble d'instances du type d'entité *auteur*. La date est optionnelle.
4. **Publications.** Un objet publication est décrit par une structure arborescente à deux niveaux. Le premier niveau est composé de deux types d'entités : *titre*, *date* et un `type set` d'*auteurs*. La date est également optionnelle ici.
5. **Voitures.** Un objet voiture est décrit par un `type tuple` composé de deux types d'entités : *marque de voiture* et *prix*.

4.3.2 Résultats

Les résultats d'extraction sont résumés dans la Tableau 4.2. Nous montrons la précision de l'étape de construction du template, i.e., comment plusieurs composants (types) du SOD ont été correctement identifiés dans les structures des pages, en se basant sur un échantillon des 20 pages les plus annotées. Nous avons vérifié dans nos tests la présence des attributs optionnels dans les source (colonne *optionnel* du tableau). La précision des résultats obtenus a été évaluée par comparaison avec le résultat d'une extraction exhaustive et manuelle à partir de chaque source (nous avons affecté le Golden Standard Test). Les résultats sont classifiés comme suit :

- **Attributs corrects (A_c) et objets (O_c).** *Un attribut* est indiqué comme étant correct (A_c) si ses valeurs ont été extraites correctement. *Un objet* est correct (O_c) si tous ses attributs ont été extraits correctement.
- **Attributs partiellement corrects (A_p) et objets (O_p).** *Un attribut* est indiqué comme étant partiellement correct (A_p) si :
    1. les valeurs pour deux ou plusieurs attributs sont extraites ensemble (comme instances du même champ dans le template) et qu'elles sont affichées de cette manière sur les pages (par exemple, le titre d'un livre et le nom de l'auteur peuvent apparaitre dans le texte de la page comme une seule information atomique),

2. les valeurs correspondantes au même type d'entité du SOD sont extraites comme instances de champs distincts (ceci peut se produire que dans le cas des pages listes)

Un objet est classifié comme étant partiellement correct (O_p) si les attributs extraits sont uniquement corrects ou partiellement corrects.

– **Attributs incorrects A_i et objets O_i.** *Un attribut* est classifié comme étant incorrecte (A_i) si les valeurs extraites sont incorrectes, i.e, elles représentent un mélange de valeurs correspondantes aux attributs distincts du schéma implicite. *Un objet* est classifié incorrecte (O_i) s'il existe au moins un attribut incorrect extrait.

Nous montrons dans le Tableau 4.2 l'ensemble des résultats obtenus (corrects, partiellement corrects et incorrects pour les attributs et les objets). La colonne N_o indique le nombre d'objets contenus dans la source.

Mesures de performances. Nous utilisons deux mesures de *précision* pour évaluer la qualité d'extraction : (i) la précision pour une extraction correcte (P_c), et (ii) la précision pour une extraction partiellement correcte (P_p), définies comme suit :

$$P_c = \frac{O_c}{N_o} \quad and \quad P_p = \frac{O_c + O_p}{N_o}$$

où O_c désigne le nombre total d'objets corrects extraits, O_p désigne le nombre d'objets partiellement corrects extraits et N_o le nombre total d'objets dans la source. Notez que dans notre contexte, *le rappel* (recall) est égale à *la précision* pour une extraction correcte, puisque le nombre d'objets existants dans les pages est égal au nombre d'objets corrects, partiellement corrects et incorrects extraits ($N_o = O_c + O_p + O_i$).

4.3.2.1 L'impact du choix de l'échantillon sur la qualité de l'extraction

Nous avons analysé la différence dans la précision de l'extraction selon la façon dont l'échantillon de pages est choisi pour la construction du template. Pour cela, nous avons comparé les résultats obtenus en utilisant l'approche de sélection décrite précédemment (Section 5.3) à une approche naïve qui choisit l'échantillon de façon aléatoire. Les résultats pour les deux approches sont résumés dans le Tableau 4.3. Ces résultats montrent qu'une sélection rigoureuse de l'échantillon, en se basent sur le SOD ciblé, peut améliorer les résultats finaux de manière significative .

			Attributs				Objets			
	Domaines	**Sites**	*Optionnel*	A_c	A_p	A_i	N_o	O_c	O_p	O_i
1.	Concerts	zvents (detail)	yes	4/4	0/4	0/4	50	50	0	0
2.		zvents (list)	yes	4/4	0/4	0/4	150	150	0	0
3.		upcoming.yahoo (detail)	yes	4/4	0/4	0/4	50	50	0	0
4.		upcoming.yahoo (list)	yes	3/4	0/4	1/4	250	0	0	250
5.		eventful (datail)	yes	1/4	2/4	1/4	50	0	0	50
6.		eventful (list)	no	3/4	0/4	0/4	500	500	0	0
7.		eventorb (detail)	yes	4/4	0/4	0/4	50	50	0	0
8.		eventorb (list)	yes	4/4	0/4	0/4	289	289	0	0
9.		bandsintown (detail)	yes	4/4	0/4	0/4	50	50	0	0
10.	Albums	amazon	yes	4/4	0/4	0/4	600	600	0	0
11.		101cd	no	1/4	2/4	0/4	1000	0	1000	0
12.		towerrecords	yes	4/4	0/4	0/4	1250	1250	0	0
13.		walmart	yes	3/4	1/4	0/4	2300	0	2300	0
14.		cdunivers	yes	4/4	0/4	0/4	1700	1700	0	0
15.		hmv	yes	4/4	0/4	0/4	600	600	0	0
16.		play	no	3/4	0/4	0/4	1000	1000	0	0
17.		sanity	yes	4/4	0/4	0/4	2000	2000	0	0
18.		secondspin	yes	4/4	0/4	0/4	2500	2500	0	0
19.		emusic (discarded)								
20.	Livres	amazon	yes	4/4	0/4	0/4	600	600	0	0
21.		bn	yes	4/4	0/4	0/4	500	500	0	0
22.		buy	no	3/4	0/4	0/4	1300	1300	0	0
23.		abebooks	no	3/4	0/4	0/4	500	500	0	0
24.		walmart	yes	3/4	0/4	1/4	2300	0	0	2300
25.		abc	yes	4/4	0/4	0/4	651	651	0	0
26.		bookdepository	yes	4/4	0/4	0/4	1000	1000	0	0
27.		booksamillion	yes	4/4	0/4	0/4	1000	1000	0	0
28.		bookstore	no	2/4	0/4	1/4	730	0	0	730
29.		powells	no	3/3	0/3	0/3	1000	1000	0	0
30.	Publications	acm		3/3	0/3	0/3	1000	1000	0	0
31.		dblp		3/3	0/3	0/3	500	500	0	0
32.		cambridge		3/3	0/3	0/3	230	230	0	0
33.		citebase		3/3	0/3	0/3	500	500	0	0
34.		citeseer		1/3	2/3	0/3	500	0	500	0
35.		DivaPortal		3/3	0/3	0/3	500	500	0	0
36.		GoogleScholar		1/3	0/3	2/3	500	0	0	500
37.		elseivier		3/3	0/3	0/3	983	983	0	0
38.		IngentaConnect		2/3	0/3	1/3	500	0	0	500
39.		IowaState		0/3	0/3	3/3	481	0	0	481
40.	Voitures	amazoncars		2/2	0/2	0/2	54	54	0	0
41.		automotive		0/2	2/2	0/2	750	0	750	0
42.		cars		2/2	0/2	0/2	500	500	0	0
43.		carmax		2/2	0/2	0/2	500	500	0	0
44.		autonation		2/2	0/2	0/2	500	500	0	0
45.		carsshop		2/2	0/2	0/2	500	500	0	0
46.		carsdirect		0/2	2/2	0/2	1500	0	1500	0
47.		usedcars		2/2	0/2	0/2	1250	1250	0	0
48.		autoweb		2/2	0/2	0/2	250	250	0	0
49.		autotrader		2/2	0/2	0/2	393	393	0	0

TABLE 4.2 – Résultats d'extraction

	Sélection basée-sur-SOD		Sélection aléatoire	
Domaines	P_c	P_p	P_c	P_p
Concerts	86.10	86.10	61.78	61.78
Albums	74.52	100	69.88	95
Livres	68.37	68.37	56.36	62
Publications	65.21	74	65.21	65.21
Voitures	75.79	100	75.79	100

TABLE 4.3 – L'impact du choix de l'échantillon sur la qualité d'extraction : selection aléatoire vs. basée-sur-SOD. (%)

4.3.2.2 Comparaison avec les approches existantes

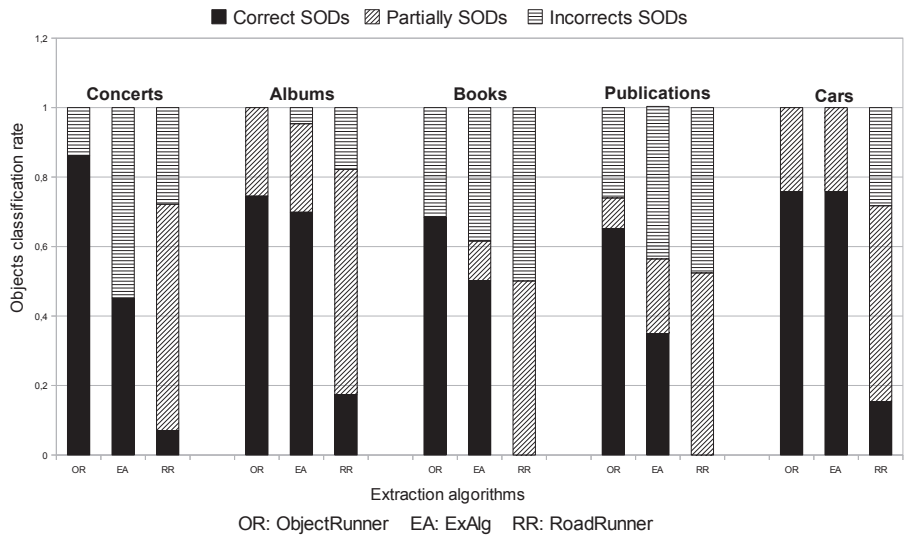
Nous avons comparé les résultats d'`ObjectRunner` aux deux travaux proches `ExAlg` [5] [AGM03] et `RoadRunner` [6] [CMM01]. Nous détaillons dans le Tableau 4.4 et la Figure 4.5 les résultats obtenus.

	OR		EA		RR	
Domaines	P_c	P_p	P_c	P_p	P_c	P_p
Concerts	86.10	86.10	45.17	45.17	6.95	72
Albums	74.52	100	69.88	95	17.37	82
Livres	68.37	68.37	50.10	62	0	50,10
Publications	65.21	74	34.83	56	0	52.39
Voitures	75.79	100	75.79	100	15.28	72

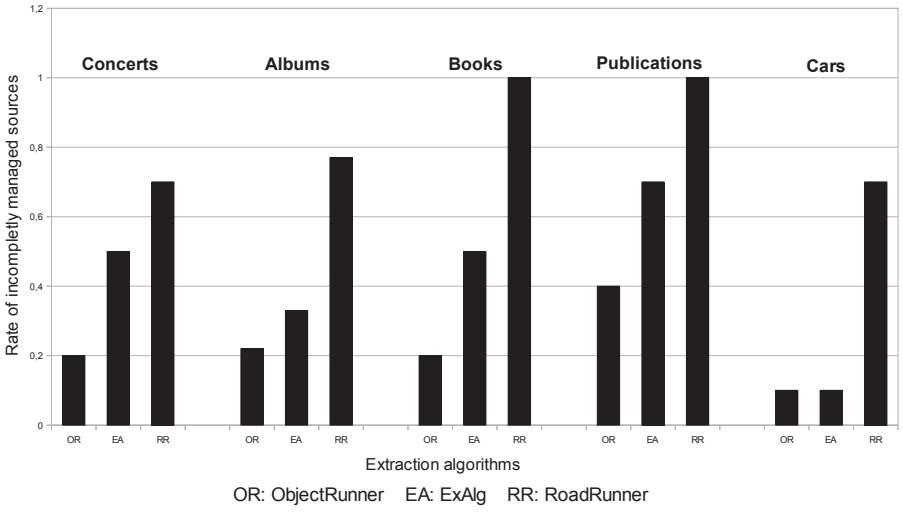
TABLE 4.4 – Résultats de performances (%)

Le Tableau 4.4 présente les deux valeurs de précision obtenues pour chaque domaine et pour chaque système. La colonne `OR` donne les résultats des tests pour `ObjectRunner`, la colonne `EA` donne les résultats pour `ExAlg` et la colonne `RR` donne les résultats pour `RoadRunner`. Nous pouvons constaté qu'`ObjectRunner` dépasse `RoadRunner` par une marge significative (environ 60%) pour les cinq domaines. Nous avons constaté que `RoadRunner` échoue en grande partie sur les pages listes, où la plupart des objets extraits sont partiellement corrects (les valeurs (instances) de listes correspondantes au même attribut du SOD sont extraites séparément). `RoadRunner` ne parvient pas à bien gérer les pages listes qui ont des structures "trop régulières". Lorsque le nombre d'enregistrements

5. Le prototype utilisé ici nous a été fourni par les auteurs.
6. Le prototype est disponible sur http ://www.dia.uniroma3.it/db/roadRunner.



(a) Classification des objets



(b) Sources gérées partiellement

FIGURE 4.5 – ObjectRunner comparison

dans les pages en entrée est constant, ce qui est le cas de nos sources pour les livres et les publications, environ 50% des extractions faites par `RoadRunner` sont partiellement corrects. Nous observons globalement que `ExAlg` est meilleur que `RoadRunner`, mais plusieurs attributs sont extraits ensemble dans plusieurs sources. Les résultats de précision des deux systèmes semblent assez faibles. Néaumoins, il faut noter que ceci se produit dans le contexte d'extraction ciblée, où seulement une fraction (petite) des informations

contenues dans les pages est intéressante.

Dans la Figure 4.5, nous fournissons deux illustrations pour la qualité d'extraction : (a) le taux des objets corrects, partiellement corrects et incorrects qui ont été extraits (en fonction de la précision du template qui a été inféré pour chaque source), et (b) la fraction de sources qui ont été incomplètement gérées (c-à-d, avec des attributs partiellement corrects ou incorrects).

Nous comparons les trois algorithmes sur leur capacité à gérer une source correctement. Nous observons que pour tous les trois domaines `ObjectRunner` dépasse `ExAlg` et `RoadRunner`. Cependant, il reste environ 20% de sources incomplètement gérées pour les trois domaines (concerts, albums et livres), 40% pour les publications et 10% pour les voitures.

Ces résultats sont plutôt encourageants puisqu'à notre avis il est bien plus important de pouvoir gérer correctement au moins quelques sources d'un domaine donné, que de gérer la plupart ou toutes les sources de façon incomplète (ou partiellement complète). Comme les données du Web ont tendance à être très redondantes (par exemple, les objets concerts présentés `yellowpages.com` sont aussi présents dans `zvents.com`), les objets qui ne sont pas correctement extraits d'une source peuvent l'être à partir d'une autre source.

4.3.2.3 Complétude du dictionnaire

En général, nous ne pouvons pas avoir un dictionnaire complet contenant toutes les instances possibles des types qu'il référence (ex, tous les titres des livres ou des albums). Concernant cet aspect, l'objectif principal de nos expérimentations était de prouver que lorsque les dictionnaires assurent une couverture raisonnable du contenu des pages, l'algorithme d'extraction fonctionne convenablement. À cet effet, dans nos tests nous avons complété chaque dictionnaire si nécessaire pour avoir au moins 20% d'instances des sources en entrée.

Nous rapportons dans le Tableau 4.5 l'influence de la complétude du dictionnaire sur la précision. Nous avons fait des tests avec un dictionnaire qui a une couverture de 10% que nous avons enrichi pour atteindre une couverture de 20%. Les tests ont été réalisés pour les domaines des livres et des albums. Nous obervons une légère dimunition de la précision pour une couverture à 10%. De notre point de vue, comme les données Web ont tendance à être très redondantes, par l'enrichissement des dictionnaires après chaque extraction par le système `ObjectRunner`, nous améliorent progressivement la précision d'extraction.

	10%		20%	
Domaines	P_c	P_p	P_c	P_p
Albums	69.88	95	74.52	100
Livres	50.10	62	68.37	68.37

TABLE 4.5 – Précision selon la couverture du dictionnaire : 10% vs. 20%. (%)

4.3.2.4 Variation du paramètre *support*

Rappelons que le template est composé de classes d'équivalence (EQs) formées à partir de tokens. Chaque EQ regroupe les tokens partageant le même vecteur d'occurrence. Toutes les EQs découvertes ne sont pas forcément gardées pour la construction du template, sauf celles qui sont *fréquentes* et *larges*. Plus précisément, les EQs dont le *support* (le nombre de pages où les tokens de la EQ apparaissent) ou le *size* (le nombre de tokens dans une EQ) insuffisants sont éliminées. Ces deux paramètres, le *support* et le *size*, peuvent avoir un impact sur la qualité de l'extraction dans certains cas. Comme dans notre approche, nous pouvons estimer automatiquement la qualité de l'extraction (en se basant sur les annotations conflictuelles 4.2.3.2, qui apparaissent sur les valeurs extraites pour le même attribut), nous varions ces paramètres et nous re-exécutons l'algorithme de construction du wrapper à nouveau, si nécessaire.

Pour les sources du domaine des publications, les pages sélectionnées ont été obtenues à travers des formulaires (Web caché), en spécifiant les mots-clés "Web information extraction" pour certains sites. Nous avons obtenus des listes de publications du domaine choisi (Web information extraction). En procédant ainsi, beaucoup de publications apparaissant dans les pages contenaient les mots web, information, ou d'autres mots-clés de ce domaine. Ainsi, ces mots peuvent se retrouver dans les EQs, alors qu'ils ne font pas partie du template. Afin d'éliminer ces EQs lors de la construction du template, nous avons essayé de jouer sur le paramètre *support* (entre 3 et 5 dans nos tests). Le Tableau 4.6 résume les résultats des tests effectués. Nous avons constater qu'en faisant varier le *support* et en re-exécutant l'algorithme, la précision de l'extraction peut être améliorée de manière significative.

	3		4		5	
Domaine	P_c	P_p	P_c	P_p	P_c	P_p
Publications	43.61	56.43	47.65	56.43	65.21	74

TABLE 4.6 – Précision en fonction de la variation du paramètre *support* : 3, 4 vs. 5. (%)

4.4 Conclusion

Ce chapitre présente une approche automatique d'extraction d'information à partir de pages Web structurées. Il défend l'idée d'une interrogation à deux étapes, dans laquelle une description intentionnelle des données ciblées est fournie avant la phase d'extraction. L'utilisateur spécifie une structure (SOD) des objets recherchés, qui sera utilisée dans la phase d'extraction pour annoter les pages sources et extraire les objets découverts.

Ce processus est indépendant du domaine, il s'applique à n'importe quelle structure, plate ou imbriquée, décrivant des objets du monde réel. Notre approche est complètement automatique et ne nécessite pas d'annotations manuelles ou des procédures d'apprentissage. Le principe général d'`ObjectRunner` est de privilégier les solutions flexibles, qui sont rapides, avec une bonne précision et un taux d'extraction satisfaisant. En pratique, il peut y avoir des sources traitées de manière peu satisfaisante, ce qui conduit à la perte de certains objets qui, étant donné la redondance du Web, pourraient très probablement être trouvés dans une autre source (voir même dans le même site).

Avoir une description intentionnelle des données ciblées a un double intérêt : (i) la qualité des données extraites peut être améliorée, et (ii) des traitements inutiles sont évités (ceux des données contenues dans les pages mais qui n'intéressent pas l'utilisateur). La qualité des extractions est améliorée grâce aux annotations qui permettent dans notre approche de distinguer de façon plus fine les rôles des tokens. La combinaison entre la structure des pages et les annotations donne les meilleurs résultats, comme le montre nos tests.

Nous validons notre approche par des expérimentations intensives et des comparaisons avec les deux systèmes les plus référencés pour l'inférence automatique de wrappers. Dans tous les cas, pour les cinq domaines choisis, notre système récupère plus d'instances d'objets avec moins d'erreurs. Nos résultats sont rendus plus pertinents par le fait que les sources testées n'ont pas été sélectionnées aléatoirement ou de manière subjective. Ces sources ont été fournies par des experts (de Mechanical Turk), comme étant les plus pertinentes pour chaque domaine.

Chapitre 5

Sélection de Sources Structurées

Dans ce chapitre nous détaillons le processus de sélection automatique de sources structurées pertinentes pour la phase d'extraction. Précisément, nous considérons le problème de pré-traitement, d'indexation et de sélection pour l'inférence de wrappers.

En suivant le paradigme d'`ObjectRunner` d'avoir une interrogation du Web à deux étapes, nous voulons intégrer l'inférence de wrappers dans un scénario complètement automatique où, étant donné la description intentionnelle (requête SOD), les sources les plus pertinentes doivent être sélectionnées à partie d'un répertoire de sources récupérées et indexées du Web (crawled). [1]

Sélectionner les sources prometteuses pour l'extraction au moment de l'interrogation (query time) nécessite une comprehension au préalable *(offline)* adaptée de la sémantiques des données, de la structure hiérarchique et des caractéristiques visuelles des pages. Nous présentons ci-dessous une approche – à notre connaissance, ce travail est le premier qui permet de déterminer les sources les plus pertinentes pour une requête donnée – pour traiter cette problématique. L'approche proposée est construite sur des techniques de : (i) recherche d'information basée sur des ontologies, (ii) indexation de données structurées, et (iii) recherche top-k sur des données semi-structurées (structures d'arbres).

5.1 Architecture générale

Le défi que nous considérons est de traiter, annoter sémantiquement, analyser et indexer de manière optimale et efficace un grand corpus de pages Web structurées. Nous proposons une approche guidée par des connaissances sémantiques, basée sur des ontologies populaires comme Freebase [BEP+08] ou Yago [SKW07].

1. Dans nos expérimentations pour la tâche d'extraction du Chapitre 4, ces sources ont été simulées avec la plateforme Mechanical Turk.

Étant donné le SOD fourni par l'utilisateur, notre système analyse les sources et sélectionne celles (les top-k) qui sont les plus pertinentes pour la requête utilisateur.

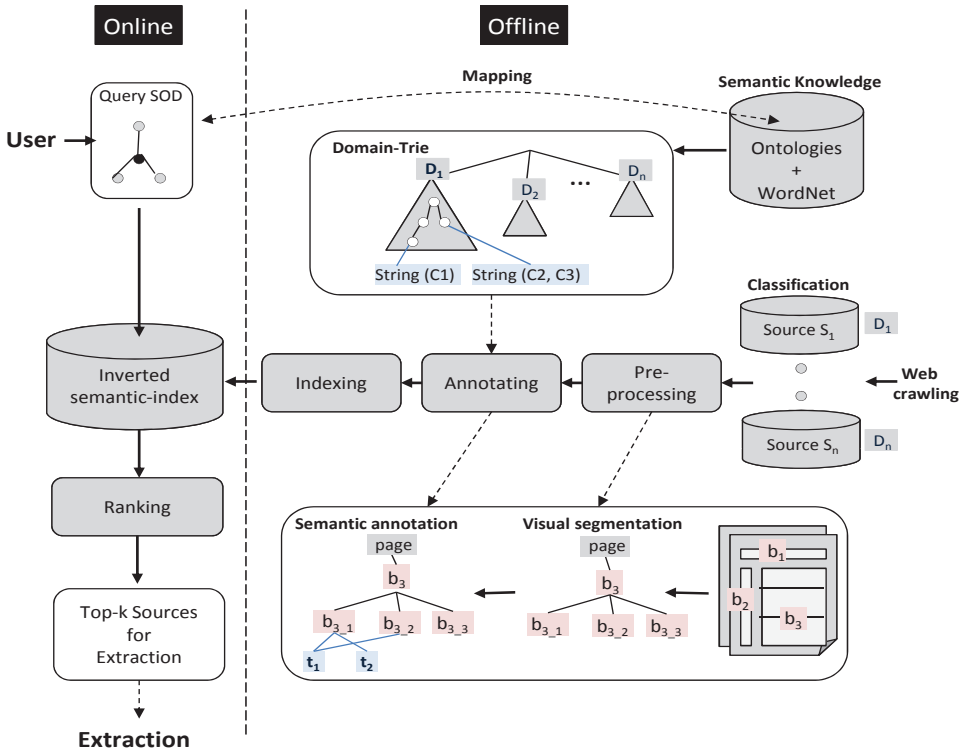


FIGURE 5.1 – Architecture générale pour la sélection de sources.

Nous présentons dans la Figure 5.1 un aperçu de l'architecture de notre système de sélection de sources, en termes de modules de traitement *offline* et *online* qui le composent.

1. Les sources Web (collections de pages) sont pré-traitées, ceci permet de déterminer la structure hiérarchique des blocs qui composent les pages de chaque source, en se basant sur les caractéristiques visuelles et structurelles des pages.
2. Une annotation sémantique est effectuée sur les contenus des blocs, en faisant correspondre les instances au sein d'un bloc aux concepts dans l'ontologie.
3. Une description est établie pour chaque source, en termes d'annotations, des caractéristiques structurelles et d'autres critères utiles pour la construction du template.
4. Les descriptions de sources sont indexées et stockées.
5. Un algorithme top-k permet de trouver les sources les plus pertinentes pour un SOD donné, en exploitant l'index construit précédemment.

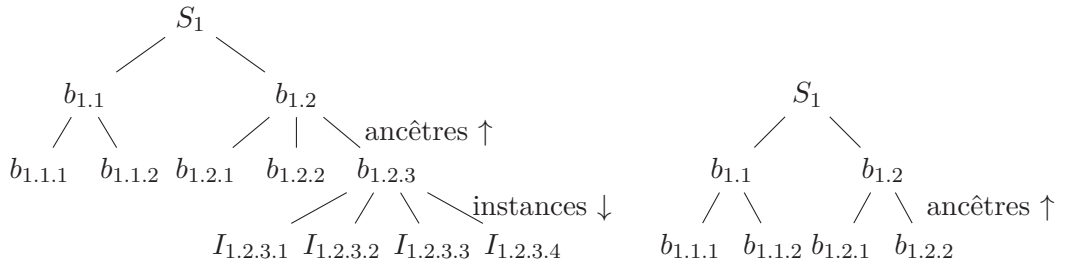
5.2 Construction de la structure hiérarchique d'une source

Dans le Web structuré, chaque page est une instance d'une structure prédéfinie spécifique à la source considérée. Pour découvrir cette structure commune, nous analysons plusieurs pages de chaque source et nous comparons leurs structures.

Une étape préliminaire consiste à déterminer le segment (bloc) visuel le plus grand et le plus riche en données, (voir Section 3.2). Une fois ce bloc identifié, sur l'ensemble des pages, nous procédons à l'identification des sous blocs présents dans toutes les pages et ceux qui peuvent être optionnels [2], en se basant sur les caractéristiques de chaque bloc : sa taille visuelle, le nom de la balise contenant son contenu, les métadonnées de cette balise (des attributs). Uniquement les blocs présents dans toutes les pages *(non optionnels)* vont constituer la structure hiérarchique de la source.

Nous distinguons deux catégories de pages Web : les *pages listes* et les *pages détaillées*. Dans une page liste, un bloc donné peut avoir des instances multiples. Par exemple, une liste de livres ou d'événements culturels représente plusieurs instances d'une même structure de bloc. Dans une page détaillée chaque bloc a au maximum une instance.

Nous pouvons ainsi parler de diverses instances (occurrences) du même bloc dans la même page ou dans différentes pages. La Figure 5.2 donne un exemple de structures de sources contenant ces deux types de pages. L'arbre de la Figure 5.2(b) représente une source contenant des pages détaillées. Dans ce cas, le bloc et son instance sont confondus dans la même structure de la source. L'arbre de la Figure 5.2(a) représente une source contenant des pages listes, le bloc $b_{1.2.3}$ a quatre instances ($I_{1.2.3.1}, I_{1.2.3.2}, I_{1.2.3.3}$, $I_{1.2.3.4}$).



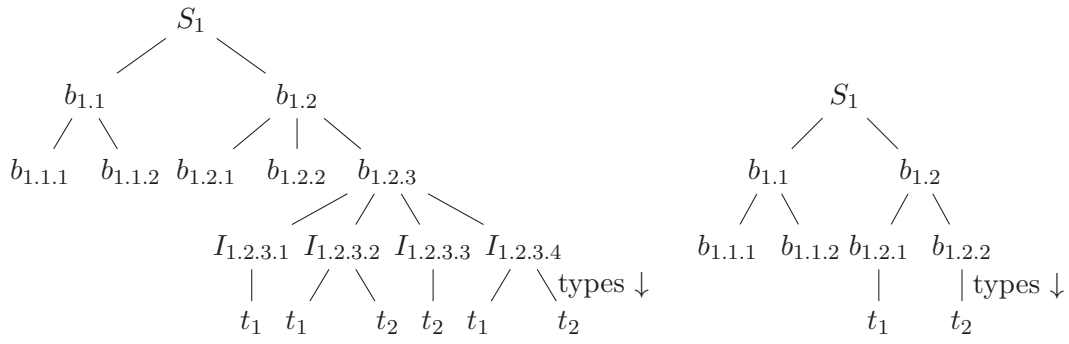
(a) Source contenant des pages listes (b) Source contenant des pages détaillées

FIGURE 5.2 – Exemple de structures d'arbres pour une source S_1.

2. Ces blocs sont présents uniquement dans certaines pages, correspondent généralement à des annonces publicitaires, des informations complémentaires, etc.

5.3 Annotation sémantique des pages

L'annotation sémantique des pages est effectuée au niveau des blocs feuilles de la structure construite pour chaque source. Elle consiste à reconnaitre les instances de types contenues dans ces blocs. La Figure 5.3 montre l'évolution des structures présentées dans la Fagure 5.2, après la phase d'annotation. Dans cet exemple (Figure 5.3(a)), le type t_1 apparait dans les instances $I_{1.2.3.1}$, $I_{1.2.3.2}$ et $I_{1.2.3.4}$ du bloc $b_{1.2.3}$. Et le type t_2 apparait dans les instances $I_{1.2.3.2}$, $I_{1.2.3.3}$ et $I_{1.2.3.4}$ du même bloc. Pour la source des pages détaillées (Figure 5.3(b)), les types sont directement associés aux blocs feuilles.



(a) Source contenant des pages listes

(b) Source contenant des pages détaillées

FIGURE 5.3 – Exemple de structures d'arbres `annotés` pour une source S_1.

5.3.1 Détection des domaines des sources

Avant d'entamer la tâche d'annotation de pages, nous avons choisi de classer les sources selon une liste de domaines, auxquels elles appartiennent. Ceci afin de réduire le nombre de concepts à traiter et faciliter la tâche d'annotation. En particulier, le taux d'erreurs d'annotation sera réduit car les instances peuvent correspondre à différents concepts au niveau de l'ontologie, qui ne sont pas forcément liés. Par exemple, dans l'ontologie YAGO l'instance "Madonna" se retrouve dans plus de 100 concepts (classes), parmi lesquels plusieurs sont étrangers au domaine de la musique.

Pour réaliser cette classification des sources, nous exploitons les métadonnées des pages (les mots-clés des métadonnées, la description des métadonnées, les titres de pages, etc.) pour déterminer les termes les plus significatifs qui conduiront à la détection du domaine de la source considérée.

Certaines ontologies existantes sont organisées en domaines. Un des exemples les plus riches et les plus populaires est Freebase, qui offre une classification en domaines brutes (plus de 90 domaines). Les données dans Freebase sont créées à partir de grands jeux de

<meta name="description" content="Online shopping for millions of new & used books on thousands of topics at everyday low prices."/>
<meta name="keywords" content="Books, New Books, Used Books, Dictionary, Harry Potter, Bible, Poetry, Magazines, Subscription, Audio Books, Textbooks
Biographies, Children's Books, Literature, Science Fiction, Fiction, Cookbooks, Psychology Books, Photography, Computers & Internet, Travel,
Pregnancy Books, Novels, Romance Novels, Reference Books, Classics, Business, Stephen King, Investing, Nora Roberts, Religion, For Dummies,
Spirituality, Health, Parenting, Mystery Novels, Family, Amazon.com"/>
<title>Amazon.com Books: New & used books,textbooks,children's books,biographies & more</title>

FIGURE 5.4 – Un segment de code source de la page Amazon Livre.

données publiquement disponibles comme Wikipedia, MusicBrainz, WordNet, etc. Elles couvrent beaucoup de thèmes populaires, parmi lesquels on trouve les films, la musique, les livres, les lieux géographiques, etc.[3] L'étude menée par [WHHK11] a montré que Freebase est la plus complète – parmi les cinq taxonomies les plus populaires testées – en terme d'instances de concepts (couvrant 80% des top 50 requêtes de Bing). Nous avons donc choisi Freebase car elle nous paraissait idéale pour le détection des domaines de sources.

Nous avons construit à partir de Freebase plusieurs structures de trie *(arbre préfixe)* : une pour tous les concepts (appelée *concept-trie*), et une pour les instances de chaque domaine (appelée *instance-trie* du domaine). Par exemple, en utilisant ces structures de tries, le type *Book* sera choisi parmi les métadonnées de la page du livre Amazon dans la Figure 5.4. Dans le même exemple, l'instance "Harry Potter" correspondrait au type *Book*, "Nora Roberts" correspondrait au type *Author*, sachant que les deux concepts apparaissent dans le domaine *Book*.

La détection du domaine est réalisée en prenant en compte :

1. la correspondance entre des métadonnées d'un échantillon de sources avec le *concept-trie*. Après une simple analyse du texte (tokenization, stemmatisation et le POS tagging), nous obtenons une liste des domaines candidats.
2. la correspondance entre des métadonnées avec les instances de la liste des domaines obtenues précédemment afin de raffiner cette liste. Ceci est effectué en utilisant les *instance-trie* de chaque domaine candidat.

Ensuite, nous sélectionnons les top-2 domaines les plus probables.

Dans l'approche proposée, le choix de sélectionner les top-2 domaines pour une source donnée n'est pas considéré arbitraire. Il suit la logique qu'une source est associée généralement à un seul *(le premier)* domaine particulier (ex, Livre, Musique, etc.). Nous avons choisi d'ajouter un domaine complémentaire *(le deuxième)* dans le cas où le

3. Freebase couvre 300 millions de triplés à travers 12 millions de thèmes qui recouvrent plus de 17.000 types et 46.000 propriétés.

SOD spécifié contient des types appartenant aussi à d'autres domaines. Par exemple, si l'utilisateur veut connaitre la salle de spectacle (le type *theater*) où le concert aura lieu, ce type appartient au domaine *Location* dans Freebase.

Après le choix des domaines, l'annotation du texte est faite avec seulement les deux domaines sélectionnés.

Généralement, dans les pages Web structurées, les données sont formatées par des balises HTML. Afin d'améliorer la précision et l'efficacité de l'annotation, nous avons choisi d'annoter uniquement le texte délimité par des balises, comme dans `<balise>Harry Potter</balise>`.

5.3.2 Sélection de la meilleure classe pour chaque instance

Malgré le fait que les domaines ont été réduits à seulement deux comme indiqué dans la section précédente, une instance i peut correspondre à plusieurs concepts de l'ontologie, pendant la phase d'annotation. Afin d'arbitrer ces cas particuliers, nous avons choisi de sélectionner et de considérer seulement la meilleure classe.

Pour ce faire, nous nous sommes appuyés sur l'algorithme TreeAscent [MC08b], qui permet l'apprentissage sur des ontologies, et calcule pour chaque classe candidate c un score basé sur une mesure de similarité taxonomique T_{sim} entre les concepts d'une ontologie O, comme suit :

$$bestClass(i) = arg\max_{c \in O} \sum_{c' \in O} score(i, c') \cdot T_{sim}(c, c') \tag{5.1}$$

Dans notre système, nous avons adapté cette mesure en utilisant les mesures de similarité sémantique de WordNet *(WordNet semantic similarity measures)*[4]. Cette similarité de classes est calculée comme suit :

$$T_{sim}(c, c') \;=\; WuP(c, c') \cdot JCn(c, c'), \tag{5.2}$$

où la mesure de WuP compare les profondeurs de deux classes dans les taxonomies de WordNet, à partir de leur ancêtre commun LCS (Least Common Subsumer). La mesure de JCn compare les relations (relatedness) entre deux classes au niveau de leurs instances.

Le résultat de l'étape d'annotation. Le résultat de la phase d'annotation est une structure arborescente de blocs, dans laquelle les blocs feuilles contiennent des annotations sémantiques avec des scores de confiance associés.

4. Voir `http://marimba.d.umn.edu/similarity/measures.html`, pour plus de détails sur ces mesures de similarité.

5.4 Indexation sémantique

Dans cette section, nous présentons la technique d'indexation sémantique que nous avons développé pour l'algorithme de recherche top-k sur lequel s'appuie la sélection de sources. Une liste inversée L_t est construite pour chaque type d'entité t, en fonction de la fréquence des occurrences de t dans chaque bloc. Nous décrirons dans ce qui suit l'étape d'indexation et les deux algorithmes de détection de sources que nous proposons. Nous détaillons comment l'index est exploité et comment le score des sources est calculé dans chacun des algorithmes proposés. Nous présentons d'abord leur principe commun, puis nous les décrivons en détail dans les Sections 5.4.2.1 et 5.4.2.2.

5.4.1 Indexation des blocs feuilles

Nous calculons le poids pour chaque type t dans les blocs feuilles en utilisant la measure *tf-idf (term frequency - inverse document frequency)*. Ceci est réalisé en se basant sur les instances associées aux blocs feuilles, et en calculant le poids pour chaque position distincte (dans le bloc) dans laquelle les instances de t apparaissent. Seule la position dominante est conservée.

Pour chaque type t, une liste d'index L_t est créée et triée par ordre décroissant, dans laquelle une entrée $sc(t, b)$ donne le score de t par rapport à un bloc feuille b. Plus précisément, la fonction de score $sc(t, b)$ est définie comme suit :

$$sc(t, b) = tf(t, b) \cdot idf(t) \tag{5.3}$$

La fréquence du terme tf est le nombre d'occurrences de t dans un bloc particulier b (i.e., dans les instances de b) contenues dans les pages d'une source S donnée.
Nous calculons cette valeur comme suit :

$$tf(t, b) = \frac{\max_{path}\{\sum_{p \in S} count(t, b, p)\}}{\sum_{t' \in T, p \in S} count(t', b, p)} \tag{5.4}$$

où $count(t, b, p)$ est le nombre d'occurrences des instances de t trouvées à une position donnée ($path$), dans le bloc b de la page Web p. Nous additionnons ces calculs pour chaque path, et nous gardons celui avec la valeur maximale.

La fréquence inverse du document idf donne l'importance d'un type t dans une collection de sources $\{S_1, \ldots, S_n\}$. Nous définissons le score idf comme suit :

$$idf(t) = log \frac{|B|}{|B_t|} \tag{5.5}$$

où $B = \{(b, p) : b \in p, p \in \{S_1, \ldots, S_n\}\}$ désigne un ensemble de blocs (ou occurrences de blocs pour les pages listes), apparaissant dans toutes les pages p de la collection de sources $\{S_1, \ldots, S_n\}$ considérée ; et $B_t = \{(b', p') : t \in b', b' \in p', p' \in \{S_1, \ldots, S_n\}\}$ est l'ensemble de blocs contenant les instances de t. B_t est un sous-ensemble de B.

5.4.2 Sélection de sources

L'Algorithm 5 donne un aperçu général de la façon avec laquelle les sources sont choisies. Nous avons en entrée un ensemble de sources Web, une requête s (un ensemble de types d'entités) et un seuil k. Le résultat en sortie est une liste classée des k sources satisfaisant la requête s (i.e., contenant des instances d'objets de s).

Algorithm 5 Selection de sources

1: **Entrée :** seuil k, sources $\{S_1, \ldots, S_n\}$, requête SOD s
2: **Sortie :** R `//`top-k sources classées
3: calcul des scores locaux $sc(t, b)$, $t \in s$ et $b \in S_i$
4: établissement des listes inversées $L_{t1} \ldots L_{tm}$ pour chaque t dans s
5: **for each** t dans s **do**
6: R = top-k `(LB)` ou top-k `(HB)` `//` l'un ou l'autre des deux algo proposés
7: **end for**
8: **return** R

Pour chaque type d'entité t dans s, les listes inversées sont constituées. Puis, nous appliquons une des deux techniques de recherche top-k que nous allons décrire ci-après, pour trouver les sources les plus pertinentes pour la requête.

5.4.2.1 Recherche top-k sur les blocs feuilles (`LB`)

Dans la première approche, nommée `LeafBlocks` (`LB`), nous construisons nos index sur les blocs feuilles uniquement. Nous recherchons les blocs feuilles qui contiennent les instances de la requête (SOD) s, chaque bloc feuille est considéré en quelque sorte comme un document à part. Nous appliquons, ensuite, l'algorithme TA (Threshold Algorithm) décrit dans [FLN03] sur les listes inversées construites dans l'étape précédente.

L'Algorithm 6 précise les étapes de notre algorithme `LB`. Les listes $L_{t_1}, \ldots, L_{t_m}$ sont parcourues parallèlement et de façon coordonnées (les premiers éléments de chaque liste, puis les seconds et ainsi de suite). Pour chaque bloc b s'il répond la requête s, il doit être contenu dans chacune des listes. Danc ce cas, nous calculons son score comme suit :

$$b.score = agg(sc(t_1, b), \ldots, sc(t_m, b)) \tag{5.6}$$

où $agg()$ est une fonction monotone qui calcule le score globale de b étant donné le score de chaque type t dans b.

Lorsque le parcours parallèle des listes est terminé, les sources sélectionnées sont choisies en fonction des scores des blocs qu'elles contiennent. Seules les sources ayant les k-meilleurs scores sont sélectionnées. Chaque source est représentée dans le résultat par son meilleur bloc (celui qui a obtenu le meilleur score).

À chaque itération, le score minimum ($min.score$) et le score maximum des résultats non vus ($max.score$) sont mis à jour et la condition d'arrêt (ligne 17) est vérifiée. Le $min.score$ est le score minimum dans R (le score du $k^{ième}$ source), et $max.score$ est l'agrégation des scores des derniers blocs traités par l'accès parallèle. Dans la première itération $max.score := agg(sc(t_1, L_{t_1}[1]), \ldots, sc(t_m, L_{t_m}[1]))$, dans la seconde itération $max.score := agg(sc(t_1, L_{t_1}[2]), \ldots, sc(t_m, L_{t_m}[2]))$, et ainsi de suite.

Algorithm 6 recherche top-k sur les blocs feuilles `(algo. LB)`

1: **input :** k, listes $L_{t_1}, \ldots, L_{t_m}$
2: **output :** R
3: $R := \phi$ //top-k sources initiales classées
4: $min.score := 0$ //le score du $k^{ième}$ bloc dans R
5: **loop**
6: //accès trié en parallèle à $L_{t_1}, \ldots, L_{t_m}$
7: $b_i := L_{t_i}.getNext()$
8: **if** ($\{b_i |\ i \in 1 \ldots m\}$ est vide) **then**
9: **return** R
10: **end if**
11: calculer $b_i.score$
12: **if** ($|R| < k$ **or** $b_i.score > min.score$) **then**
13: R.insérer(b_i)
14: mette à jour $min.score$
15: **end if**
16: calculer $max.score$ //le seuil courant
17: **if** ($|R| \geq k$ **and** $min.score \geq max.score$) **then**
18: **return** R
19: **end if**
20: **end loop**
21: **return** R

5.4.2.2 Recherche top-k en tenant compte de la hiérarchie des blocs (HB)

Dans la deuxième approche, nommée `HierarchyBlocks` (`HB`), nous étendons notre technique aux blocs intermédiaires. Nous supposons qu'un bloc intermédiaire b peut répondre à une requête s, si pour chaque type t dans s, il y a un bloc feuille descendant de b qui contient t. Cela suit l'idée que la structure que l'utilisateur recherche peut être éclatée dans plusieurs blocs feuilles d'une source donnée. Dans ce cas, le plus petit ancêtre commun (LCA, *Lowest Common Ancestor*) de ces blocs sera sélectionné.

Nous attribuons à chaque bloc (et à chaque instance) de cette hiérarchie un identifiant *Dewey id* qui mémorise les relations père-fils, et permet de déterminer facilement le LCA d'ensemble de blocs.

Example. Considérons l'exemple de la Figure 5.3, contenant les types d'entités $\{t_1, t_2\}$. Le bloc $b_{1.2.3}$ est un LCA pour les blocs feuilles qui contiennent ces deux types dans la page liste, et le bloc $b_{1.2}$ est un LCA pour la page détaillée. L'identifiant du bloc LCA correspond à la partie commune des *Dewey ids* de tous les nœuds feuilles considérés.

Le problème de la sélection top-k dans cette approche peut être formalisé comme suit. Étant donné un SOD $s = [t_1, \ldots, t_m]$, soit L_{t_i} la liste inversée constituée des blocs feuilles contenants t_i. Les LCAs de s sont des LCAs des m types $t_1, \ldots, t_m$, qui peuvent être définis comme : $LCA(t_1, \ldots, t_m) = LCA(L_{t_1}, \ldots, L_{t_m}) = \{LCA(b_1, \ldots, b_m) | b_1 \in L_{t_1}, \ldots, b_m \in L_{t_m}\}$.

Afin d'attribuer un score aux blocs LCAs, nous étendons notre fonction de score. Étant donné un bloc b feuille et un type t, le score de t dans b est calculé comme suit ! :

$$sc'(t, b) = sc(t, b) \cdot \mu(d) = tf(t, b) \cdot idf(t, b) \cdot \mu(d) \tag{5.7}$$

où tf et idf sont calculés comme décrit précédemment (selon les formules 5.4 et 5.5) et d représente le nombre de niveaux séparant le bloc b du bloc LCA considéré. Cette distance peut être calculée facilement en comparant les longueurs des *Dewey ids* du bloc b et du lca :

$$d = length(id_b) - length(id_{lca}) \tag{5.8}$$

La fonction de damping $\mu()$ diminue le score des types associés à des blocs feuilles éloignés du lca, $\mu()$ est fixé à 0.9 dans nos expérimentations.

L'Algorithme 7 précise les étapes de notre seconde technique de sélection de sources (`HB`) qui prend en compte la hiérarchie des blocs. Cet algorithme est inspirée du travail réalisé sur la recherche top-k par mots-clés dans les bases de données XML [CP10].

Nous supposons que pour chaque type t, l'index inversé L_t contient la liste des numéros *Dewey ids* des blocs dans lesquels se trouvent les instances du type t. Les listes

$L_{t_1}, \ldots, L_{t_m}$ en entrée sont ordonnées par les scores locaux $sc(t, b)$, calculés en utilisant l'Équation 5.3, où $sc(t, b)$ est le poids du type t dans le bloc b de la source considérée.

Afin de déterminer les blocs LCAs, nous recherchons les jointures des numéros *Dewey ids* qui composent les listes $L_{t_1}, \ldots, L_{t_m}$. Ceci est effectué étape par étape, en utilisant une recherche ascendante (bottom-up) dans la hiérarchie de blocs. Nous comparons les longueurs des numéros *Dewey ids* dans chaque liste d'index L_t afin de déterminer la longueur minimale $min\{d_1 \ldots d_m\}$, à partir de laquelle la recherche du *lca* peut démarrer. L'opération de jointure commence par la valeur minimum d, qui correspond à la position (longueur du *Dewey id*) du premier *lca* possible dans la hiérarchie de blocs. À chaque étape, nous recherchons les blocs LCAs à un niveau donné d dans la hiérarchie. Pour ce faire, nous ne considérons que les d premiers niveaux des *Dewey ids* $(\pi_d(L_{t_1}), \ldots, \pi_d(L_{t_m}))$ quand nous réalisons la jointure.

À chaque valeur de d, nous cherchons si un *lca* existe et nous calculons son score $lca.score$, en utilisant la formule suivante :

$$lca.score = agg(sc'(t_1, b), \ldots, sc'(t_m, b))$$

où $b \in L_{t_1}, \ldots, b_i \in L_{t_m}$ et b est un bloc LCA au niveau d considéré. Les scores locaux de ces blocs sont calculés en utilisant l'Équation 5.7, qui prend en compte le facteur de damping relatif à leur distance du bloc *lca*. La fonction d'agrégation $agg()$ calcule le score globale $lca.score$, basé sur les m scores locaux obtenus.

Quand un bloc *lca* est trouvé à un niveau d, nous calculons le score des résultats qui ne sont pas encore traités *(unseen)*, qui est le score maximum $max.score$ possible pour les résultats non vus, calculé comme dans [CP10]. Lors du parcours des listes $L_{t_1}, \ldots, L_{t_m}$, soit sc'_i le score du bloc se retrouvant juste après le curseur dans L_{t_i}, et msc'_i le score maximum de la liste L_{t_i} (i.e., le score du premier bloc), sans les blocs LCAs courants.

$$max.score = \max_i \left(sc'_i + \sum_{j \in 1 \ldots m, j \neq i} msc'_j\right)) \tag{5.9}$$

À chaque niveau d les numéros *ids* des blocs LCAs trouvés sont supprimés des listes inversées, et le seuil est fixé à la valeur du score $lca.score$ le plus élevé attendu au prochain niveau $d - 1$.

5.4.2.3 Co-occurrence de types

Nous décrivons dans cette section une extension de l'algorithme `HB`, `HBI(Hierarchy Block with Instances)`. Les sources où nous pouvons trouver une instance d'un bloc qui répond complètement à la requête s (i.e., une instance d'un bloc contenant tous les types $t_1, \ldots, t_m$) ont, à priori, plus de chance de contenir les objets recherchés. Nous

Algorithm 7 Recherche top-k sur les blocs LCAs (`algo. HB`)

input : k, listes $L_{t_1}, \ldots, L_{t_m}$
output : R
for i in 1 **to** m **do**
  $d_i := max\_id\_length(L_{t_i})$
end for
for d in $min\{d_1 \ldots d_m\}$ **to** 1 **do**
  $lca := \pi_d(L_{t_1}) \bowtie \ldots \bowtie \pi_d(L_{t_n})$
  if lca est trouvé **then**
    `calculé` $lca.score$
    if lca a des $instances$ **then**
      $lca.score$:=$lca.score$ + $instance.score$
    end if
    `calculer` $max.score$
    if ($|R| < k$ **and** $max.score < lca.score$) **then**
      R.insérer(lca)
    end if
  end if
end for
return R

devons donc donner plus de poids à ces sources pour améliorer leur classement dans le résultat de sélection. Pour cela, nous calculons le nombre d'instances qui correspond complètement à la requête s. La valeur obtenu ($instance.score$) est ajoutée au score du bloc lca :

$$instance.score = \frac{|I_b^s|}{|I_b|} \cdot y \tag{5.10}$$

où y est une constante, I_b est le nombre d'instances de b, et I_b^s est le nombre d'instances de b répondant complètement à s.

Exemple. Considérons les sources $S_1, \ldots, S_n$ montrées dans la Figure 5.5, et la requête $s = \{t_1, t_2\}$. Supposons que le degré de damping est de 0.9 ($\mu(d_{lca}) = 0.9^{d_{lca}}$). Étant donné $max\_id\_length(L_{t_1}) = 4$ et $max\_id\_length(L_{t_2}) = 3$, alors l'opération de jointure commence au troisième niveau ($d = 3$), comme montré dans la Figure 5.6(b). À chaque étape, le score des blocs ayant leurs numéros *Dewey ids* dans les listes inversés sont calculés. Ainsi, le score du numéro du bloc 4.1.2 est mis à 0.81 ($sc(4.1.2.2) \times 0.9^1$) dans la Figure 5.6(b).

La première opération de jointure indique que le numéro du bloc 2.2.3 est un bloc

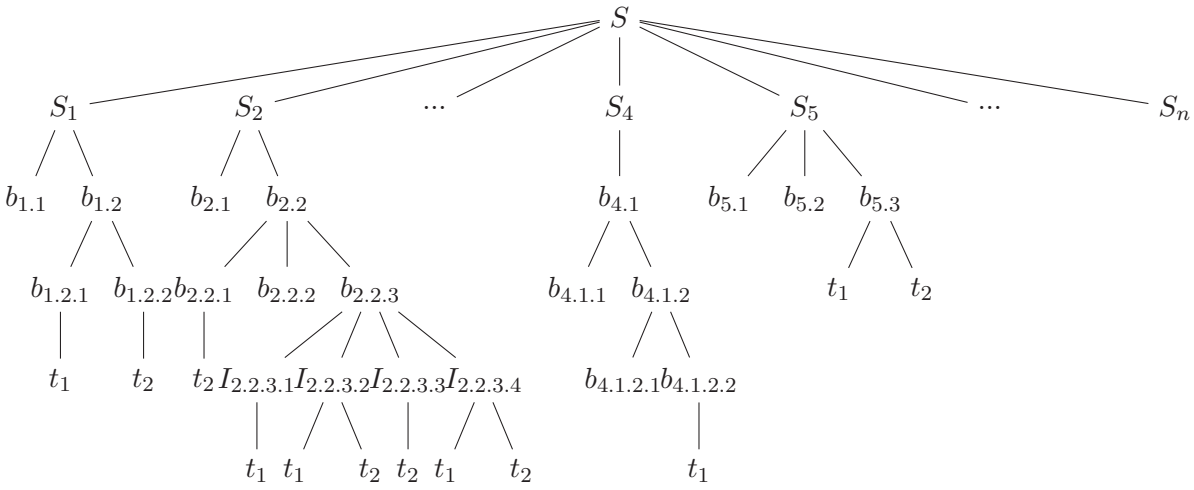


FIGURE 5.5 – Exemple d'arbres de sources

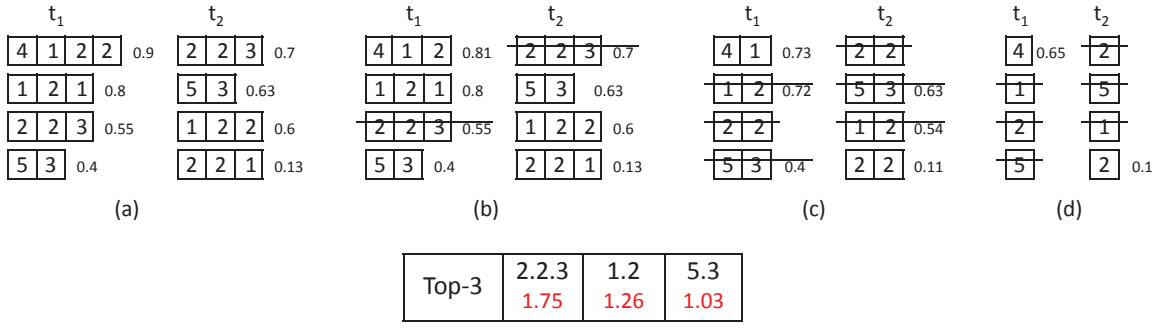


FIGURE 5.6 – Execution de l'algorithme HB

LCA. Son score d'agrégation est : $agg(0.55, 0.7) + instance.score = 0.55 + 0.7 + 0.5 = 1.75$. Supposons que le curseur est au troisième élément de chaque liste, alors le seuil à ce moment est : $max.score = max((0.4 + 0, 63), (0.13 + 0.81)) = 1.03$, qui est inférieur au score du bloc LCA trouvé. Par conséquent, le numéro du bloc 2.2.3 est inséré dans la liste top-k (résultat de la sélection).

L'opération de jointure continue au niveau 2 ($d = 2$), comme illustré dans la Figure 5.6(c). Deux blocs LCA sont trouvés : le numéro 1.2 et le numéro 5.3. Leurs scores sont $0.72 + 0.54 = 1.26$ et $0.4 + 0.63 = 1.03$. Le seuil est $max((0.11 + 0.73), (0.4 + 0.63)) = 1.03$, pour le premier *lca* trouvé (bloc 1.2, curseurs sur le quatrième élément de chaque liste), et il est de $(0.11 + 0.73) = 0.84$ pour le bloc *lca* 5.3 (curseurs sur le dernier élément de la

liste inversée L_{t_1}, et le quatrième élément de la liste inversée L_{t_2}). Par conséquent, les deux blocs sont ajoutés l'un après l'autre à la liste top-k.

5.4.3 Expansion transparente de la requête

Un aspect important dans la recherche de sources pertinentes est de pouvoir étendre de manière transparente les annotations aux types qui peuvent être pertinents pour la requête, sans être explicitement spécifiés par cette dernière. Par exemple, quand l'utilisateur spécifie le type *Artist* dans le SOD, les sources contenant le type *Singer* devraient également être considérées comme pertinentes, puisque *Singer* représente une sous-classe de la classe *Artist*.

Nous pouvons envisager une forme d'expansion de la requête guidée par une ontologie, dans laquelle des mesures comme la distance (ou la similarité) entre les classes, ainsi que les relations entre ces classes, doivent être exploitées. Dans la littérature, [TSW05] propose une méthode d'expansion de requêtes pour la recherche top-k, où l'expansion de chaque terme – basé sur des mesures de Wordnet – est effectuée de manière transparente, sans avoir à construire une requête disjonctive complexe et potentiellement coûteuse. Nous utilisons dans notre système une approche similaire, mais qui s'appuie sur des mesures de distance sémantique entre les concepts contenus dans une ontologie et la hiérarchie qui les structure. Pour calculer le score de similarité entre les concepts, nous utilisons la mesure T_{sim} de l'Équation 5.2.

5.5 Expérimentations

Nous présentons dans cette section nos résultats expérimentaux pour la sélection de sources. Nous comparons principalement les deux algorithmes proposés (`HB` et `LB`) avec une approche naïve qui ignore les blocs et qui interprète chaque page comme un document "plat". Nous avons implanté l'approche naïve à l'aide du système Lucene. L'algorithme obtenu est appelé `FD` (Flat Document) ci-dessous. Il annote le contenu de chaque page et construit des listes inversées selon un simple score *tf-idf* calculé pour chaque page.

Dans notre système les tâches de pré-traitement et d'indexation (construction des listes inversées) sont réalisées au préalable *(offline)*. L'execution des requêtes de sélection de sources se font donc très rapidement (ordre de millisecondes). Les expérimentations ont été effectuées sur une station de travail équipée d'un processeur intel pentium 2.8GHz et une mémoire vive de 3Go.

5.5.1 Jeu de données

Nos expérimentations ont été réalisées avec les sources utilisées dans les expérimentations de l'algorithme d'extraction proposé dans le Chapitre 4 (source obtenues à l'aide de la plateforme Mechanical Turk de Amazon). Nous avons choisi quatre domaines : concerts, albums, livres et voitures. Pour chaque domaine, nous avons ajouté 3 à 4 sources non structurées [5], i.e., des sources qui contiennent l'information recherchée mais sous forme textuelle (blogs, sites biographiques, pages de wikipedia).

Pour la sélection de sources nous nous sommes appuyés sur les requêtes (SOD) élémentaires suivants :

- **SOD Concerts.** composé de deux types d'entités, *artist* et *location*.
- **SOD Albums.** composé de deux types d'entités, *album* et *artist*.
- **SOD Livres.** composé de deux types d'entités, *book* et *author*.
- **SOD Voitures.** composé de deux types d'entités, *model* et *mark*.

Dans nos expérimentations, nous avons considéré seulement les reconnaisseurs de types contenus dans les ontologies (types `isInstanceOf`).

Intégration de Freebase. Nous avons utilisé Freebase pour réaliser la tâche de détection de domaine et d'annotation. Les requêtes sur la base de données Freebase ont été effectuées en utilisant le *"Metaweb Query Language"* (MQL), et une API Java de Freebase. Nous avons utilisé PATRICIA tries (Practical Algorithm to Retrieve Information Coded in Alphanumeric) pour les concepts et les instances de Freebase, qui permettent d'accélérer la recherche et utilisant moins d'espace mémoire (jusqu'à *5* fois moins de mémoire).

Nous avons testé 50 sources (40 structurées et 10 non structurées), chaque source contient environ 50 pages. Trois sources ont été éliminées dans la phase de segmentation, car VIPS (outil permettant la segmentation de pages en se basant sur leurs caractéristiques visuelles, voir Section 2.1.5.4) a échoué dans le chargement des pages de ces sources *(bookstore, autonation, eventonb-detail)*. Une source (*carshops*) a été éliminée dans la phase d'annotation, du fait que la phase d'annotation avec Freebase n'a rien donné (bien que le domaine a été correctement détecté).

5.5.2 Résultats

Nous comparons dans cette section les résultats de la sélection de sources pour les quatre algorithmes : `FD`, `LB`, `HB`, `HBI`. Nous prenons les `top-5` sources sélectionnées pour

5. livres(wiki-book, wiki-author, biography-author), voitures(wiki-car, autoblog, autocar-blog), albums et concerts(wiki-artist, wiki-album, wiki-location, biography-artist).

chaque SOD. Nous décrivons dans ce qui suit la précision de la sélection obtenue.

Concernant l'évaluation de l'efficacité de ces algorithmes, nous avons utilisé la mesure Discounted Cumulative Gain (DCG) [JK00]. Pour une requête donnée, cette mesure indique la pertinence des sources considérées et leur classement (de 1 à k). L'évaluation du classement d'une source est réalisée pour chaque position (1 à k) dans le résultat de la sélection. Le DCG cumulé à une position k dans le résultat de sélection est évalué comme suit :

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(1+i)}. \tag{5.11}$$

où rel_i est la pertinence de la source à la position i. Cette valeur est prédéfinie pour toutes les sources. Pour nos expérimentations, nous avons fixé une échelle de $0-3$ pour la pertinence des sources : 0 pour les sources complètement non pertinentes (non structurées et domaine incorrect), 1 pour les sources avec un domaine pertinent mais qui sont non structurées, 2 pour les sources structurées mais avec un domaine proche mais incorrect (ex, des sources de concerts pour une requête sur des albums), et 3 pour les sources complètement pertinentes (structurées avec un domaine correct).
Ensuite, nous calculons le DCG normalisé comme suit :

$$nDCG_k = \frac{DCG_k}{iDCG_k} \tag{5.12}$$

où $iDCG$ correspond au DCG idéal, i.e., le DCG d'un classement parfait du résultat. Dans nos tests, nous supposons que l'échelle de pertinence d'une source est fixée à 3 pour la mesure $iDCG$. Par exemple, si la sélection concerne les top-5 sources pour un SOD donné et que les sources obtenues par l'algorithme choisi ont comme valeurs de pertinences (rel_i) la liste suivante $[2, 3, 3, 3, 1]$. Étant donné que les valeurs de pertinences idéales pour une sélection de sources est la liste $[3, 3, 3, 3, 3]$, le DCG normalisé pour ce résultat ($nDCG_5$) serait de 0.82 (calculé selon les formules 5.11 et 5.12).

5.5.2.1 Comparaison des techniques proposées

Le Tableau 5.1 compare les performances des quatre algorithmes. nous pouvons noter que pour les quatre domaines, `HBI` dépasse les trois autres algorithmes. Il dépasse l'algorithme basique `FD` par une marge significative (20% à 40%). Notez également que `HB` dépasse `LB` pour tous les domaines, et que `HBI` peut améliorer `HB`. Ces résultats sont très prometteurs et montrent que notre techique de sélection de sources est efficace. La structure visuelle des pages et la co-occurrence des annotations dans les instances des blocs forment des paramètres importants dans la sélection d'une source.

Nous remarquons également dans ce tableau que les résultats obtenus pour le domaine des concerts sont pratiquement les mêmes pour les quatre algorithmes.

Table 5.1 – Résultats de performances (%)

Domaines	FD	LB	HB	HBI
Concerts	88.22	88.22	1	1
Albums	49.79	75.38	88.22	91.92
Livres	60.13	62.54	88.22	90.63
Voitures	61.42	71.90	73.20	82.56

5.5.2.2 Paramétrage du processus de selection des sources

Dans cette section, nous montrons l'impact des choix faits au niveau du pré-traitement sur la performance de la sélection. Ces choix sont les suivants : (1) limitation de l'annotation au bloc central, (2) limitation des annotations à la meilleurs classes, (3) limitation des annotations aux top-2 domaines détectés. L'impact des divers choix engagés dans le pré-traitement des sources sont évalués : la simplification au bloc central (riche en données), le choix d'annoter les instances uniquement avec une seule classe (la meilleure) et le choix de détecter d'abord les top-2 domaines et de limiter les annotations à ces derniers. Pour chacun de ces trois paramètres, nous décrivons les performances des quatre algorithmes testés, selon que ces paramètres sont pris avec et sans le raffinement en question.

Simplification au segment central

Les résultats pour les deux variantes (limitation au bloc central ou non) sont présentés dans la Figure 5.7, pour les quatre domaines. Ils indiquent que la sélection du segment central peut améliorer de manière significative la précision des résultats.

Détection du domaine

La Figure 5.8 montre les résultats avec ou sans l'étape de détection du domaine pour les quatre jeux de données considérés. Les résultats indiquent que cette étape peut également améliorer considérablement la précision de la sélection d'une source. Par exemple, sans cette étape, la précision pour le domaine des concerts est 0% pour les quatre algorithmes, et augmente jusqu'à 80% à 100% lorsque les top-2 domaines pertinents sont sélectionnés. Notez que le choix des top-2 domaines donnait de meilleurs résultats pour ce jeu de données et les SODs testés. Cependant, ce paramètre peut varier selon les requêtes et les sources considérés.

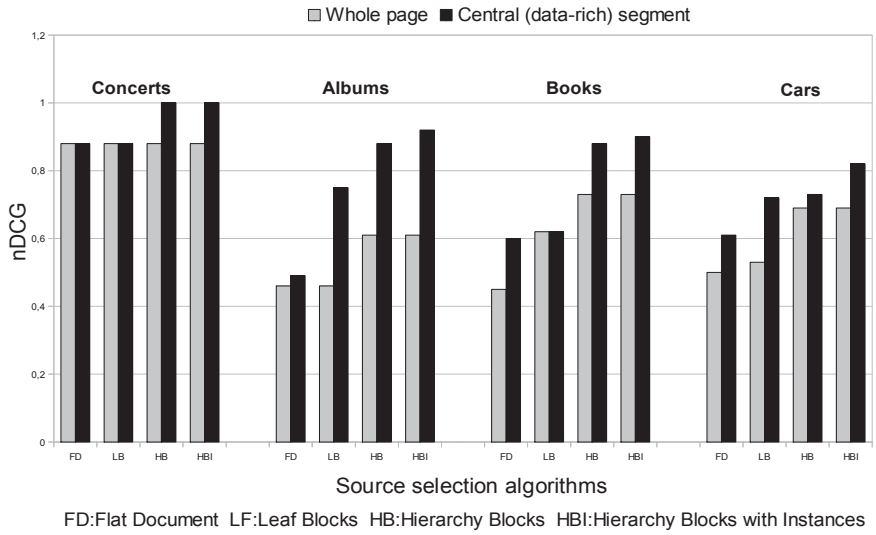


FIGURE 5.7 –Précision avec ou sans la sélection du bloc central

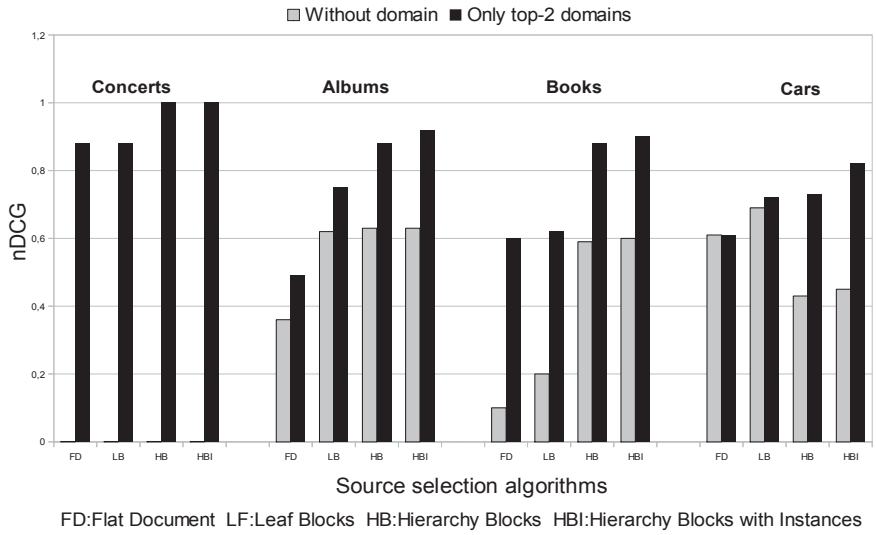


FIGURE 5.8 – Précision avec ou sans la sélection du domaine

Sélection de la meilleure classe

Nous avons également évalué l'impact de la limitation des annotations à la meilleure classe (c-à-d, choisir une seule annotation – classe dans l'ontologie – pour une instance dans le texte) sur la précision de la sélection. Les résultats sont donnés dans la Figure 5.9. Ils indiquent que ce choix peut orienter la sélection de manière efficace. Des gains en précision ont été relevés pour les quatre algorithmes et sur tous les jeux de données

considérées.

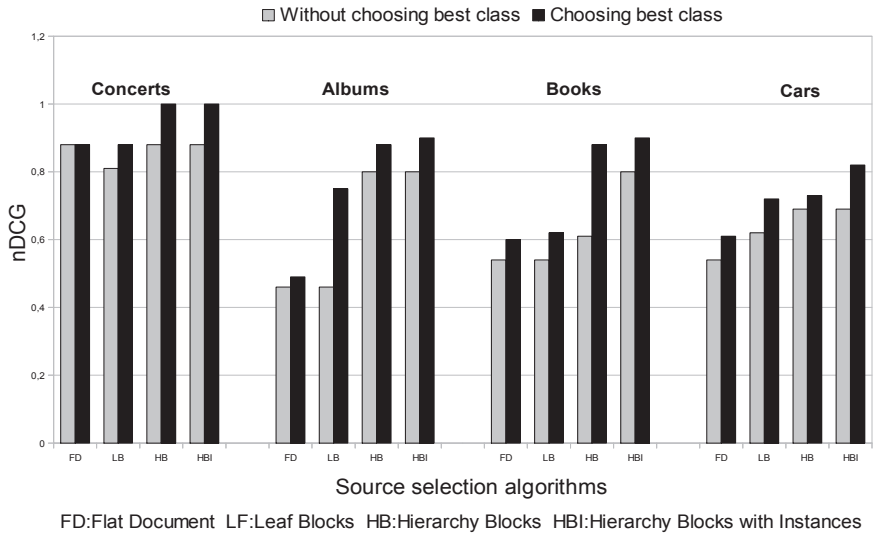


FIGURE 5.9 – Précision avec ou sans le choix de la meilleure classe

5.6 Conclusion

Nous avons décrit dans ce chapitre le problème du traitement, de l'indexation et de la sélection de sources Web pertinentes pour l'inférence de wrappers et pour l'extraction d'information. L'approche proposée est complètement automatique où, étant donnée une description intentionnelle des objets recherchés, les sources les plus pertinentes et les plus riches en données sont sélectionnées à partir d'un important répertoire de sources récupérées (crawled) du Web.

Notre approche est construite sur des techniques de recherche d'information guidée par des ontologies, ce qui nous permet d'évaluer le degré de pertinence de chaque source par rapport à la requête de l'utilisateur (SOD). Nous présentons dans ce chapitre, les algorithmes que nous avons implanté, aussi que les tests que nous avons réalisé pour évaluer notre approche.

Nos premier résultats réalisé sur des jeux de données pour quatre domaines (concerts, albums, livres et voitures) indiquent un niveau de précision élevé au niveau de la sélection des sources pertinentes. Des tests sur des jeux de données plus larges devraient confirmer ces résultats.

Chapitre 6

Conclusion

Nous nous sommes intéressés dans cette thèse à la recherche d'objets complexes dans le Web structuré. Plus précisément, à la sélection et à l'extraction des données structurées publiées dans des sources Web (un ensemble de pages `HTML`) par un *template* commun, souvent générées automatiquement. Après une étude approfondie de l'état de l'art et des problèmes fondamentaux que soulève ce domaine de recherche, nous avons proposé une approche d'interrogation à deux étapes qui nous semble la plus pratique et la plus efficace pour une interrogation ciblée du contenu du Web. Ces deux étapes consistent en :

1. *La spécification de la structure des objets ciblés.* L'utilisateur spécifie le type des objets (schéma et sémantique) qui doivent être récupérés à partir du Web. Des objets (instances de ce type) sont reconnus et extraits de manière automatique. Le résultat de cette étape est stocké dans une base de données structurées contenant les objets extraits.
2. *L'interrogation.* Une fois la première phase terminée, l'utilisateur peut interroger le résultat de l'extraction de la même manière qu'une base de données relationnelles. Ceci est fait dans notre système à l'aide d'une interface de type QBE.

Le système `ObjectRunner` que nous avons développé dans cette thèse s'appuie sur cette approche. Dans ce système, l'utilisateur donne une description intentionnelle (SOD) des données qu'il recherche. Chaque SOD représente un schéma de concepts, des relations et des reconnaisseurs reliés à des bases de connaissances sémantiques (dictionnaires construits à partir d'ontologies ou de corpus Web, des expressions régulières, etc.). Ce schéma joue un rôle central : il est utilisé pour identifier les sources pertinentes et pour extraire des objets structurés à partir de ces sources. L'approche proposée est complètement automatique et indépendante du domaine, elle s'applique à n'importe quelle structure, plate ou imbriquée, décrivant des objets du monde réel : le domaine et le schéma des objets structurés sont utilisés pour guider le processus de sélection et

d'extraction, dans le but d'atteindre efficacement les objets recherchés.

La première contribution de cette thèse concerne l'extraction d'information ciblée à partir de sources Web structurées (Chapitre 4). Nous avons proposé une approche d'extraction automatique guidée par la structure des objets ciblés (SOD). Notre approche exploite à la fois la structure des pages contenues dans chaque source considérée et un domaine de connaissances pour la reconnaissance des entités composant la structure recherchée dans le contenu textuel de ces pages. Nous avons évalué notre système sur cinq domaines et cinquante sources Web. Les résultats obtenus indiquent que c'est la combinaison entre le contenu structurel et la sémantique qui permet l'inférence du template d'extraction le plus précis, et par conséquent qui donne les meilleurs résultats. Les sources qui ont servi aux tests nous ont été fournies par des experts (de Mechanical Turk), comme étant les plus pertinentes pour chacun des domaines proposés. Nous n'avons donc pas choisi aléatoirement ou de manière subjective les sources testées, pour obtenir des résultats les plus objectifs possibles.

La seconde contribution de cette thèse est une technique de sélection de sources pertinentes (Chapitre 5). Notre but était d'intégrer l'inférence de wrappers dans un processus complètement automatique, où les sources pertinentes pour un SOD donné sont sélectionnées par le système au moment de la spécification de la structure des objets ciblés. La techniques proposée permet d'analyser des sources Web (par exemple, suite à un crawl) et de les indexer, en exploitant les contenus sémantiques des pages de ces sources, ainsi que leurs caractéristiques visuelles et structurelles. Un algorithme top-k est proposé pour permettre de trouver les sources les plus pertinentes pour le SOD spécifié, en exploitant l'index construit précédemment.

En résumé, les points importants qui différencient `ObjectRunner` des systèmes de recherche d'information existants sont :

- *Une recherche ciblée de données complexes.* La principale originalité de notre système est qu'il permet une recherche ciblée de données complexes, en se basant sur une description intentionnelle fournie au préalable par l'utilisateur pour guider le processus d'extraction.
- *La combinaison du contenu structurel et sémantique des pages.* La qualité d'extraction est améliorée grâce à la combinaison entre les informations structurelles et les annotations sémantiques des pages. Plus précisément, cette combinaison permet de distinguer de façon plus fine les rôles des tokens et, par conséquent éviter de mélanger des instances de types différents dans le même attribut du résultat final.
- *une approche rapide, avec une bonne précision et un taux d'extraction satisfaisant.* `ObjectRunner` privilégie la flexibilité et permet une bonne précision des résultats et un taux d'extraction satisfaisant. En effet, l'utilisation de la sémantique pour

guider le processus d'extraction a de multiples avantages : (i) la qualité des données extraites est améliorée, (ii) des traitements inutiles sont évités (ceux des données contenues dans les pages mais qui ne correspondent pas aux objets recherchés), (iii) le taux d'extraction est amélioré en permettant de reconnaitre et d'extraire correctement des données que les approches antérieures étaient incapable de traiter, et (iv) un coût de traitement optimisé en permettant l'arrêt du processus de construction de wrappers si la collection de pages s'avère non pertinente pour l'extraction (ne contient pas les données correspondant aux objets recherchés).

6.1 Perspectives

Outre les tâches de sélection et d'extraction, d'autres problématiques liées à la recherche d'informations complexes, n'ont pas été traitées dans cette thèse. En particulier : (i) notre système pourrait être étendu pour permettre la recherche de sources Web structurées de façon non supervisée (au niveau du crawler même), (ii) améliorer l'efficacité et la précision du système sur certains aspects (faciliter la spécification du SOD, parallélisation des modules, tester des relations plus complexes).

Découvertes de sources structurées. Dans certains scénarios d'application du monde réel, la sélection de sources ne peut pas se faire au préalable *(offline)*. En d'autres termes, nous devons être capable de découvrir automatiquement des sites Web structurés au moment du crawl *(online)*. Cela reste un défi en soi. Une direction possible serait de faire une collecte/crawl sur le Web générique, sans cibler un domaine précis. Une autre direction qui suit la logique de ce que nous avons fait pour la sélection de sources serait de guider le crawl en utilisant la description des données ciblées (SOD). Nous avons proposé dans la sélection de sources une approche qui exploite : (i) les métadonnées des pages pour la classification des sources en domaine, et (ii) les caractéristiques visuelles des pages pour la découverte de la structure hiérarchique commune à toutes les pages d'une même source. Nous pensons qu'une approche dans ce style, combinée aux algorithmes de clustering de pages Web existants [CMM05], pourrait être efficace pour identifier les sites structurés, i.e., qui publient des pages construites selon un template commun.

Plus d'interaction avec YAGO et d'autres bases de connaissances.

– *Autres moyens pour définir/inférer les SODs.* Dans notre système, l'utilisateur construit son SOD en spécifiant les différents concepts composant les objets recherchés. Nous pensons que cette spécification pourrait se faire de façon interactive, en précisant d'abord des instances de ces concepts. Le système pourrait choisir dans

l'ontologie, sur la base des instances fournies, les concepts les plus appropriés pour ces instances (à la Google set).

– *Enrichissement de YAGO.* Les nouvelles instances découvertes durant la phase d'extraction peuvent être utilisées pour enrichir automatiquement l'ontologie YAGO. Ceci pourrait être complémentaire utilisées pour peupler YAGO (SOFIE [SSW09]). Des valeurs de confiance peuvent être également attribuées aux instances avant de les ajouter, en se basant sur le résultat d'extraction et les occurrences de ses instances dans le Web. Cependant, il est nécessaire de résoudre le problème de la de-duplication avant l'ajout des instances.

Exploitation de la structure du SOD lors de la reconnaissance des entités. Pour la tâche de sélection, nous avons testé notre approche en interprétant chaque SOD comme une structure plate (ensemble de types d'entités simples). Nous voulons étendre cette approche pour exploiter une structure d'arbre de concepts, pour mieux sélectionner les sources les plus pertinentes. Une direction envisageable est d'utiliser une approche *bottom-up*, qui commence par les types qui se retrouvent au niveau inférieur du SOD, et qui tient compte de la distance dans les pages qui sépare les instances des différentes entités composant le SOD. Par exemple, pour un SOD `concert` composé d'un tuple $[artist, date, location]$, sachant que *location* est un tuple composé de $[address, theater]$. Intuitivement, les entités *addrees* et *theater* doivent se retrouver à proximité l'une de l'autre dans les pages.

Parallélisation. Notre technique d'inférence de wrappers basée sur les classes d'équivalence peut être facilement parallélisée, et ainsi réduire le temps nécessaire pour l'inférence des wrappers. Nous étudions actuellement la mise en œuvre de nos algorithmes sur une architecture distribuée.

Annexe A

Démonstration du système

Cette annexe décrit de manière générale comment le système fonctionne en pratique. Cette démonstration se focalise sur les outils utilisés pour la sélection de sources, l'extraction et l'interrogation. Nous avons conçu une interface d'interrogation qui permet aux utilisateurs (i) de définir les SODs, et (ii) d'interroger les données extraites.

A.1 Extraction de données structurées

La Figure A.1 présente une capture d'écran de l'interface d'extraction. D'abord, comme illustré sur cette capture, l'utilisateur sera en mesure de définir son SOD. Pour cela, il peut soit utiliser les SODs déjà spécifiés ou les types existants, ou décrire de nouveaux types en spécifiant les moyens pour les reconnaitre (e.x, en utilisant l'ontologie YAGO). Dans cet exemple, l'utilisateur a spécifié un SOD *concert*, en le composant de trois types d'entités simples *Artist Name*, *Date* et *Address*. Il a utilisé des reconnsaisseurs prédéfinis (Regular Expression), "Date" et "Location" pour les dates et les adresses respectivement. La multiplié est fixée à "1" pour ces deux types. Pour le nom d'artiste, il a utilisé le reconnsaisseur *isInstanceOf* associé à un dictionnaire de valeurs (instances) reliées à la classe *Artist*, en utilisant le Web Hearst (les valeurs récupérées à partir du Web en utilisant les patterns de Hearst). La multiplicité est fixée à "+" pour ce type.

Une fois le SOD défini, le système sélectionne automatiquement une liste de sources structurées pertinentes au SOD spécifié (la sélection de sources n'a pas encore été entièrement intégrée dans le système, nous avons fixé le nombre de sources sélectionnées à 2 dans cette démonstration). Finalement, l'utilisateur choisit la source à partir de laquelle il souhaite extraire. Le système construit automatiquement un wrapper pour la source sélectionnée, infère le template en se basant sur la description spécifiée (SOD) et extrait les objets contenus dans cette source.

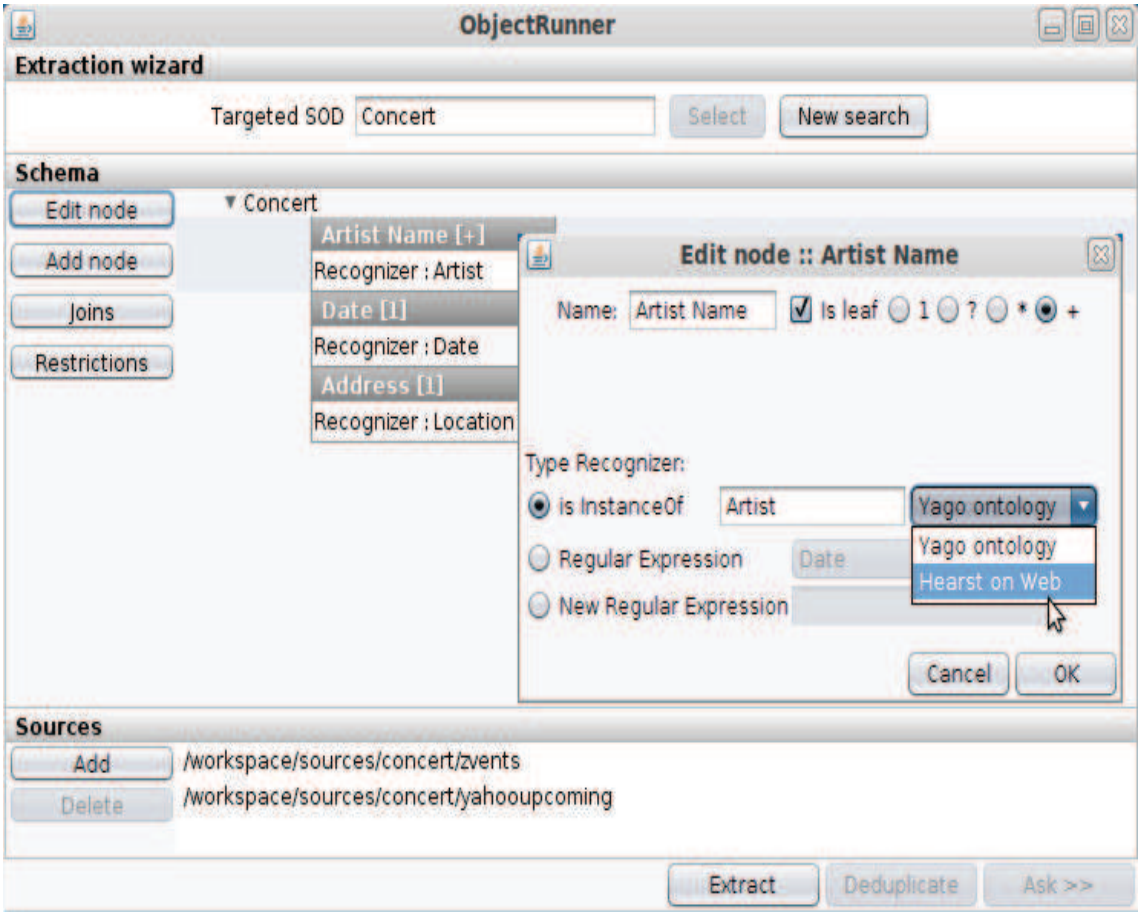


FIGURE A.1 – Interface d'extraction

A.2 Interrogation des données extraites

Dans l'interface d'interrogation (Figure A.2), l'utilisateur peut choisir quels SODs seront utilisés pour l'interrogation des données structurées extraites auparavant. Ainsi, une interface dans le style de QBE (Query By Example) lui permet de spécifier des valeurs de restrictions, de jointures sur les SODs et des restrictions de mots-clés se référant aux pages sources contenant les objets. Les sources qui doivent être interrogées peuvent être également choisies dans cette étape. Dans cet exemple, l'utilisateur a utilisé deux SODs pour sa requête, un SOD *concert* composé d'un tuple $[ArtistName, Date, Address]$ et un SOD *album* composé d'un tuple $[Title, ArtistName, Price]$. Il peut également choisir les sources à partir desquelles il veut trouver les objets correspondant à sa requête. Il

peut même spécifier des restrictions de mots-clés sur les objets recherchés. Par exemple, trouver les objets concerts et albums d'un artist *X*. Une fois sa requête définie, le système recherche les objets correspondant à sa requête.

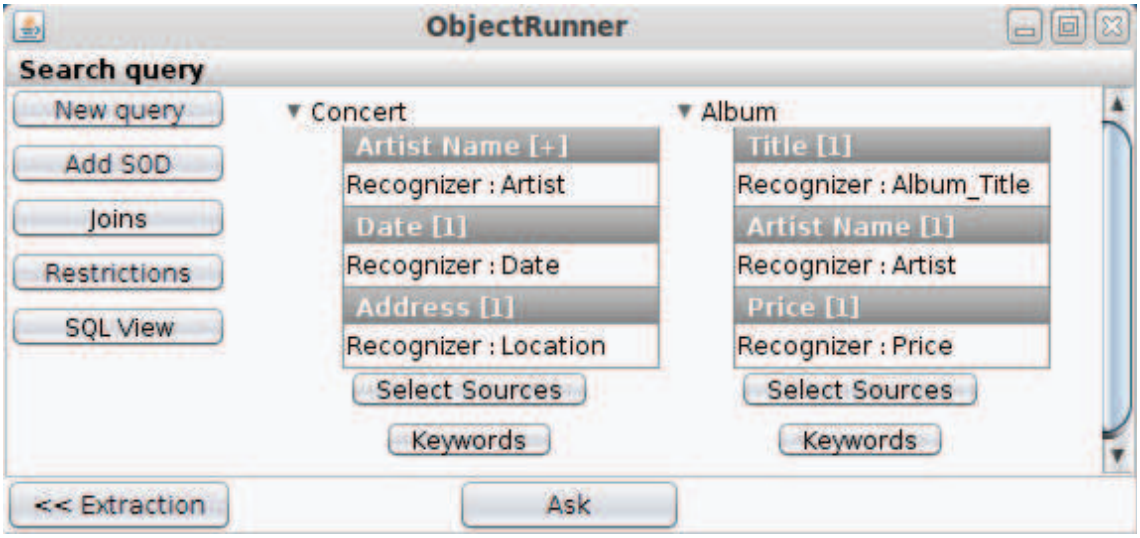


FIGURE A.2 – Interface d'interrogation

Annexe B

Lexique anglais-français

Cette annexe conclut un lexique anglais-français des termes et expressions techniques utilisés dans cette thèse.

binding engine moteur de liaison
bootsraping amorçage
bottom-up approach approche ascendante
classification catégorisation
clustering classification
data-centric centré sur les données
deep Web Web profond
domain knowledge domaine de connaissances
focused crawling exploration guidée
hidden Web Web caché
HTML form formulaires HTML
isInstanceOf est une instance de
join jointure
label étiquette
nested structure structure imbriquée
pattern motif
pattern matching correspondance de motifs
query requête
ranking ordonnancement
record enregistrement
structured object description description d'une structure des objets
string matching correspondance de chaines de caractères

tag balise
template modèle
(to)crawl the Web explorer le Web
token mot ou balise HTML
tree matching correspondance d'arbres
tuple n-uplet
wrapper extracteur

Publications

Articles dans des conférences internationales

[1] T. Abdessalem, B. Cautis, and N. Derouiche. Objectrunner : lightweight, targeted extraction and querying of structured web data. *In Proc. 36th International Conference on Very Large Data Bases (VLDB)*, volume 3, pages 1585-1588, Singapore, September 2010.

[2] N. Derouiche, B. Cautis, and T. Abdessalem. Automatic extraction of structured web data with domain knowledge. *In Proc. 28th IEEE International Conference on Data Engineering (ICDE)*, Washington, DC, USA, April 2012.

Articles dans des conférences nationales

[3] T. Abdessalem, B. Cautis, and N. Derouiche. Lightweight, targeted extraction of structured web data. In *Proc. 26èmes journées Bases de Données Avancées (BDA)*, France, October 2010.

Articles en cours

[4] N. Derouiche, B. Cautis, T. Abdessalem and A. Goel. A Semantic Indexing-based Approach to Top-k Retrieval of Structured Web Sources.

Bibliographie

[ABK+07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia : A nucleus for a web of open data. In *ISWC/ASWC*, pages 722–735, 2007.

[AG00] Eugene Agichtein and Luis Gravano. Snowball : extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*, DL '00, pages 85–94, New York, NY, USA, 2000. ACM.

[AG03] Eugene Agichtein and Luis Gravano. Querying text databases for efficient information extraction. In *ICDE*, pages 113–124, 2003.

[AGM03] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 337–348, New York, NY, USA, 2003. ACM.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[BCMP08] Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Flint : Google-basing the web. In *Proceedings of the 11th international conference on Extending database technology : Advances in database technology*, EDBT '08, pages 720–724, New York, NY, USA, 2008. ACM.

[BCS+07] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the web. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 2670–2676, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[BDM11] Lorenzo Blanco, Nilesh Dalvi, and Ashwin Machanavajjhala. Highly efficient algorithms for structural clustering of large websites. In *Proceedings of the*

20th international conference on World wide web, WWW '11, pages 437–446, New York, NY, USA, 2011. ACM.

[BEP+08] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase : a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1247–1250, New York, NY, USA, 2008. ACM.

[BHBLH11] Christian Bizer, Tom Heath, Tim Berners-Lee, and Michael Hausenblas. 4th linked data on the web workshop (ldow2011). In *WWW*, pages 303–304, 2011.

[BHIBL08] Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. Linked data on the web (ldow2008). In *WWW*, pages 1265–1266, 2008.

[Biz10] Christian Bizer. The web of linked data : a global public dataspace on the web : Webdb 2010 keynote. In *Procceedings of the 13th International Workshop on the Web and Databases*, WebDB, 2010.

[BK11] Raju Balakrishnan and Subbarao Kambhampati. Sourcerank : relevance and trust assessment for deep web sources based on inter-source agreement. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 227–236, New York, NY, USA, 2011. ACM.

[Bri99] Sergey Brin. Extracting patterns and relations from the world wide web. In *Selected papers from the International Workshop on The World Wide Web and Databases*, pages 172–183, London, UK, 1999. Springer-Verlag.

[CCZ07] Shui-Lung Chuang, Kevin Chen-Chuan Chang, and ChengXiang Zhai. Context-aware wrapping : synchronized data extraction. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 699–710. VLDB Endowment, 2007.

[CDSE05] Michael J. Cafarella, Doug Downey, Stephen Soderland, and Oren Etzioni. Knowitnow : fast, scalable information extraction from the web. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 563–570, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[CGMP98] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through url ordering. In *Proceedings of the seventh international conference on World Wide Web 7*, WWW7, pages 161–172, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[CHW+08] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables : exploring the power of tables on the web. *Proc. VLDB Endow.*, 1 :538–549, August 2008.

[CHZ+08] Michael J. Cafarella, Alon Y. Halevy, Yang Zhang, Daisy Zhe Wang, and Eugene Wu. Uncovering the relational web. In *WebDB*, 2008.

[CK04] Chia-Hui Chang and Shih-Chien Kuo. Olera : Semisupervised web-data extraction with visual support. *IEEE Intelligent Systems*, 19 :56–64, November 2004.

[CL01] Chia-Hui Chang and Shao-Chen Lui. Iepad : information extraction based on pattern discovery. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 681–688, New York, NY, USA, 2001. ACM.

[CM04] W. Cohen and A. McCallum. Information extraction and integration : an overview. In *KDD*, 2004.

[CMBT02] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE : A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, PA, USA*, 2002.

[CMM01] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner : Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 109–118, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[CMM05] Valter Crescenzi, Paolo Merialdo, and Paolo Missier. Clustering web pages based on their structure. *Data Knowl. Eng.*, 54 :279–299, September 2005.

[CP10] Liang Jeff Chen and Yannis Papakonstantinou. Supporting top-k keyword search in xml databases. In *ICDE*, pages 689–700, 2010.

[CvdBD99] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling : a new approach to topic-specific web resource discovery. In *Proceedings of the eighth international conference on World Wide Web*, WWW '99, pages 1623–1640, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[CYWM03] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Extracting content structure for web pages based on visual representation. In *Proceedings of the 5th Asia-Pacific web conference on Web technologies and applications*, APWeb'03, pages 406–417, Berlin, Heidelberg, 2003. Springer-Verlag.

[DCL+00] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 527–534, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[FCL+11] Miriam Fernández, Iván Cantador, Vanesa López, David Vallet, Pablo Castells, and Enrico Motta. Semantically enhanced information retrieval : An ontology-based approach. *Web Semant.*, 9 :434–452, December 2011.

[Fel98] Christiane Fellbaum, editor. *WordNet : An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, illustrated edition edition, May 1998.

[FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4) :614–656, 2003.

[GKB+04] Georg Gottlob, Christoph Koch, Robert Baumgartner, Marcus Herzog, and Sergio Flesca. The Lixto data extraction project - back and forth between theory and practice. In *PODS*, 2004.

[HB02] Ajay Hemnani and Stéphane Bressan. Extracting information from semi-structured web documents. In *OOIS Workshops*, 2002.

[HD98] Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Inf. Syst.*, 1998.

[Hea92] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics - Volume 2*, COLING '92, pages 539–545, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[HK05] Andrew Hogue and David R. Karger. Thresher : automating the unwrapping of semantic content from the world wide web. In *WWW*, 2005.

[HPZC07] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. Accessing the deep web. *Commun. ACM*, 50 :94–101, May 2007.

[HSB+11] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, Edwin Lewis-Kelham, Gerard de Melo, and Gerhard Weikum. Yago2 : exploring and querying world knowledge in time, space, context, and many languages. In *WWW (Companion Volume)*, 2011.

[Htm] HtmlCleaner. http ://htmlcleaner.sourceforge.net/.

[IAJG06] Panagiotis G. Ipeirotis, Eugene Agichtein, Pranay Jain, and Luis Gravano. To search or to crawl ? : towards a query optimizer for text-centric tasks. In *SIGMOD Conference*, pages 265–276, 2006.

[IG04] Panagiotis G. Ipeirotis and Luis Gravano. When one sample is not enough : Improving text database selection using shrinkage. In *SIGMOD*, 2004.

[JK00] Kalervo Järvelin and Jaana Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *SIGIR*, 2000.

[JL10] Nitin Jindal and Bing Liu. A generalized tree matching algorithm considering nested lists for web data extraction. In *SDM*, 2010.

[JTi] JTidy. http ://jtidy.sourceforge.net.

[KS06] Mohammed Kayed and Khaled F. Shaalan. A survey of web information extraction systems. *IEEE TKDE*, 2006.

[Kus97] Nicholas Kushmerick. *Wrapper induction for information extraction*. PhD thesis, 1997. AAI9819266.

[LGZ03] Bing Liu, Robert Grossman, and Yanhong Zhai. Mining data records in web pages. In *KDD*, 2003.

[Liu06] Bing Liu. *Web Data Mining : Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[LMM10] Wei Liu, Xiaofeng Meng, and Weiyi Meng. Vide : A vision-based approach for deep web data extraction. *IEEE Trans. on Knowl. and Data Eng.*, 2010.

[LN04] Zhao Li and Wee Keong Ng. WICCAP : From semi-structured data to structured data. In *ECBS*, 2004.

[LSC10] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *PVLDB*, 3(1), 2010.

[LSS+10] Hady W. Lauw1, Ralf Schenkel, Fabian Suchanek, Martin Theobald, and GerhardWeikum. Harvesting knowledge from web data and text. In *CIKM*, 2010.

[MAAH09] Jayant Madhavan, Loredana Afanasiev, Lyublena Antova, and Alon Halevy. Harnessing the deep web : Present and future. In *4th Biennial Conference on Innovative Data Systems Research (CIDR)*, January 2009.

[MC08a] Luke McDowell and Michael J. Cafarella. Ontology-driven, unsupervised instance population. *J. Web Sem.*, 2008.

[MC08b] Luke K. McDowell and Michael Cafarella. Ontology-driven, unsupervised instance population. *Web Semant.*, 2008.

[MMK99] Ion Muslea, Steven Minton, and Craig A. Knoblock. A hierarchical approach to wrapper induction. In *Agents*, 1999.

[Mor68] Donald R. Morrison. Patricia-practical algorithm to retrieve information coded in alphanumeric. *J. ACM*, 15 :514–534, October 1968.

[MPS04] Filippo Menczer, Gautam Pant, and Padmini Srinivasan. Topical web crawlers : Evaluating adaptive algorithms. *ACM Trans. Internet Technol.*, 4 :378–419, November 2004.

[RGM01] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 129–138, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[RPA+02] Juan Raposo, Alberto Pan, Manuel Álvarez, Justo Hidalgo, and Ángel Vi na. The wargo system : Semi-automatic wrapper generation in presence of complex data access modes. In *DEXA*, 2002.

[Sar08] Sunita Sarawagi. Information extraction. *Foundations and Trends in Databases*, 2008.

[SKW07] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago : a core of semantic knowledge. In *WWW*, 2007.

[SL05] Kai Simon and Georg Lausen. Viper : augmenting automatic information extraction with visual perceptions. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM, 2005.

[SMM+08] Pierre Senellart, Avin Mittal, Daniel Muschick, Rémi Gilleron, and Marc Tommasi. Automatic wrapper induction from hidden-web sources with domain knowledge. In *WIDM*, 2008.

[Sod99] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 1999.

[SSW09] Fabian M. Suchanek, Mauro Sozio, and Gerhard Weikum. Sofie : a self-organizing framework for information extraction. In *WWW*, pages 631–640, 2009.

[SWL09] Weifeng Su, Jiying Wang, and Frederick H. Lochovsky. Ode : Ontology-assisted data extraction. *ACM Trans. Database Syst.*, 2009.

[TSW05] Martin Theobald, Ralf Schenkel, and Gerhard Weikum. Efficient and self-tuning incremental query expansion for top-k query processing. SIGIR, 2005.

[WHHK11] Wu Wentao, Li Hongsong, Wang Haixun, and Zhu Kenny. Towards a probabilistic taxonomy of many concepts. In *PVLDB*, 2011.

[WL03] Jiying Wang and Frederick H. Lochovsky. Data extraction and label assignment for web databases. In *WWW*, 2003.

[Yan91] Wuu Yang. Identifying syntactic differences between two programs. *Softw. Pract. Exper.*, 21 :739–755, June 1991.

[YG98] Roman Yangarber and Ralph Grishman. Nyu : Description of the proteus/pet system as used for muc-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998.

[ZL05] Yanhong Zhai and Bing Liu. Web data extraction based on partial tree alignment. In *WWW*, 2005.

[ZMW+05] Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay Raghavan, and Clement Yu. Fully automatic wrapper generation for search engines. In *WWW*, 2005.

[ZMY06] Hongkun Zhao, Weiyi Meng, and Clement Yu. Automatic extraction of dynamic record sections from search engine result pages. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB, 2006.

[ZNW+06] Jun Zhu, Zaiqing Nie, Ji-Rong Wen, Bo Zhang, and Wei-Ying Ma. Simultaneous record detection and attribute labeling in web data extraction. In *KDD*, 2006.