6IR : Un index paramétrable pour les requêtes ramifiées

Y. Péron

Laboratoire Valoria
Université de Bretagne Sud
Bât. Yves Coppens – B.P. 573
56017 Vannes Cedex
youen.peron@univ-ubs.fr

RÉSUMÉ. *Cet article contient une présentation de notre travail en cours de développement dans le domaine de la recherche d'informations dans des bases de documents semi-structurées. Nous cherchons à construire un système d'interrogation – dénommé 6IR pour Structure based IndeX Information Retrieval – qui fournisse une liste de documents similaires au contenu et à la structure d'une requête ramifiée. L'extraction des documents est basée sur l'identification de points communs entre leur structure et celle de la requête. Nous détaillons le processus d'indexation qui consiste à extraire des documents de la base tous les points d'accrochage exploitables dans le processus d'interrogation. Nous montrons comment parvenir à maîtriser l'explosion combinatoire de la taille de l'index en paramétrant la taille des points d'ancrage et les propriétés qui en découlent pour les documents candidats obtenus lors du processus d'interrogation.*

ABSTRACT. *This paper contains a presentation of our work in progress in the domain of information retrieval in base of semi-structured documents. We try to build a querying engine – called 6IR for Structure based IndeX Information Retrieval – which provides a list of documents similar in content and structure of a twig query. The extraction of documents is based on the identification of structure pattern. We detail the indexing process that consists of extracting all the patterns of the documents of the base useable for the process of interrogation. We show how to control the combinatorial explosion in the size of the index by setting the size of the patterns and the properties that followed on the documents obtained during the interrogation.*

MOTS-CLÉS : *recherche d'information, XML, requête ramifiée, fragmentation de document, indexation de document, recherche dans une collection*

KEYWORDS: *information retrieval, XML, twig query, document fragmentation, document indexing, search in a collection*

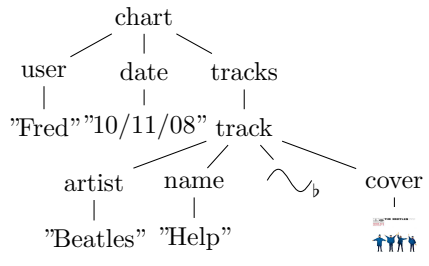
Youen Péron

1. Introduction

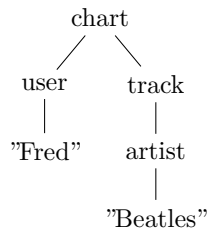
L'utilisation massive du standard XML pour la représentation de l'information implique de nouveaux problèmes liés au stockage, à l'indexation et à la recherche d'informations. Les documents XML ont une structure d'arbre (arbre DOM) qui leur permet d'embarquer du contenu non structuré et hétérogène. Par exemple l'arbre de la figure 1a regroupe des information aussi diverses qu'une date, un mot-clé, une série temporelle ou une image. Les modèles de bases de données classiques ont une structure trop rigide pour gérer l'hétérogénéité ; à l'opposé les listes inverses utilisées pour la recherche plein texte ne prennent pas en compte la structure des documents. Or il est naturel d'exprimer les requêtes, exactes ou approchées, sur les bases de documents XML avec des arbres reliant plusieurs critères de recherche. Par exemple le langage FleXPath (Amer-Yahia *et al.,* 2004) permet de définir des requête de la forme 1c qui correspond à l'arbre de la figure 1b. On qualifie ces requêtes de ramifiées *(twig query)*. Retrouver rapidement l'ensemble des documents D d'une collection $\mathcal{C}$ correspondant à une requête ramifiée R est une opération incontournable dans le processus d'interrogation des bases de documents XML.

Les auteurs de (Catania *et al.,* 2005) ont proposé une classification des systèmes de gestion des collections de documents XML selon trois types d'approches :

– **Décomposition des documents en chemins :** pour chaque feuille du document on construit le chemin qui relie cette feuille à la racine de l'arbre afin de créer une table avec comme clé la feuille et comme valeur le chemin qui lui est associé. L'approche de (Abiteboul, 1997) est très efficace pour les requêtes ne contenant qu'un seul chemin. Cependant un post-traitement est nécessaire



(a) Arbre DOM d'un document XML.



(b) Requête ramifiée.

$$//chart[.user[.contains("Fred")] \ and \ .track[.artist[.contains("Beatles")]]]$$

(c) Requête FleXPath

Figure 1. *Exemple de document* XML

si l'on veut répondre à des requêtes ramifiées comme dans (Popovici *et al.,* 2006).

– **Décomposition de la requête en jonctions :** Les approches (Bruno *et al.,* 2002) et (Jiang *et al.,* 2003) indexent les nœuds d'un arbre en fonction de leurs valeurs ou de leurs tags. Après interrogation de l'index, un algorithme vérifie que les relations de parentés concordent avec les relations de la requête. Les limitations de ces approches est de décomposé la requête ramifiée en plusieurs sous requêtes de type ancêtre–descendant.

– **Représentation séquentielle de la requête :** en utilisant une représentation séquentielle des arbres, les systèmes (Wang *et al.,* 2003) et (Rao *et al.,* 2004) utilisent des algorithmes de recherche de sous-séquences pour répondre à une requête. Une telle approche évite la décomposition de la requête en sous-requêtes et donc les jointures nécessaires à la fusion des résultats des sous-requêtes. Cependant la linéarisation des arbres engendre des faux-positifs que l'on doit filtrer.

Actuellement, on peut constater que les systèmes de résolution de requête ramifiée favorisent les relations ancêtre–descendant, que ce soit dans le choix du découpage de l'arbre, ou dans le choix de la représentation de la structure des document XML. Dans les deux premières approches citées ci dessus le contenu non-structuré sert de point d'accroche pour retrouver des réponses potentielles, puis le système vérifie la correspondance entre la structure de la requête et celle du document candidat. Ces approches basées sur l'indexation du contenu non-structuré impliquent que les documents XML contiennent des informations de même type (paragraphes, mots-clés, séries-temporelles).

Par rapport aux approches précédentes, nous cherchons un moyen pour indexer les documents XML qui permette à la fois :

– des comparaisons spécifiques à chaque type de contenu présent dans un document (image, texte, série temporelle) ;

– de mieux intégrer les relations de voisinage dans la structure.

Nous proposons dans cet article une approche qui, dans un premier temps, utilise la structure des documents comme points d'ancrage d'une requête et, dans un second temps, utilise une distance spécifique à chaque nœud pour en comparer le contenu avec celui de la requête.

Le reste de l'article est organisé de la manière suivante. Dans la section 2 nous décrivons l'ensemble de notre approche en nous focalisant sur le problème de la construction de l'index et la solution que nous proposons pour maîtriser l'explosion combinatoire qui en résulte. Dans la section 3 nous discutons de la pertinence de l'index ainsi construit et des possibilités offertes par sa paramétrisation. Nous terminons à la section 4 par un état des lieux de l'avance de nos travaux et les expérimentations en cours pour valider et enrichir notre approche.

2. L'approche 6IR

Étant donné une base de documents XML, on cherche à construire un système d'interrogation qui fournisse une liste de documents similaires au contenu et à la structure d'une requête ramifiée. L'extraction des documents candidats est basée sur l'identification de points communs entre leur structure et celle de la requête. Le processus d'indexation consiste à extraire des documents de la base tous les points d'accroches exploitables dans le processus d'interrogation. Ces points d'accroches sont des sous-arbres XML de taille très réduite (quelques nœuds). Nous détaillons dans cette section le processus d'indexation et son exploitation lors du processus d'interrogation de la base.

2.1. *Processus d'indexation de la base*

On s'intéresse dans un premier temps à évaluer N_P, le nombre total de points d'ancrage à extraire d'un ensemble de $|\mathcal{C}|$ documents d'une collection $\mathcal{C}$. Le nombre de points d'ancrage extraits d'un document D correspond au nombre de sous-arbres contenus dans l'arbre XML de ce document. Si on note $|D|$ la taille en nombre de nœuds d'un document D, $|D|_{max}$ la taille du plus grand document de la base et $\binom{n}{k}$ le coefficient binomial de k parmi n, alors le nombre total de points d'ancrage N_P de la collection $\mathcal{C}$ est borné par la relation :

$$N_P \leq \sum_{D \in \mathcal{C}} \left(\sum_{k=0}^{|D|} \binom{|D|}{k} \right) \leq |\mathcal{C}| \times 2^{|D|_{max}}$$

L'explosion combinatoire du nombre de points d'ancrage potentiels qui en résulte nous incite à fragmenter les documents pour réduire le facteur $|D|_{max}$ ainsi qu'a limiter et fixer la taille des points d'ancrages (facteur k). C'est pourquoi lors de l'indexation nous procédons en deux phases successives :

1) une phase de fragmentation qui consiste à réduire le nombre de nœuds dans l'arbre XML de chaque document ;

2) une phase d'extraction des points d'ancrage dont la taille (le nombre de nœuds) est fixée à une valeur arbitraire k.

2.1.1. *La fragmentation des documents et la table $\mathcal{T}_F$*

La fragmentation des documents consiste à extraire de l'arbre XML d'un document un ensemble de sous-arbres dont la structure est simplifiée. L'extraction est basée sur la structure des documents XML dont on suppose connaître soit les DTD soit les XML Schema : Tout document contenant une liste d'éléments provoque la génération d'une liste de fragments de documents contenant une occurrence de chaque élément. Ce remplacement est supervisé, de manière,

d'une part à limiter la multiplication du nombre de fragments, et d'autre part à privilégier certaines structures de requêtes adaptée à l'application visée.

Chaque fragment extrait d'un document est enregistré dans une table des fragments nommée $\mathcal{T}_F$. La clé d'une entrée est un identifiant composé d'un entier suffixé par le numéro de l'occurrence de l'élément dans la liste qui a servie à la fragmentation. La valeur associée à cette clé est une référence qui permet de retrouver le document dans le système de fichiers de la base XML.

2.1.2. *L'extraction des points d'ancrage et la table $\mathcal{T}_A$*

Maintenant que la structure des documents est simplifiée, on en extrait les points d'ancrage qui vont servir d'entrées dans l'index de la manière suivante.

On fixe arbitrairement la valeur de k, le nombre de nœuds de chaque point d'ancrage. Le choix de la valeur de k et ses conséquences sont discutées au paragraphe 3.

On construit de manière exhaustive l'ensemble des sous-arbres de taille k à partir de chaque fragment. Le nombre de points d'ancrage ainsi obtenu pour un fragment de taille n est borné par $\binom{n}{k}$. Ce nombre étant relativement important même pour de petites valeurs de k [1], nous optimisons l'utilisation de la mémoire en codant les arbres par des séquences de Prüfer (Prufer, 1918) comme dans les systèmes décrits dans (Rao *et al.,* 2004) et (Agarwal *et al.,* 2007).

Les points d'ancrage ainsi obtenus sont mémorisés dans une table $\mathcal{T}_A$. La clé d'une entrée est le codage de Prüfer du point d'ancrage. La valeur associée est l'identifiant du fragment (2.1.1) dont est extrait le point d'ancrage ; ou plus précisément, comme un point d'ancrage peut être présent dans plusieurs fragments, la valeur associée est l'ensemble des identifiants des fragments contenant ce point d'ancrage.

2.2. *Processus d'interrogation de la base*

On soumet au système 6IR une requête ramifiée que l'on décompose en points d'accroche suivant un algorithme identique à celui utilisé pour l'indexation (cf paragraphe 2.1.2) et avec la même valeur du paramètre k (taille des points d'accroche). Chaque point d'accroche issu de la décomposition est codé sous la forme d'une séquence de Prüfer et sert de point d'entrée dans la table $\mathcal{T}_A$. La lecture de la valeur associée à cette entrée dans la table $\mathcal{T}_A$ est un ensemble d'identifiants de fragments. L'intersection de tous les ensembles obtenus pour tous les points d'accroche de la requête fournit (via la table des fragments $\mathcal{T}_F$) la liste des documents candidats, similaires du point de vue de *leur structure* à celle de la requête.

1. Comme indiqué dans le paragraphe 3.3, les valeurs de k considérées vont de 1 à 3

Il reste ensuite à comparer *le contenu* de la requête avec celui des documents candidats. Nous opérons en deux phases successives.

1) On aligne la structure des arbres XML des documents candidats avec celle de l'arbre de la requête. La phase d'alignement est un problème connu ; on utilise la solution donnée dans (Bruno *et al.,* 2002).

2) On compare les données non structurées de la requête qui sont alignées avec celles des documents candidats.

La phase de comparaison du contenu peut ainsi dépendre du type d'information enregistré dans un nœud. Par exemple, s'il s'agit de mots clés, on peut utiliser un calcul de distance basé sur l'algorithme de Levenshtein (Levenshtein, 1966). Si le contenu est formé de paragraphes, une solution basée sur un TF-IDF sera mieux adaptée. De manière générale, notre approche s'adapte à tout type de données non structurées présentes dans les documents XML ; il faut disposer d'un algorithme de calcul de distance sur le type considéré.

3. Discussion

Dans la partie 2 on a expliqué comment extraire un ensemble de documents structurellement proches de la requête. On discute dans cette partie de la pertinence des documents extraits. Cette discussion va dans un premier temps s'intéresser aux gains apportés par l'index sur l'espace de recherche. Dans un deuxième temps nous allons discuter des conséquences de la fragmentation. Enfin nous étudierons l'influence du paramètre k (taille des points d'accroches) sur la qualité du voisinage structurelle considéré.

3.1. *Pertinence de l'index*

On peut se demander si les résultats d'un index basé que sur la structure réduit de façon conséquente l'espace de recherche. En effet si les documents ont tous la même DTD, alors les nœuds qui composent les arbres XML auront un nombre de labels très limité avec des relations identiques (celles guidées par la DTD). Les combinaisons possibles pour extraire les points d'ancrages seront elles aussi limitées. La probabilité qu'un fragment contienne un point d'accroche sera par conséquent très élevé, et l'espace de recherche sera peu réduit.

Pour répondre à ce problème, on propose de diversifier les éléments de structures en rajoutant des signatures issues du contenu non-structuré dans la structure. Par exemple dans l'arbre de la figure 1a on peut utiliser la signature « première lettre du mot-clé » pour rajouter un nœud ayant comme label « F » entre le nœud « user » et le contenu « Fred ».

En contrepartie la taille des fragments est augmentée et la recherche sur le contenu non-structuré est conditionnée par l'égalité entre la signature des données de la requête et du fragment. Le choix d'une fonction de signature peut s'avérer imposible pour des données plus importantes qu'un simple mot clé, comme un paragraphe ou une série temporelle.

3.2. *Conséquence de la fragmentation*

En découpant les documents en fragments de document, on supprime les liens entre les données contenus dans chaque fragment. De même que les approches basées sur la décomposition des arbres XML en chemins considèrent chaque chemin indépendant, l'approche 6IR considère que les fragments ne sont pas corrélés.

Cette hypothèse se justifie par le fait que la fragmentation, guidée par les DTD est supervisée par un administrateur pour répondre au mieux à l'application visée en se basant sur la sémantique des balises XML.

3.3. *Influence de la taille des points d'accroches*

Dans cette dernière partie de la discussion on s'intéresse aux conséquences du choix du facteur k. La qualité des fragments obtenus avec l'approche 6IR augmente avec la taille des points d'accroches.

En effet si on choisit des points d'accroche de taille $k = 1$ les points d'accroche sont des nœuds. Tous les nœuds de la requête ramifié sont contenus dans les fragments retournés par l'index mais la structure qui les relie peut être totalement différente. En prenant la valeur $k = 2$ on conserve le lien de parenté entre les nœuds cependant l'ordre des nœuds n'est pas conservé et il est possible de trouver des cas de faux positifs (les mêmes cas que dans le système VIST (Wang *et al.,* 2003). À partir d'une taille $k = 3$ on conserve la structure avec les triplets de la forme (fils,fils,parent) l'ordre des nœuds est respecté et les faux positifs de VIST sont éliminés.

L'inconvénient majeur d'augmenter la taille des points d'ancrage est de multiplier leur nombre et d'augmenter sensiblement la table $\mathcal{T}_A$.

4. Conclusion

Nous avons présenté dans cet article une approche qui indexe les documents XML en fonction d'éléments de petite taille de leur structure. La nouveauté dans cette approche est de ne pas indexer le contenu mais la structure ce qui permet de comparer les données non structurées et hétérogène avec une distance adéquate.

Nous avons developpé un prototype de l'application 6IR qui valide les mécanismes de fragmentation, d'extraction de points d'accroche, d'utilisation de signature des mots clés et l'interrogation de l'index. Cependant il n'a pas encore été testé sur une grande masse de documents contenant plusieurs types de contenu. Actuellement seule la distance de Levenshtein est utilisée.

Nous avons vu dans la partie discussion que l'indexation de la structure prenait tout son sens pour une valeur de k (taille des points d'accroches) égale à 3. Il nous parait intéressant d'appliquer cette méthode pour l'interrogation d'une collection de triplets RDF qui sont de plus en plus utilisés pour diffuser les informations du web sémantique.

5. Bibliographie

Abiteboul S., *Querying Semi-Structured Data*, Springer, 1997.

Agarwal N., Oliveras M. G., Chen Y., « Approximate Structural Matching over Ordered XML Documents », *Database Engineering and Applications Symposium, 2007. IDEAS 2007. 11th International*, p. 54-62, 2007.

Amer-Yahia S., Lakshmanan L. V. S., Pandit S., « FleXPath : flexible structure and full-text querying for XML », *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, ACM New York, NY, USA, p. 83-94, 2004.

Bruno N., Koudas N., Srivastava D., « Holistic twig joins : optimal XML pattern matching », *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, ACM Press New York, NY, USA, p. 310-321, 2002.

Catania B., Maddalena A., Vakali A., « XML document indexes : a classification », *Internet Computing, IEEE*, vol. 9, n° 5, p. 64-71, 2005.

Jiang H., Wang W., Lu H., Yu J. X., « Holistic twig joins on indexed XML documents », *VLDB '2003 : Proceedings of the 29th international conference on Very large data bases*, VLDB Endowment, p. 273-284, 2003.

Levenshtein V. I., « Binary Codes Capable of Correcting Deletions, Insertions and Reversals », *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

Popovici E., Menier G., Marteau P., « Recherche approchée d'information dans une base de documents semi-structurés », *3ème Conférence en Recherche d'Information et Application*, p. 53-64, 2006.

Prufer H., « Neuer Beweis eines Satzesuber Permutationen », *Archiv fur Mathematik und Physik*, vol. 27, p. 142-144, 1918.

Rao P., Moon B., « PRIX : indexing and querying XML using prufer sequences », *Data Engineering, 2004. Proceedings. 20th International Conference on Data Engineering (ICDE'04)*, vol. 0, p. 288-299, 2004.

Wang H., Park S., Fan W., Yu P. S., « ViST : a dynamic index method for querying XML data by tree structures », *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, ACM Press New York, NY, USA, p. 110-121, 2003.