TIGER : interrogation d'une table relationnelle par extension de critères

Yoann Pitarch[1], Dominique Laurent[2],
Pascal Poncelet[1] et Nicolas Spyratos[3]
(1) LIRMM - CNRS - Univ Montpellier 2, 34392 Montpellier Cedex 5
{pitarch,poncelet}@lirmm.fr
(2) ETIS - CNRS - Univ Cergy-Pontoise, F-95000 Cergy-Pontoise
dominique.laurent@u-cergy.fr
(3) LRI CNRS, Univ Paris 11, F-91405 Orsay
spyratos@lri.fr

Abstract

Aujourd'hui, le web offre une sorte de place publique où les sites marchands prolifèrent. Dans un contexte extrêmement concurrentiel, il est capital pour un site de retenir un utilisateur lorsqu'il effectue une recherche. Parmi les stratégies possibles pour atteindre cet objectif, offrir à l'utilisateur un outil de recherche flexible est un axe intéressant à étudier. A l'heure actuelle, les formulaires présents sur les sites marchands interprètent de manière stricte les critères saisis. Proposer une gestion plus flexible de ces critères permettrait de diminuer le nombre de requêtes formulées et améliorerait donc la qualité de la recherche. Nous proposons, TIGER,une méthode pour une prise en compte flexible des critères utilisateurs. Ainsi, si les critères sont trop contraignants, nous proposons à l'utilisateur une combinaison d'extensions de ses critères qui maximise la qualité du résultat tout en minimisant la distance à la requête d'origine. Les expérimentations menées permettent de valider la qualité des résultats obtenus.

1 Introduction

Aujourd'hui, le web offre une sorte de place publique où des entreprises en ligne prolifèrent. Dans cet espace virtuel, la concurrence entre les sites marchands est accrue et se démarquer ou fidéliser sa clientèle est devenu indispensable pour perdurer. Un objectif primordial à atteindre est le maintien de l'utilisateur

sur le site lorsqu'il effectue une recherche. Outre les systèmes de recommandation [AT05, Bur02, MSR04] prédisant les besoins d'un utilisateur, un aspect tout aussi important est de faciliter la recherche de produits. Pour cela, les sites marchands proposent des formulaires aux utilisateurs. Même si ces derniers offrent un nombre croissant de critères à renseigner, leur interprétation par le moteur de recherche demeure stricte. En effet, un produit ne sera retourné que s'il valide tous les critères. Considérons, par exemple, le cas d'un utilisateur à la recherche d'une voiture d'occasion sur un site marchand. Deux comportements sont envisageables. Un premier est d'égrainer manuellement la liste des voitures en vente. Malgré l'exhaustivité d'une telle solution, celle-ci n'est pas viable car trop consommatrice en temps. Un autre comportement plus réaliste est que l'utilisateur renseigne une liste de critères via un formulaire de recherche. Puisqu'une voiture ne sera affichée que si elle valide tous les critères, la liste fournie sera restreinte. S'il n'est pas satisfait (pas de réponse ou peu de réponses intéressantes), il devra alors étendre certains critères en soumettant une nouvelle requête moins précise. Dans l'hypothèse optimiste où l'utilisateur restera sur le site visité, ce phénomène d'extension de critères sera réitéré jusqu'à arriver aux résultats recherchés. Il est toutefois peu probable que l'utilisateur poursuive sa recherche sur le même site et s'orientera alors vers un site marchand concurrent. Un tel comportement serait préjudiciable au site marchand et pourrait être évité en interprétant les critères de l'utilisateur de façon moins stricte.

Dans cet article, nous proposons, TIGER, une solution pour rendre les formulaires de recherche plus flexibles et faciliter ainsi la recherche de produits intéressants (par l'utilisateur) dans une grande base de données. Notre objectif est d'améliorer l'expérience de navigation de l'utilisateur en réduisant le nombre de requêtes formulées pour parvenir à une réponse satisfaisante. Pour cela, nous proposons de réécrire la requête initiale de l'utilisateur en étendant le moins possible certains critères pour maximiser la qualité des résultats renvoyés. Pour évaluer la qualité de la réponse fournie par le moteur de recherche, nous nous basons sur le nombre de produits qui valident certains critères *fixes* spécifiés par l'utilisateur. Par exemple, supposons qu'un utilisateur formule la requête suivante : "*Quelles sont les Clio blanches de moins de 10 000 kms qui sont en vente à moins de 5000 euros?* ". Il est possible que l'utilisateur ne veuille pas céder sur le prix même si cette requête ne renvoie aucun résultat. Les critères non fixes (dits *extensibles*) sont alors étendus pour augmenter significativement le nombre de Clio blanches de moins de 5000 euros retournées. Nous montrons également comment prendre en compte les préférences utilisateurs (explicites ou apprises) pour étendre certains critères portant sur des attributs symboliques. Une fonction de score considérant aussi bien la dis-

tance entre une combinaison et la requête initiale que le gain engendré par cette même combinaison est introduite. La combinaison d'extensions dont le score est maximal est alors retenue. Nous validons expérimentalement notre approche en vérifiant que la méthode proposée peut être intégrée à un site marchand (*i.e.*, le temps de calcul de la meilleure combinaison est faible) et que la qualité des réponses est effectivement accrue.

La suite de cet article est organisée de la façon suivante. La section 2 présente un panorama des approches proposant la réécriture de requêtes. Dans la section 3, un cas d'étude illustre la problématique et la méthode proposée. Les concepts associés à cette méthode sont définies dans la section 4. L'approche TIGER est développée dans la section 5. La section 6 présente et discute les résultats de nos expérimentations. Enfin, la dernière section conclut ce travail en dressant quelques perspectives.

2 Etat de l'art

Notre travail aborde la problématique de la réécriture et de la relaxation de requêtes dans les bases de données. Dans la suite, nous verrons que cette problématique a été introduite au début des années 90 mais ne s'est développée que récemment à cause de l'émergence du web.

Dans [GGM92], Gaasterland introduit la problématique de la relaxation de requêtes dans des bases de données déductives grâce à l'utilisation de règles logiques pour indiquer les relaxations autorisées. Dans [CYC+96], Chu présente, CoBase, un système de relaxation de requêtes dans des bases relationnelles. Pour cela, il introduit le concept de "type-abstraction ontology" pour représenter la hiérarchie associée à chaque attribut. Les résultats obtenus sont alors sémantiquement proches de la requête initiale. Malgré de nombreuses idées intéressantes, les relaxations possibles doivent être explicitement définies a priori et ne peuvent donc pas être ajustées. Par exemple, s'il est spécifié qu'un attribut *Kilométrage* peut être relâché de 5000 kms à 10000 kms, cette relaxation sera systématiquement effectuée même s'il aurait suffit de relâcher cet attribut à 7000 kms pour satisfaire l'utilisateur. De plus, l'introduction de mots-clés qui étendent le langage SQL pour spécifier explicitement les relaxations à faire rend l'utilisation d'un tel système difficile pour un non-spécialiste.

Face à l'intérêt économique pour les moteurs de recherche et pour les sites marchands de fournir à l'utilisateur un moyen plus flexible de requêter le web ou leur base de produits en vente, cette problématique a récemment connu un regain d'intérêt. Globalement, ces techniques peuvent être classées en deux catégories : les techniques automatiques ou interactives.

Parmi les techniques automatiques de très nombreuses approches ont été proposées pour améliorer la pertinence de recherche d'information. Ainsi celles utilisant la "pseudo-relevance" feedback procède de la manière suivante. Une première requête est formulée par l'utilisateur. Les documents retournés par cette requête sont analysés afin d'extraire des termes pertinents qui viendront enrichir la requête initiale. Une seconde requête enrichie est alors calculée et les documents associés sont renvoyés à l'utilisateur. Comme le montre [Rut03], la qualité de la requête générée est discutable car les termes permettant d'enrichir la seconde requête peuvent être inappropriés. Plutôt que d'enrichir une requête initiale, l'approche proposée dans [TC04] substitue certains termes de cette requête par des termes jugés plus pertinents. Ici aussi, les substitutions proposées peuvent être inappropriées. En outre, le temps de calcul de ces enrichissements et substitutions n'est pas négligeable. Dans [JF03], un autre mécanisme est utilisé pour augmenter le nombre de réponses retournées à l'utilisateur : la suppression de critères. Le fait d'éliminer des critères permet évidemment de pouvoir renvoyer un plus grand nombre de résultats mais au dépend d'une grande perte de précision. Dans [JRMG06], la modification de la requête est basée sur le principe suivant. Une base de similarité entre requêtes et phrases est apprise hors-ligne grâce aux requêtes passées. Ainsi, à chaque nouvelle requête formulée, le système génère un ensemble de requêtes similaires. Une fonction de score est ensuite appliquée pour choisir la requête la plus adaptée. Dans [RBC+08], la problématique étudiée est d'améliorer la qualité et le profit généré par les publicités affichées par les moteurs de recherche. Pour cela, les auteurs proposent également une approche en deux phases. Dans un premier temps, une base de correspondance entre des termes et des publicités est construite par apprentissage sur les recherches précédentes. Ensuite, pour chaque nouvelle requête formulée, les correspondances entre cette nouvelle requête et les connaissances stockées dans la base sont effectuées pour afficher les publicités les plus pertinentes.

Parmi les techniques interactives, dans [Fur85], les requêtes d'utilisateurs volontaires sont analysées pour améliorer la qualité de la recherche d'information. En effet, le système demande explicitement si la recherche faite par l'utilisateur peut être utilisée pour améliorer des futures recherches. Plus récemment l'approche proposée dans [Ani03] demande à l'utilisateur de choisir les substitutions et enrichissements les plus adaptés. Cette technique permet de désambiguïser les propositions faites et diminue ainsi la charge cognitive du système.

La plupart des travaux présentés dans cette section aborde la problématique de la réécriture automatique de requêtes en exploitant une connaissance apprise lors de recherches précédentes. Outre le fait que cela suppose l'existence d'une base de connaissances suffisamment volumineuse et riche, ces approches

supposent l'existence de comportements communs. Une requête atypique sera alors mal réécrite. De plus, dans un contexte d'achat sur un site marchand, les recherches peuvent être radicalement différentes d'un utilisateur à un autre. Ainsi, nous préférons aborder la problématique de la réécriture de requêtes selon un point de vue centré utilisateur et proposons une solution n'exploitant pas une base de connaissances précédemment apprise. De plus, impliquer l'utilisateur en cours de processus risquerait de provoquer un phénomène de rejet. Il nous semble alors préférable d'exploiter les informations initialement saisies par l'utilisateur et ne pas l'inclure dans le reste du processus.

3 Cas d'étude

Pour illustrer la problématique liée à notre approche, nous considérons le cas d'un utilisateur recherchant une voiture à acheter. Sans perte de généralité, nous faisons l'hypothèse simplificatrice que les voitures à vendre sont stockées dans une table unique `VOITURES`. Chaque voiture est décrite selon 5 caractéristiques : un identifiant unique, le modèle, la couleur, le kilométrage et le prix. Un extrait de la table `VOITURES` est présenté dans le tableau 1. Dans la mesure où l'utilisateur ne peut raisonnablement pas considérer l'intégralité des voitures en vente, nous supposons qu'il formule la requête suivante : "*Quelles sont les Clio blanches dont le kilométrage n'excède pas 5 000 kilomètres et dont le prix est inférieur à 5 000 euros ?*". Si l'on considère cette requête, nous nous apercevons qu'il n'existe aucune voiture satisfaisant l'intégralité des critères de l'utilisateur. Dans une telle situation, un comportement attendu de l'utilisateur serait qu'il élargisse son espace de recherche en reformulant une requête avec des critères moins forts. S'il ne trouve toujours pas une voiture qui l'intéresse alors deux attitudes sont envisageables. Une première, positive, serait qu'il effectue une succession de requêtes de moins en moins précises jusqu'à trouver une voiture intéressante. La seconde, plus réaliste dans un contexte concurrentiel, serait qu'il considère que le site visité n'est pas en mesure de l'aider et qu'il se dirige donc vers un site marchand concurrent.

L'objectif dans cet article est de proposer à l'utilisateur de ne formuler qu'une seule requête et d'étendre automatiquement ses critères de sélection pour maximiser les réponses proches de la requête initiale. Pour cela, nous nous appuyons sur une modélisation des préférences de l'utilisateur. Ici, quelques hypothèses réalistes sont :

- L'utilisateur exige une voiture dont le prix ne dépasse pas celui indiqué dans sa requête initiale (*i.e.*, 5 000 euros);

- L'utilisateur préfère les voitures françaises, puis les allemandes, puis les autres ;
- L'utilisateur préfère les voitures blanches, puis noires ou grises et enfin les autres ;
- L'utilisateur est prêt à céder d'abord sur le kilométrage, puis sur la couleur et enfin sur le modèle.

Dès lors, nous recherchons à réécrire la requête initiale de l'utilisateur en intégrant ses préférences de sorte que (1) les critères qu'il juge immuable reste inchangés, (2) que les critères pour lesquels il est prêt à être moins exigeant soient les plus proches de la requête initiale et (3) que les n-uplets retournés par cette nouvelle requête soient plus nombreux et surtout plus pertinents pour l'utilisateur. Pour cela, nous nous appuyons sur une fonction de score qui mesure l'éloignement de la requête réécrite par rapport au gain de n-uplets permis par cette requête. Dans la suite de cet article, nous formalisons les différents concepts nécessaires à l'établissement de cette fonction de score et présentons la méthodologie adoptée pour la calculer.

Id	Modèle	Couleur	Kms	Prix
1	Clio	Blanche	6000	5000
2	Polo	Grise	10000	4000
3	206	Grise	7000	5000
4	Golf	Noire	6000	4500
5	Ibiza	Jaune	7000	4000
6	Ibiza	Rouge	4000	5000
7	Clio	Rouge	4000	5000
8	206	Noire	6000	4500
9	Polo	Noire	7000	5000
10	Polo	Blanche	10000	4000
11	Ibiza	Rouge	10000	5000
12	Golf	Blanche	7000	5000
13	206	Blanche	6000	4500
14	Polo	Grise	4000	4000
15	Clio	Grise	7000	4000

Table 1: Extrait de la table `VOITURES`.

4 Définitions et problématique

4.1 Définitions préliminaires

Soit $\mathcal{T} = (A_1, \ldots, A_n)$ une relation où sont stockées les données décrites sur n attributs $A_1, \ldots, A_n$. Nous notons $\mathcal{A}$ l'ensemble de ces attributs. Généralement,

les données stockées sur une site marchand sont si nombreuses qu'une modélisation plus appropriée (modélisation en étoile, flocon ou constellation) est adoptée. Nous assumons néanmoins qu'une telle relation peut toujours être calculée ou matérialisée.

Définition 1. *Un critère c_i est un triplet $c_i = (A_i, op_i, a_i)$ tel que A_i est un attribut de $\mathcal{T}$, op_i est un opérateur tel que $op_i \in \{\leq, =, \geq, BETWEEN\}$ et a_i est un ensemble d'instances de A_i. La sémantique associée au critère $c = (A_i, op, a_i)$ est alors $\{\ t \mid t \in \mathcal{T}$ et $t_{A_i}\ op_i\ a_i$ est vrai $\}$.*

Soit $c_i = (A_i, op_i, a_i)$ un critère. Si $op_i \in \{\leq, =, \geq\}$ alors $|a_i| = 1$ et nous notons a_{i1} l'unique valeur de cet ensemble (a_i s'il n'y a pas d'ambiguïté). Si $op_i \in \{BETWEEN\}$ alors $|a_i| = 2$ et nous notons a_{i1} (resp. a_{i2}) la première (resp. la seconde) valeur de cet ensemble. La sémantique associée à un tel critère est "A_i doit être entre à a_{i1} et a_{i2} (au sens large)". Enfin, si $op_i \in \{IN\}$ alors $|a_i| = n$ (avec $1 \leq n \leq |Dom(A_i)|$). Nous notons a_{ik} (avec $1 \leq k \leq n$) le $k^{ème}$ élément de cet ensemble. La sémantique associée à ce critère est "A_i doit être égale à a_{i1} ou ... ou à a_{in}".

Exemple 1. *Le critère $c = (Prix, \leq, 5000)$ signifie que l'utilisateur est intéressé par des produits dont le prix est inférieur ou égal à 5000.*

Définition 2. *Un attribut A est dit extensible s'il existe un ordre $<_A$ sur les valeurs de cet attribut et nous notons $\mathcal{A}_E$ l'ensemble des attributs extensibles de $\mathcal{T}$. Un critère $c = (A, op, a)$ est alors dit extensible si A est un attribut extensible. A contrario, un attribut A est dit fixe s'il n'est pas extensible et l'ensemble des attributs fixes de $\mathcal{T}$ est noté $\mathcal{A}_F$. Un critère $c = (A, op, a)$ est alors dit fixe si A est un attribut fixe.*

Exemple 2. *Si l'on considère le cas d'étude précédent, les attributs* Modèle, Couleur *et* Kms *sont des attributs extensibles et le critère $c = (Kms, \leq, 5000)$ est donc un critère extensible. Ensuite, dans la mesure où l'utilisateur ne souhaite pas céder sur le prix de la voiture recherchée, l'attribut* Prix *est un attribut fixe et le critère $c = (Prix, \leq, 5000)$ est donc un critère fixe.*

Remarque 1. *L'ordre dont les valeurs d'un attribut est muni peut être de natures différentes. Si l'attribut est numérique et continu (*e.g., *un kilométrage), cet ordre sera naturellement l'ordre des réels dans $\mathbb{R}$. Dans le cas où l'attribut A est symbolique, un ordre possible serait l'ordre lexicographique. Un autre ordre sémantiquement plus riche serait un ordre exprimant les préférences de l'utilisateur. Par exemple, Grise $<_{Coul.}$ Blanche exprimerait le fait que l'utilisateur préfère les voitures blanches aux voitures grises.*

Définition 3. *Une requête utilisateur $\mathcal{R}$ est définie par $\mathcal{R} = \mathcal{CF}_R \wedge \mathcal{CE}_R$ avec $\mathcal{CE}_R$ une conjonction non vide de critères extensibles et $\mathcal{CF}_R$ une conjonction non vide de critères fixes. De plus, chaque critère d'une requête utilisateur porte sur un attribut différent de $\mathcal{T}$.*

Exemple 3. *La requête formulée par l'utilisateur peut être écrite $\mathcal{R} = \mathcal{CF}_R \wedge \mathcal{CE}_R$ avec $\mathcal{CF}_R = (Prix, \leq, 5000)$ et $\mathcal{CE}_R = (Modèle, =, Clio) \wedge (Couleur, =, Blanche) \wedge (Kms, \leq, 5000)$.*

Définition 4. *Soit $\mathcal{R} = \mathcal{CF}_R \wedge \mathcal{CE}_R$ une requête utilisateur. Une extension e_i est un critère (A_i, op, a'_i) tel que $(A_i, op, a_i) \in \mathcal{CE}_R$ (*i.e., *A_i est un attribut extensible) et $a_i \leq_{A_i} a'_i$. Une combinaison d'extensions C_i est une alors une conjonction de critères tels qu'au moins l'un des critères soit une extension.*

Exemple 4. *Selon le cas d'étude, $(Kms, \leq, 7000)$ est une extension et $(Modèle, =, Clio) \wedge (Couleur, =, Grise) \wedge (Kms, \leq, 7000)$ une combinaison d'extensions.*

4.2 Problématique

Soit une requête utilisateur $\mathcal{R} = \mathcal{CF}_R \wedge \mathcal{CE}_R$. La problématique associée à cet article est la réécriture de $\mathcal{R}$ en une requête $\mathcal{R}' = \mathcal{CF}_{R'} \wedge \mathcal{CE}_{R'}$ telle que :

- $\mathcal{CF}_R = \mathcal{CF}_{R'}$;
- Les critères $c'_i \in \mathcal{CE}_{R'}$ soient les plus proches possibles des critères $c_i \in \mathcal{CE}_R$;
- Le nombre de n-uplets retournés par $\mathcal{R}'$ soit supérieur à celui retourné par $\mathcal{R}$.

En d'autres termes, il s'agit de trouver une requête $\mathcal{R}'$ qui respecte au mieux le compromis entre la pertinence de la réponse et l'éloignement de la requête d'origine.

5 TIGER

Dans cette section, nous décrivons, TIGER, une méthode visant à élire une combinaison d'extensions pour réécrire la requête initiale de l'utilisateur. Plusieurs étapes sont nécessaires. Dans un premier temps, un traitement doit être appliqué sur les valeurs des attributs extensibles de $\mathcal{T}$. Si l'attribut est numérique, il faut alors discrétiser ses valeurs pour réduire le nombre de combinaisons. Si l'attribut est symbolique, un ordre sur ses valeurs est défini. Il sera alors possible de déterminer une distance entre les valeurs d'un tel attribut et le critère initial. Pendant cette étape, nous montrons comment intégrer les

préférences utilisateurs intra-attributs (définies parmi les valeurs d'un même attribut). Au cours de la seconde étape, les n-uplets ne vérifiant pas les critères fixes ne sont plus considérés. Une fonction mesurant la distance entre un n-uplet restant et la requête initiale est introduite. Au cours de la troisième étape, la construction d'un treillis partiel permet de mesurer l'ordre existant entre les différentes combinaisons. Ce treillis est finalement exploité pour calculer le score associé à chaque combinaison. La combinaison dont le score est maximal sera alors retenue. Intuitivement, cette combinaison représente le meilleur compromis entre la distance à la requête initiale et le gain engendré. Nous formalisons maintenant ces 4 étapes.

5.1 Réécriture des attributs extensibles

Dans un premier temps, un traitement est appliqué sur les valeurs des attributs extensibles de $\mathcal{T}$. Pour les attributs extensibles numériques, il s'agit de discrétiser les valeurs prises par cet attribut afin de réduire le nombre d'extensions possibles. Concernant les attributs symboliques, nous montrons comment intégrer les préférences utilisateurs intra-attributs (*i.e.,* des préférences sur les valeurs d'un attribut) au cours de cette phase.

5.1.1 Attributs extensibles numériques

Dans le cas où l'attribut associé à un critère extensible est numérique (*e.g.,* le prix d'une voiture), il serait peu intéressant de considérer l'extension d'un attribut à toutes les valeurs prises par cet attribut. Pour cette raison, nous choisissons de discrétiser les valeurs prises par cet attribut afin de ne considérer qu'un nombre plus restreint d'extensions. De nombreuses techniques de discrétisation [Cat91] peuvent être utilisées comme par exemple :

- Découpage equi-width : chaque zone est un intervalle de même largeur ;
- Découpage equi-depth : chaque zone est un intervalle comprenant le même nombre de valeurs.

Nous introduisons ainsi la fonction $Disc_{A_i}(a_i)$ (où $a_i \in Dom(A_i)$) afin de représenter le résultat de cette discrétisation.

Définition 5. *Soit A_i un attribut extensible numérique. Nous définissons la fonction $Disc_{A_i} : Dom(A_i) \rightarrow [0; nbClasses]$ (où $nbClasses$ est le nombre de classes générées lors de la discrétisation) telle que :*

- $Disc_{A_i}(a_k) \leq Disc_{A_i}(a_l)$ *si* $a_k >_{A_i} a_l$ *où* $(a_k, a_l) \in Dom(A_i) \times Dom(A_i)$*;*
- $Disc_{A_i}(a_k) = Disc_{A_i}(a_l)$ *si* $a_k = a_l$ *où* $(a_k, a_l) \in Dom(A_i) \times Dom(A_i)$*;*

Exemple 5. *Dans la table* `VOITURES`*, l'attribut Kms est le seul attribut extensible numérique. Le tableau 2 (gauche) présente les valeurs prises par la fonction* $Disc_{Kms}$*.*

x	$Disc_{Kms}(x)$
4000	0
6000	1
7000	2
10000	3

x	$Disc_{Coul.}(x)$
$Blanche$	0
$\{Noire, Grise\}$	1
$\{Jaune, Rouge\}$	2

Table 2: Illustration des fonctions $Disc_{Kms}$ et $Disc_{Coul.}$

5.1.2 Prise en compte des préférences intra-attributs pour les attributs extensibles symboliques

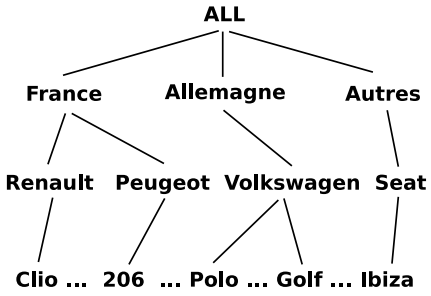
Comme nous l'avons précédemment mentionné, plusieurs relations d'ordre peuvent exister pour des attributs symboliques. Ici, nous nous focalisons sur une relation d'ordre qui exprimerait les préférences de l'utilisateur [Cho03, Cho07, Kie02]. Ces préférences peuvent être exprimées au niveau où sont stockées les données(*e.g.,* la couleur de la voiture) ou à un niveau de granularité plus général (*e.g.,* la nationalité du fabricant de voitures).

Dans le cas où les préférences de l'attribut A_i sont spécifiées sur les valeurs brutes prises par l'attribut, la fonction $Disc_{A_i}$ précédemment définie peut être appliquée.

Exemple 6. *Supposons que les préférences utilisateurs sur l'attribut Couleur sont les suivantes :* $\{autres\} < \{noire, grise\} < \{blanche\}$*. L'utilisateur préfère donc les voitures blanches, puis les noires ou grises et enfin les voitures d'une autre couleur. Le tableau 2 (droite) présente les valeurs prises par la fonction* $Disc_{Coul}$ *qui traduit ces préférences.*

Considérons maintenant le cas où l'utilisateur exprime ses préférences sur un niveau plus général que celui où sont stockées les données dans la table $\mathcal{T}$. Nous notons $A_i^{\uparrow}$ ce niveau plus général, $a_1^{\uparrow} > a_2^{\uparrow}$ pour spécifier que l'utilisateur préfère $a_2^{\uparrow}$ à $a_1^{\uparrow}$ et $a_i^{\uparrow} = Up(a_j)$ pour indiquer que a_j est une spécialisation de $a_i^{\uparrow}$. Avec ces notations, la fonction $Disc_{A_i}(a_i)$ (où $a_i \in Dom(A_i)$) peut être définie ainsi.

Définition 6. *Soit* A_i *un attribut extensible symbolique. La fonction* $Disc_{A_i}$ *est telle que :*



x	$Disc_{Mod.}(x)$
$\{Clio, 206\}$	0
$\{Polo, Golf\}$	1
$Ibiza$	2

Figure 1: A gauche, extrait de la hiérarchie associée à l'attribut $Modèle$. A droite, valeurs associées à $Disc_{Mod..}$ intégrant les préférences intra-attribut de l'utilisateur

- $Disc_{A_i}(a_k) < Disc_{A_i}(a_l)$ *si* $Up(a_k) > Up(a_l)$ *où* $(a_k, a_l) \in Dom(A_i) \times Dom(A_i)$*;*
- $Disc_{A_i}(a_k) = Disc_{A_i}(a_l)$ *si* $Up(a_k) = Up(a_l)$ *où* $(a_k, a_l) \in Dom(A_i) \times Dom(A_i)$*;*

Exemple 7. *Les préférences sur la nationalité du constructeur automobile illustrent une préférence utilisateur décrite sur un niveau plus général que celui où sont stockées les données. En se basant sur un extrait de la hiérarchie des modèles présentée dans la partie gauche de la figure 1, nous pouvons modéliser la préférence "*L'utilisateur préfère les voitures françaises, puis les allemandes, puis les autres*" comme présenté dans le tableau en partie droite de la figure 1.*

5.2 Prise en compte de la distance à la requête initiale

Afin de déterminer quelle est la meilleure combinaison d'extensions, la table $\mathcal{T}$ doit refléter la distance entre chaque n-uplet de $\mathcal{T}$ et la requête de l'utilisateur $\mathcal{R}$. Pour ce faire, nous définissons une fonction, $Trans_{A_i}$, appliquée à chaque attribut extensible..

Définition 7. *Soit $c_i = (A_i, op_i, a_i)$ un critère extensible. Nous définissons la fonction $Trans_{A_i}(Dom(A_i)) \rightarrow \mathbb{N}$ ainsi :*

- ***Si* op $= \leq$**

$$Trans(a'_i) = \begin{cases} 0 & si\ Disc(a'_i) \leq Disc(a_i) \\ Disc(a'_i) - Disc(a_i) & sinon \end{cases}$$

- ***Si* op $= \geq$**

$$Trans(a'_i) = \begin{cases} 0 & si\ Disc(a_i) \leq Disc(a'_i) \\ Disc(a_i) - Disc(a'_i) & sinon \end{cases}$$

- ***Si* op $= =$**

$$Trans(a'_i) = \begin{cases} 0 & si\ Disc(a'_i) = Disc(a_i) \\ |Disc(a'_i) - Disc(a_i)| & sinon \end{cases}$$

- ***Si* op $=$ *BETWEEN***

$$Trans(a'_i) = \begin{cases} 0 & si\ Disc(a_{i1}) \leq Disc(a'_i) \leq Disc(a_{i2}) \\ Disc(a_{i1}) - Disc(a'_i) & si\ Disc(a'_i) < Disc(a_{i1}) \\ Disc(a'_i) - Disc(a_{i2}) & si\ Disc(a_{i2}) < Disc(a'_i) \end{cases}$$

La fonction $Trans$ permet de mesurer la distance entre la valeur d'un attribut et celle spécifiée par l'utilisateur (*i.e.,* à travers un critère). Nous notons $Trans^{-1}$ la fonction inverse de $Trans$. Nous définissons maintenant une fonction, $CombTrans$, mesurant la distance entre une combinaison d'extensions C et une requête utilisateur $\mathcal{R}$.

Définition 8. *Soient $\mathcal{R} = \mathcal{CF}_R \wedge \mathcal{CE}_R$ une requête utilisateur et $C = (c_1, \ldots, c_m)$ une combinaison d'extensions avec $c_i = (A_i, op_i, a'_i)$, (où A_i est un attribut extensible). La fonction $CombTrans$ est définie ainsi :*

$$CombTrans(C) = \sum_{i=l}^{m} (Trans(a'_i))$$

Exemple 8. *Considérons le premier n-uplet du tableau 1. La combinaison d'extensions associée est $C = (Modèle, =, Clio) \wedge (Couleur, =, Blanche) \wedge (Kms, \leq, 6000)$. Par conséquent, $CombTrans(C) = 0 + 0 + 1 = 1$.*

Soit $C = (c_1, \ldots, c_n)$ une combinaison d'extensions telle que $c_i = (A_i, op, a'_i)$. Dans la suite de cet article, cette même combinaison pourra également être notée $C = (Trans(a'_1), \ldots, Trans(a'_n))$. Par exemple, la combinaison $C = (Modèle, =, Clio) \wedge (Couleur, =, Blanche) \wedge (Kms, \leq, 6000)$ pourra également être notée $(0, 0, 1)$.

Dans un premier temps, les n-uplets qui ne satisfont pas les critères fixes sont exclus. Ensuite, pour chaque valeur d'un attribut extensible de chaque

n-uplet restant, la fonction $Trans$ est appliquée et $CombTrans$ est calculée. Les n-uplets sont alors groupés et ordonnés (ascendant) en fonction des valeurs de $CombTrans$. Ce regroupement permet de quantifier le nombre de n-uplets qu'une combinaison d'extension entraîne. Nous notons $Gain(C_i)$ le nombre de n-uplets de $\mathcal{T}$ de même combinaison. Il faut cependant remarquer que ce nombre ne représente pas le gain (en terme de n-uplets) total d'une combinaison d'extensions. Par exemple, pour considérer le nombre de n-uplets réellement retournés par la combinaison $(1, 0, 1)$, il faut considérer $Gain(1, 0, 1)$ mais aussi $Gain(0, 0, 1)$, $Gain(1, 0, 0)$ et $Gain(0, 0, 0)$. Ainsi, ordonner les combinaisons selon $CombTrans$ permet de s'assurer qu'une combinaison ne sera considérée qu'une fois toutes les combinaisons utiles au calcul du gain effectif auront été considérées.

Id	$Trans_{Mod.}$	$Trans_{Coul.}$	$Trans_{Kms}$
1	0	0	1
2	1	1	3
3	0	1	2
4	1	1	1
5	2	2	2
6	2	2	0
7	0	2	0
8	0	1	1
9	1	1	2
10	1	0	3
11	2	2	3
12	1	0	2
13	0	0	1
14	1	1	0
15	0	1	2

Table 3: Application de la fonction $Trans$ à chaque valeur d'un attribut extensible de $\mathcal{T}$

Prise en compte des préférences utilisateur inter-attributs. Jusqu'ici nous avons supposé que l'utilisateur n'avait pas de préférence quant aux critères qu'il souhaitait étendre. Nous considérons à présent la possibilité de pouvoir étendre certains critères plutôt que d'autres.Par exemple, l'utilisateur peut être moins exigeant sur la couleur de la voiture alors qu'il tient à ce que le kilométrage de la voiture s'éloigne le moins possible de sa contrainte initiale. Nous montrons ici comment ces préférences peuvent être prises en compte.

Soient $A_1, \ldots, A_n$ les attributs de la table $\mathcal{T}$. Les préférences inter-attributs de l'utilisateur sont modélisées sous la forme d'un ordre partiel $\prec_{PIA}$. Ainsi, $\mathcal{A}_1 \prec_{PIA} \mathcal{A}_2 \prec_{PIA} \ldots \prec_{PIA} \mathcal{A}_l$ (avec $\mathcal{A}_i = \{A_{i1}, \ldots, A_{ik}\}$, $\mathcal{A}_1 \cup \mathcal{A}_2 \cup \ldots \cup \mathcal{A}_l =$

$\mathcal{A}$ et $\mathcal{A}_1 \cap \mathcal{A}_2 \cap \ldots \cap \mathcal{A}_l = \emptyset$) modélise le fait que l'utilisateur souhaite étendre les attributs de $\mathcal{A}_1$ avant ceux de $\mathcal{A}_2$ et ainsi de suite. Dès lors, il est légitime de pénaliser plus fortement une extension de critère si elle concerne un attribut important pour un utilisateur par rapport à une extension portant sur un attribut peu prioritaire pour l'utilisateur. Nous introduisons alors un ensemble de poids $\sigma_1, \sigma_2, \ldots, \sigma_l$ associés à chaque ensemble $\mathcal{A}_i$ $(1 \leq i \leq l)$ tel que $\sigma_m < \sigma_n$ implique $\mathcal{A}_m \prec \mathcal{A}_n$. Ces poids permettent de définir la fonction $TransPref$. Intuitivement, cette fonction représente une distance pondérée au sein du domaine de définition d'un attribut. Plus un attribut est important pour un utilisateur, plus une extension sera considérée comme significativement éloignée du critère initial.

Définition 9. *Soit $c_i = (A_i, op_i, a_i)$ un critère extensible. Nous définissons la fonction $TransPref_{A_i}(Dom(A_i)) \rightarrow \mathbb{N}$ ainsi :*

$$TransPref_{A_i}(a'_i) = \sigma_j \times Trans_{A_i}(a'_i) \quad où \quad a_i \in \mathcal{A}_i$$

Exemple 9. *Dans la mesure où l'utilisateur préfère céder sur le kilométrage plutôt que sur le modèle, nous affectons un poids plus faible à l'attribut* Kilométrage *qu'à l'attribut* Modèle. *Les poids associés à ces attributs peuvent donc être $\sigma_{Kms} = 1$ et $\sigma_{Mod.} = 3$.*

5.3 Construction du treillis partiel

Nous venons d'évoquer le fait que pour une combinaison C_i donnée, $Gain(C_i)$ ne permet pas de déterminer le nombre de n-uplets réellement retournés. Nous formalisons maintenant ces relations entre combinaisons à l'aide d'un treillis.

Définition 10. *Soit $G = (\mathcal{E}, \leq_G)$ un treillis où $\mathcal{E}$ est l'ensemble des extensions et $\leq_G$ est une relation d'ordre partiel telle que pour $C_1 = (c_{11}, \ldots, c_{1k})$ et $C_2 = (c_{21}, \ldots, c_{2k})$, nous avons $C_1 \leq_G C_2$ si $\forall i \in [1, k]$, $c_{1i} \leq c_{2i}$. De plus, $\forall C_1, C_2$, leur borne inférieure et leur borne supérieure sont définies ainsi :*

- $Inf(C_1, C_2) = (min(c_{11}, c_{21}), \ldots, min(c_{1k}, c_{2k}))$
- $Sup(C_1, C_2) = (max(c_{11}, c_{21}), \ldots, max(c_{1k}, c_{2k}))$

Exemple 10. *Le treillis partiel des combinaisons d'extensions possibles dans $\mathcal{T}$ est présenté dans la figure 2. Ce treillis partiel permet de représenter la relation d'ordre partiel qui existe entre chaque ligne du tableau 3. Pour chaque combinaison, le gain associé est indiqué à droite de la combinaison.*

La fonction $GainTotal(C)$ est introduite pour comptabiliser le nombre de n-uplets effectivement permis par C.

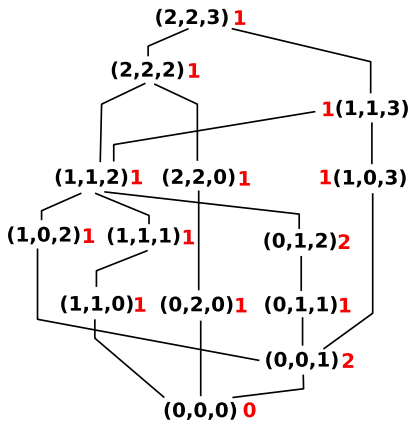


Figure 2: Treillis partiel des combinaisons d'extensions possibles selon $\mathcal{T}$

Définition 11. *Soit C une combinaison d'extensions. Le nombre de n-uplets que C permet est donné par la fonction GainTotal telle que*

$$GainTotal(C) = \sum_{C_i \leq_G C} Gain(C_i)$$

Exemple 11. *Considérons la combinaison $(0, 1, 2)$. En étudiant le treillis partiel présenté dans la figure 2, nous constatons que $GainTotal(0, 1, 2) = Gain(0, 1, 2) + Gain(0, 1, 1) + Gain(0, 0, 1) + Gain(0, 0, 0) = 2 + 1 + 2 + 0 = 5$.*

5.4 Fonction de score

Une extension de critères est pertinente si elle maximise le nombre de n-uplets renvoyés par la nouvelle requête tout en minimisant l'écart entre la requête initiale et cette nouvelle requête. La fonction proposée répond à cette attente car elle est le rapport entre le gain total rapporté par la nouvelle requête (*i.e.*, le nombre de n-uplets qui seront renvoyés par cette requête) et la distance *relative* (*i.e.*, la distance entre la requête utilisateur et la nouvelle requête divisé par le gain permis par cette extension). Nous définissons maintenant cette fonction de score.

Définition 12. *Soient C une combinaison d'extensions et $\mathcal{T}$ une table relationnelle. Le score associé à la combinaison C dans la table $\mathcal{T}$ est :*

$$Score_{\mathcal{T}}(C) = \frac{GainTotal(C)}{\frac{CombTrans(C)}{Gain(C)}}$$

Exemple 12. *Nous illustrons la fonction de score en calculant Score(0,1,2).*

$$Score_{\mathcal{T}}(0,1,2) = \frac{2+1+2+0}{\frac{3}{2}} = \frac{10}{3}$$

Ce score est calculé pour toutes les combinaisons possibles. La combinaison retenue est celle dont le score est maximal. Si plusieurs combinaisons possèdent le même score maximal, nous privilégions celle qui est plus proche de la requête initiale de l'utilisateur (*i.e.*, la combinaison ayant le plus petit $CombTrans$). Nous notons $BestComb = (Trans(a_{b1}), \ldots, Trans(a_{bn}))$ cette combinaison.

$Trans_{Mod.}$	$Trans_{Coul}$	$Trans_{Kms}$	$CombTrans$	$Gain$	$GainTotal$	$Score$
0	0	1	1	2	2	1
1	1	0	2	1	1	$\frac{1}{2}$
0	2	0	2	1	1	$\frac{1}{2}$
0	1	1	2	1	3	$\frac{3}{2}$
1	0	2	3	1	3	1
1	1	1	3	1	2	$\frac{2}{3}$
0	**1**	**2**	**3**	2	5	$\frac{\mathbf{10}}{\mathbf{3}}$
1	1	2	4	1	10	$\frac{5}{2}$
2	2	0	4	2	2	$\frac{1}{2}$
1	0	3	4	1	3	$\frac{3}{4}$
1	1	3	5	1	11	$\frac{11}{5}$
2	2	2	6	1	12	$\frac{13}{6}$
2	2	3	7	1	15	$\frac{15}{7}$

Table 4: Tableau des scores associés à chaque combinaison

Le tableau 4 présente les scores associés à chaque combinaison d'extensions. La combinaison d'extensions retenue est donc $(0,1,2)$. La requête générée par la méthode proposée est alors $\mathcal{R}' = \mathcal{CF}_R \wedge \mathcal{CE}_{R'}$ où $\mathcal{R}$ est la requête utilisateur et $\mathcal{CE}_{R'} = ((A_1, op_1, Trans^{-1}(Trans(a_{b1}))), \ldots, (A_n, op_1, Trans^{-1}(Trans(a_{bn}))))$. La nouvelle requête est donc $\mathcal{R}' = \mathcal{CF}_R \wedge \mathcal{CE}_{R'}$ avec $\mathcal{CF}_R = (Prix, \leq, 5000)$ et $\mathcal{CE}_R = (Modèle, IN, \{Clio, 206\}) \wedge (Couleur, =, \{Blanche, Noire, Grise\}) \wedge (Kms, \leq, 7000)$. Le tableau 5 présente les voitures qui seront alors proposées à l'utilisateur.

6 Expérimentations

Dans cette section, nous évaluons l'approche proposée en nous posant les deux questions suivantes. Premièrement, l'implantation de notre méthode dans un

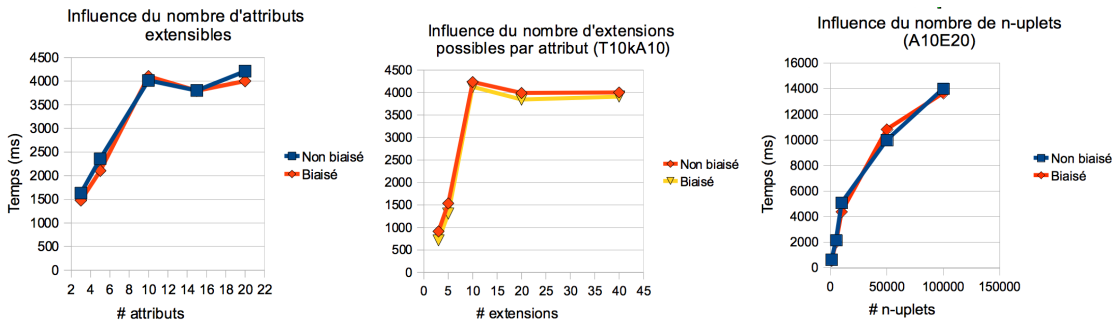
Id	Modèle	Couleur	Kms	Prix
1	Clio	Blanche	6000	5000
3	206	Grise	7000	5000
8	206	Noire	6000	4500
13	206	Blanche	6000	4500
15	Clio	Grise	7000	4000

Table 5: Voitures répondants aux critères de la requête étendue $\mathcal{R}'$

site marchand est-elle réaliste ? Dans un tel contexte, il est important que la réponse à une requête soit calculée et retournée rapidement. Nous évaluons donc le temps nécessaire pour trouver la meilleure combinaison. Ensuite, nous réalisons une étude qualitative de la combinaison retournée en évaluant le gain engendré par la combinaison et sa distance à la requête d'origine. Les jeux de données ont été générés aléatoirement et nous adoptons la convention suivante pour les nommer : A10E20T10k signifie que le jeu de données a 10 attributs extensibles, que chaque attribut extensible possède 20 niveaux d'extensions et que le jeu de données est composé de 10000 n-uplets. Deux types de jeu de données sont générés. Dans le premier, les n-uplets sont générés aléatoirement et uniformément. Ce type de données, dites non biaisées, permettent de mesurer le comportement de notre méthode dans un contexte d'évaluation critique. Dans le second, nous tentons de modéliser une situation plus réaliste où l'essentiel des valeurs sont regroupées dans un intervalle de valeurs assez proches (*i.e.*, il y a peu de valeurs extrêmes). Pour cela, nous générons les jeux de données en faisant en sorte que 80% des valeurs d'attributs générés aléatoirement aient une valeur de $Trans$ entre 0 et $MaxTrans/3$. Ces jeux de données seront dits biaisés dans la suite de cette section. Par exemple, si A10E20T10k est biaisé, alors 80% des $Trans$ générés seront compris entre 0 et 6 (*i.e.*, la partie entière de 20/3). Cette série d'expérimentations a été réalisée sur un MacBook 2,4 GHz Intel Core 2 Duo avec 2 Go de RAM. Les méthodes ont été implémentées en JAVA 1.6.

Evaluation quantitative. Nous évaluons l'impact de 3 paramètres sur le temps de recherche de la meilleure combinaison : le nombre d'attributs extensibles, le nombres d'extensions par attribut et le nombre de n-uplets validant les critères fixes. La figure 3 présente les résultats obtenus sur des jeux de données biaisés ou non. Faute de place, nous ne pouvons décrire chacune des courbes précisément. Nous dressons malgré tout quelques remarques. Premièrement, les performances obtenues sur des jeux biaisés ou non sont très similaires. Il faut remarquer néanmoins que lorsque les valeurs associées aux attributs sont raisonnables (*i.e.,* le nombre de nœuds dans le treillis partiel est

significativement inférieur au nombre de n-uplets dans la table), la recherche de la meilleure combinaison est légèrement plus rapide pour les jeux biaisés. Ensuite, les expérimentations mesurant l'impact du nombre d'attributs et du nombre d'extensions montrent que le temps de recherche de la meilleure combinaison est borné autour de 4 secondes. Enfin, le paramètre dont l'impact est le plus significatif est le nombre de n-uplets vérifiant les critères fixes. Cela s'explique par le fait que plus ce nombre est grand, plus le treillis partiel aura de nœuds. Dans la mesure où un parcours de tous les nœuds est effectué pour retrouver le score maximum, cet impact est logique. Cependant, dans la mesure où cette quantité est le nombre de n-uplets validant les critères fixes, une recherche de la meilleure combinaisons effectuée en 5 secondes pour 10000 n-uplets considérés est très acceptable.



(a) # attributs (b) # extensions (c) # n-uplets

Figure 3: Impact de certains paramètres sur le temps d'exécution

Evaluation qualitative. Nous évaluons ici la qualité des combinaisons retenues en fonction de deux paramètres : le nombre d'attributs extensibles et le nombres d'extensions par attribut. Deux critères sont retenues pour mesurer cette qualité : le gain engendré par la combinaison et la distance entre la requête initiale et celle réécrite. Nous mesurons le gain par le rapport entre le nombre de n-uplets supplémentaires retournés et le nombre de n-uplets qui auraient pu l'être si les extensions étaient maximales. Par exemple, si 80 n-uplets sont retournés parmi les 100 qui auraient pu l'être, le gain sera 80%. Le second critère évalue la distance entre la requête initiale (*i.e.*, où toutes les valeurs de $Trans$ sont égales à 0) et la combinaison retenue. Cette distance est donnée par $CombTrans$ qui représente le niveau du treillis dans lequel se situe la combinaison. Les jeux de données sont biaisés. Le tableau 6 présente les résultats obtenus. Globalement, les combinaisons retenues permettent un gain

#attributs ext.	Gain (en %)	CombTrans
3	68	1 sur 25
5	71	10 sur 75
10	53	19 sur 190
15	21	50 sur 285
20	26	61 sur 380

#extensions	Gain (en %)	CombTrans
3	63	1 sur 20
5	65	1 sur 40
10	50	12 sur 90
20	23	70 sur 190
40	19	90 sur 390

Table 6: Impact de certains paramètres sur la qualité de la combinaison retenue

significatif de n-uplets retournés tout en s'éloignant peu de la requête initiale. Par contre, lorsque le nombre de nœuds dans le treillis partiel est très proche du nombre de n-uplets dans la table, la qualité des résultats se dégrade. En effet, cela signifie que peu de nœuds permettent un gain significatif et donc que la combinaison retenue sera située à un niveau plus haut dans le treillis.

Pou résumer, cette étude expérimentale nous a permis de constater que (1) le temps de recherche de la meilleure combinaison est relativement faible et que (2) le gain engendré est significatif tout en restant proche de la requête initiale. Dès lors, il est tout à fait envisageable d'implanter une telle approche dans un site marchand.

7 Conclusion

Dans cet article, nous justifions le besoin stratégique pour un site marchand de fournir à l'utilisateur un système de recherche plus flexible que l'interprétation stricte des champs d'un formulaire. Nous proposons alors une solution pour réécrire une requête en étendant certains critères pour maximiser la qualité de la réponse retournée. Pour cela, l'ensemble des combinaisons d'extensions possible est évalué afin d'élire celle qui maximise le compromis entre le gain de n-uplets retournés et la distance à la requête d'origine. Une fonction de score est introduite à cette fin. Une étude expérimentale valide notre approche en montrant que (1) le temps de calcul de la meilleure combinaison est faible et (2) que le gain de n-uplets intéressants retournés est important.

Plusieurs perspectives peuvent être envisagées à la suite de ce travail. Dans un premier temps, il serait intéressant d'évaluer notre approche sur des données réelles. Cette série d'expérimentations serait alors l'occasion d'inclure un ensemble d'utilisateurs dans le processus de validation. Ensuite, l'étude des interactions possibles entre cette approche et un système de recommandation apparaît pertinente pour enrichir chacune des deux méthodes. Enfin, inclure une base statique de connaissances telle que WordNet [F+98] permettrait d'identifier des concepts similaires et améliorerait ainsi la qualité de la

requête reformulée.

References

[Ani03] P. Anick. Using terminological feedback for web search refinement: a log-based study. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 88–95, Toronto, Canada, 2003. ACM.

[AT05] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[Bur02] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, November 2002.

[Cat91] J. Catlett. On changing continuous attributes into ordered discrete attributes. In *Machine Learning—EWSL-91*, pages 164–178. Springer, 1991.

[Cho03] J. Chomicki. Preference formulas in relational queries. *ACM Transactions on Database Systems (TODS)*, 28(4):427–466, 2003.

[Cho07] J. Chomicki. Semantic optimization techniques for preference queries. *Information Systems*, 32(5):670–684, 2007.

[CYC+96] W.W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, and C. Larson. Cobase: A scalable and extensible cooperative information system. *Journal of Intelligent Information Systems*, 6(2):223–259, 1996.

[F+98] C. Fellbaum et al. *WordNet: An electronic lexical database*. MIT press Cambridge, MA, 1998.

[Fur85] G. W Furnas. Experience with an adaptive indexing scheme. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 131–135, 1985.

[GGM92] T. Gaasterland, P. Godfrey, and J. Minker. Relaxation as a platform for cooperative answering. *Journal of Intelligent Information Systems*, 1(3):293–321, 1992.

[JF03] R. Jones and D. C. Fain. Query word deletion prediction. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 435–436, Toronto, Canada, 2003. ACM.

[JRMG06] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web*, pages 387–396. ACM, 2006.

[Kie02] W. Kießling. Foundations of preferences in database systems. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 311–322. VLDB Endowment, 2002.

[MSR04] S. E Middleton, N. R Shadbolt, and D. C De Roure. Ontological user profiling in recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):54–88, 2004.

[RBC+08] F. Radlinski, A. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, and L. Riedel. Optimizing relevance and revenue in ad search: a query substitution approach. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 403–410, Singapore, Singapore, 2008. ACM.

[Rut03] I. Ruthven. Re-examining the potential effectiveness of interactive query expansion. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 213–220, Toronto, Canada, 2003. ACM.

[TC04] E. Terra and C. L.A. Clarke. Scoring missing terms in information retrieval tasks. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 50–58, Washington, D.C., USA, 2004. ACM.