Un système coopératif pour le tri des résultats de moteurs de recherche

Rushed Kanawati

LIPN - CNRS UMR 7030
99 av. J.B. Clément 93430 Villetaneuse
rushed.kanawati@lipn.univ-paris13.fr

Résumé : Ce papier décrit un système coopératif d'aide au tri des résultats de recherches sur le Web. L'approche proposée repose sur le principe de partage implicite des expériences de recherche d'information au sein d'un groupe d'utilisateurs qui ont des centres d'intérêts communs. Un agent assistant personnel est associé à chaque membre du groupe. L'agent observe les requêtes soumises par l'utilisateur aux moteurs de recherche sur le Web et intercepte les résultats retournés. Le couple (requête, résultats) est alors transmis aux autres agents assistants qui vont chacun proposer un classement des documents retournés en fonction de leurs propres expériences. La méthodologie du raisonnement à partir de cas est utilisée par les agents pour calculer les classements. L'agent émetteur fusionne les listes triées proposées par les tous les agents et présente le résultat à l'utilisateur. Dans ce papier nous décrivons l'architecture général du système, le Cycle RàPC appliqué par chaque agent et le protocole de coopération employé. Les premiers résultats expérimentaux sont aussi rapportés.

Mots-clés : Agents RàPC, Système égal à égal, Recherche d'information, Tri de résultats, Web.

1 Introduction

La présentation et le tri des résultats des moteurs de recherches est un problème important dans le domaine de la recherche d'information sur le Web. En soumettant une requête à un moteur de recherche l'utilisateur attend en retour un ensemble de documents *triés* en fonction de leur pertinence par rapport à ses besoins informationnels. Les réponses des moteurs de recherche sont rarement conformes aux besoins de l'utilisateur. Les sources du problème sont multiples : Les requêtes utilisateurs sont souvent vagues. Les mots clés utilisés pour l'indexation des documents ne sont pas les mêmes utilisés dans les requêtes des utilisateurs. Le réel besoin informationnel de l'utilisateur est difficile à identifier à partir des requêtes posées. Le besoin informationnel est défini par le thème de recherche de l'utilisateur mais aussi par la nature des résultats attendus (on cherche un document spécifique, une adresse spécifique, des documents sur un thème, ou encore une recherche exploratoire). A tout cela s'ajoute le problème de la disparité des qualités des ressources indexées sur le Web.

Différents travaux se sont intéressés récemment au problème du tri de résultats de recherche. Nous les classifions selon les trois axes suivants :

1. Approches fondées sur l'exploration de la structure du Web. L'idée est d'inférer une mesure de la qualité (ou de l'autorité) d'une ressource à partir des liens hypertextes entrants et sortants qui la relie aux autres ressources (e.g pages web). Les ressources qui ont une grande autorité sont alors placées en tête de la liste des résultats. L'exemple le plus connu de ces approches est l'algorithme *PageRank* employé par le moteur de recherche *Google* (Brin et. al. , 1998). D'autres exemples sont : l'approche *Salsa* (Lemep et.al. 2000), l'algorithme *HITS* (Ding, 2002), l'approche de Kleinberg (Keleinberg, 1999) et l'algorithme baysien proposé dans (Borodin et. al., 2002).
2. Approches fondées sur l'exploration des données d'usage. L'idée ici est d'améliorer le tri en analysant le comportement de l'utilisateur et d'inférer une sorte de *profil utilisateur* qui sera utilisé pour trier les résultats retournés par les moteurs de recherche. Des exemples de telles approches sont décrits dans (Chen et.al. , 2000), (Arezki et. al., 2004).
3. Approches coopératives ou approches orientées communauté d'utilisateurs. L'idée est d'appliquer des techniques de filtrage coopératif pour le tri des résultats. Un premier exemple est le système proposé dans (Chidlovski et. al., 2000) où les auteurs proposent un couplage entre un méta-moteur de recherche et un système de recommandation explicite de documents. Les recommandations faites dans ce dernier système sont utilisées pour trier les résultats retournés par le méta-moteur de recherche. Un autre exemple est le système I-SPY (Freyen et. al., 2004).

Dans ce papier nous proposons une approche qui combine l'exploration des données d'usage et l'approche coopérative. L'idée est de permettre à un groupe d'utilisateurs qui ont des centres d'intérêts communs (e.g. une équipe de R&D) de partager d'une manière implicite leurs expériences de recherche d'information. D'autres systèmes d'aide à la recherche d'information ont adopté une telle approche : (Kanawati et. al, 1999), (Trousse et. al., 1999). Ces systèmes sont construits selon une architecture centralisée. Or la centralisation des données d'usage pose des problèmes de sécurité et de protection des vies privées des utilisateurs. Dans notre système nous proposons une architecture complètement décentralisée où chaque utilisateur sauvegarde sur sa machine locale, ses propre trace d'usage. Le partage d'expérience est fait par des agents logiciels assistant interposés.

Le système proposé fonctionne comme suit : chaque utilisateur est associé à un agent assistant personnel. Cet agent trace dans un fichier les requêtes posées et les sélections des documents faites par l'utilisateur. L'agent intercepte les résultats retournés par un moteur de recherche et demande aux autres de reclasser ces résultats en leur envoyant la requête q et les résultats obtenus R. Chaque agent cherche dans sa base de traces des requêtes similaires à q et utilise les sélections faites par l'utilisateur pour proposer un nouveau classement de la liste R. La méthodologie de raisonnement à partir de cas (RàPC) (Aamodt et. al., 1994) est employée à cet effet. L'agent

demandeur reçoit les différents classements proposés par les différents agents et infère le classement à présenter à l'utilisateur.

La suite de l'article est organisée comme suit. La section 2 décrit l'architecture générale du système. L'architecture interne d'un agent assistant est détaillé et la constitution de la base de trace est expliquée. L'algorithme de tri est détaillé en section 3. Le cycle RàPC employé par chaque agent est décrit en détails. Les premiers résultats expérimentaux sont rapportés dans la section 4. Finalement une conclusion est donnée en section 5.

2 Architecture générale du système

Le système proposé est construit selon une architecture multi-agents de type égal-à-égal (Peer-To-Peer). Chaque utilisateur est associé à un agent assistant personnel. Un agent d'enregistrement est gère l'adhésion des agents assistants au groupe.

Un agent assistant est composé de quatre principaux modules : un module *d'interaction avec l'utilisateur*, un module de *gestion de la communauté*, un module de *communication* et un module de *tri*. Le fonctionnement du denier module représente la contribution principale de ce papier et sera détaillé en section 3. Le module de communication offre les facilités classiques de communication entre agents. Le rôle du module de gestion de la communauté est de permettre à l'agent de choisir un sous-ensemble des agents les plus compétents pour lui fournir de l'aide. Dans la version actuelle du système aucune stratégie de formation de communauté n'est implémentée. Chaque agent diffuse sa demande d'aide à tous les autres agents. Le module d'interaction avec l'utilisateur peut être vu comme un *proxy* qui s'interpose entre l'utilisateur et les moteurs de recherche utilisés. Son rôle est a) d'observer et de tracer le comportement de l'utilisateur, 2) de notifier aux autres modules les événements nécessaires pour leur fonctionnement et 3) de transférer les requêtes utilisateurs aux moteurs de recherche, et de présenter les résultats à l'utilisateur après avoir été triés par le module du tri.

Les traces du comportement de l'utilisateur sont enregistrées dans un fichier de traces (log). Ce fichier est composé d'un ensemble d'enregistrements. Un enregistrement contient les informations suivantes :

1. Identificateur de requête (*Qid*). C'est un identificateur unique attribué par le module d'interaction à chaque nouvelle requête soumise par l'utilisateur.
2. La requête (Q). Dans la version actuelle, nous considérons des requêtes simples composées de listes de mots clés.
3. La liste des résultats (R) : C'est la liste des documents retournés par le moteur de recherche en réponse à la requête Q. Chaque document est représenté par un couple : adresse du document (i.e. URL) et un vecteur de mots clés représentant le document. Ce vecteur est extrait automatiquement à partir du résumé envoyé par les moteurs de recherche.
4. La sélection (S). C'est une sous-liste ordonnée de R qui représente les documents sélectionnés par l'utilisateur. Nous faisons l'hypothèse qu'un document

sélectionné par l'utilisateur est jugé pertinent pour la requête Q. Cette hypothèse est justifiée par le fait que l'utilisateur lit d'abord le résumé avant de sélectionner un document.

Un nouvel enregistrement est crée pour chaque requête. L'enregistrement est mis à jour avec chaque sélection d'un nouveau document appartenant à la liste R. Un enregistrement sera fermé si aucune mise à jour n'est effectuée pendant un certain période de temps. Le fichier de trace fournit les données brutes qui seront utilisées par le module de tri pour calculer le classemen des résultats de nouvelles requêtes. Ce calcul est expliqué dans la section suivante.

3 Approche de tri coopératif

Présentation générale

Etant donné une requête Q et une liste de résultats R retournée par un moteur de recherche, l'objectif du module de tri est de trouver une permutation R^* de R de sorte que les documents les plus pertinents pour l'utilisateur soient placés en tête de la liste R^*. L'approche que nous proposons est la suivante : pour chaque couple $<Q, R>$, l'agent assistant A_i explore sa propre base de traces afin de calculer une permutation R_i^*. En même temps l'agent A_i demande aux autres agents A_j de lui proposer des permutations R_j^*. La permutation R* est donnée par une combinaison linéaire des toutes les permutations calculées et la liste originelle R. La combinaison des listes permutées est un processus similaire à celui appliqué classiquement par les méta-moteurs de recherche afin de fusionner les résultats retournés par les différents moteurs de recherche (Dwork et. al, 2001) (McCabe et. al., 1999). Noter que le processus ici est plus simple puisque toutes les listes contiennent les mêmes documents (mais dans des ordres éventuellement différents) et on n'a pas à traiter des évaluations hétérogènes des documents.

Pour calculer une permutation R_k^* un agent A_k applique la méthodologie du raisonnement à partir de cas (RàPC). Le principe du RàPC est de résoudre un problème, appelé aussi *cas cible*, en réutilisant des solutions éprouvées lors de la résolution des problèmes similaires dans le passé. L'expérience de résolution de problèmes passés est stockée dans une base de cas dits *cas sources*. Un cas source, dans sa forme la plus simple est composé d'un couple <problème, solution>. Le cycle RàPC est composé de quatre étapes : une étape de *recherche* qui permet de retrouver les k cas sources les plus similaires au cas cible, une phase de *réutilisation* dont l'objectif est d'adapter les solutions des cas sources retrouvés afin de proposer une solution au cas cible. La solution calculée peut être révisée durant la phase de *révision*. Et enfin, le nouveau cas résolu peut être ajouté à la base de cas, pendant la dernière phase dite *d'apprentissage*, afin d'enrichir l'expérience du système.

Dans notre système un cas cible est tout naturellement composé du couple $<Q,R>$: la requête soumise par l'utilisateur et la liste des réponses retournées par le moteur de recherche. Les cas sources seront extrait des données de traces décrites en

section 2. Dans les sections suivantes nous dérivons la structure et la procédure d'extraction des cas sources. Ensuite nous décrivons les deux premiers phases du cycle RàPC.

Structure et extraction des cas sources

Chaque enregistrement fermé *Rec* dans le fichier de traces peut engendrer au plus un cas source. Un cas *c* est composé des attributs suivants :

1. La requête (noté $c.Q$). C'est une copie de l'attribut requête de l'enregistrement *Rec*. ($c.Q$=$Rec.Q$).
2. Le tri (noté $c.K$). C'est la liste des documents retournés par le moteur de recherche en réponse à la requête Q où les documents sélectionnés par l'utilisateur sont placés en tête (dans l'ordre de sélection). Formellement $c.K = Rec.S \oplus [Rec.R - Rec.S]$ où $\oplus$ est l'opérateur de concaténation de listes.

Lorsque un enregistrement est déclaré fermé, un cas source sera immédiatement extrait et rajouté à la base de cas. Cependant une attention particulière doit être faite afin de ne pas augmenter la taille de la base de cas par l'ajout des cas inutiles. En effet, la maintenance de la base de cas est un aspect central dans tout système de RàPC (Leake & Wilson, 2002). Mais la discussion de cet aspect dépasse le cadre de ce papier. Des heuristiques simples sont proposées afin de limiter l'ajout des cas inutiles. Aucun cas source n'est extrait dans les situations suivantes :

1. L'enregistrement a une liste de résultats vide..
2. L'enregistrement a une liste de sélections vide
3. La sélection est un préfixe de la liste de résultats. Autrement dit, l'utilisateur semble approuver le classement fait par le moteur de recherche.

La phase de recherche

L'objectif de cette phase est de retourner les k cas sources les plus similaires à un cas cible. k étant un paramètre du système. Afin d'accélérer la sélection des cas sources nous proposons un algorithme de recherche qui opère en deux étapes :

Etape 1. Lorsque l'utilisateur soumet une requête Q, l'agent local envoie cette requête aussi aux autres agents. Chaque agent chercher dans sa base locale les cas sources dont la similarité de l'attribut requête avec Q dépasse un certain seuil σ_q. La similarité entre deux requêtes $q1$ et $q2$ est donnée par la fonction : $Sim_{query}(q1, q2) = ||q1 \cap q2|| / ||q1 \cup q2||$. On désigne par Γq l'ensemble des cas sources retenus à l'issue de cette étape. Noter que cette étape se fait en parallèle de chargement de la liste des réponses à partir d'un moteur de recherche.

Etape 2. A l'arrivée de la liste de réponse R, les cas retenus dans Γq seront examinés pur déterminer les cas les plus similaires au problème <Q, R> en utilisant

la similarités entre les listes de réponses R et $c.K$. La similarité entre deux listes de documents $R1$ et $R2$ est donnée par la somme des similarités entre chaque document de $R1$ et chaque document de $R2$. La similarité est normalisée en divisant la somme obtenue par la somme des tailles de $R1$ et $R2$. La similarité entre deux documents est donnée par la fonction suivante :

Sim_{doc} *(d1, d2) :*
Si URL(d1)== URL(d2) Alors
Return 1
Sinon
Return $||Contenu(d1) \cap Contenu(d2)|| / ||Contenu(d1) \cup Contenu(d2)||$

Où *URL(d)* est une fonction qui retourne l'adresse du document d et *contenu(d)* retourne le vecteur de mots clés décrivant le document d. Les k cas les plus similaires seront alors retournés par la phase de recherche et seront utilisés par la phase suivante. On désigne l'ensemble des cas sources retournés par ΓR

La phase de réutilisation

Etant donné un cas cible composé d'un couple requête, liste de résultat <Q, R> et un ensemble de cas sources remémorés durant la phase de recherche ΓR, l'objectif de la phase de réutilisation est de calculer une permutation R_k* où les documents de R qui sont jugés être les plus pertinents pour l'utilisateur sont placés en tête.

L'idée de base est de permettre à chaque cas source $c \in \Gamma R$ de voter sur l'ordre relative de chaque couple de documents d_i, $d_j \in R$. Pour faciliter la compréhension du principe de vote, considérions la situation idéale où un couple d_i, d_j figure aussi dans la liste c.K. Le vote du cas c concernant ce couple est donné par la formule :

$$vote_c(d_i, d_j) = rang(d_i, c.K) - rang(d_j, c.K)$$

où *rang(d, c.K)* retourne le rang du document d dans la liste ordonnée $c.K$.

Dans la réalité, les documents de R ne figurent pas forcément dans les listes $c.K$ de tous les cas sources $c \in \Gamma R$. Pour résoudre ce problème, avant de calculer le vote d'un cas c sur les couples d_i, $d_j \in R$, nous commençons par associer à chaque document $d \in R$ un document $d^h \in c.K$ appelé document homologue. Par définition le document homologue à un document d est le document le plus similaire à d dans $c.K$. La similarité entre documents est calculée par la même fonction définie en section 3.3. Un seuil de similarité minimal σh doit être satisfait par les documents homologues. Si plusieurs documents dans $c.K$ sont à égale distance de d, le document homologue sera le document le plus haut placé dans $c.K$. Si aucun document homologue à d_i n'est trouvé dans la liste $c.K$ alors le cas c ne vote pas sur l'ordre de n'importe couple impliquant le document d_i

Pour calculer la permutation R_k* l'agent A_k applique l'algorithme suivant : La liste R_k* est initialisé à [d_1] où d_1 est le document en tête de R. Pour chaque document d_i suivant dans R on calcule la somme des votes des cas $c \in IR$ avec les documents d_k dans R_k*. Trois cas de figure peuvent avoir lieu :

1. La somme des votes est négative. Dans ce cas le document d_i est inséré avant le document d_k dans R_k*.
2. La somme des votes et nulle. Dans ce cas d_i est inséré immédiatement après d_k dans R_k*.
3. La somme des votes est positive. Dans ce cas, on calcule la somme des votes sur couple d_{k+1} et d_i et on applique à nouveau les mêmes règles.
4.

L'exemple suivant illustre l'algorithme proposé. Considérons une requête Q à laquelle un moteur de recherche répond par une liste R de 5 documents R= {d1, d2, …, d5}. Si aucun cas similaire au cas cible <Q, R> n'est trouvé par l'agent A_k, la permutation R_k* sera tout simplement la liste R elle-même. Autrement dit l'agent ne change pas l'ordre donné par le moteur de recherche. Maintenant, considérons la situation où l'utilisateur choisit parmi les documents proposés le document d5. Un cas source sera extrait dont la partie c.K est égale à c.K= {d5,d1,d2,d3,d4}. Si l'utilisateur soumet la même requête et le moteur répond par la même liste R. le cas source c sera retourné par la phase de recherche (similarité égale à 1). Les votants sont le moteur de recherche et le cas c. Les quatre premiers documents seront dans le même ordre que dans R puisqu'ils ont les mêmes ordres à la fois dans R et dans c.K. Par contre pour le document d5, le cas c le place avant d1 et le moteur le place après. La somme des votes est égale à :

$$Votes(d5,d1) = rang(R,d5) - rang(R,d1) + rang(c.K,d5)\ \ rang(c.K, d1) = 1.5 > 0$$

Donc un deuxième vote sur le couple d5 et d2 doit avoir lieu. La somme des votes Votes(d5,d2) = 0.5. En conséquence il faut voter sur l'ordre du couple (d5, d3). Ici Votes(d5, d3) = -0.5. Par conséquent, le document d5 sera inséré dans la liste R_k* entre d2 et d3. la liste R_k* est égale à R_k*={d1, d2, d5, d3, d4}.

4 Expérimentation

Afin de valider notre approche nous avons exécuter l'algorithme sur n jeux de données fictives. Nous avons généré un ensemble D de 5000 documents. Chaque document est représenté par 1) un identificateur qui joue le rôle de l'adresse et 2) un vecteur de mots clés. Les tailles des vecteurs des mots clés varient aléatoirement entre 5 et 15 mots. Les mots clés sont sélectionnés aléatoirement parmi 2000 termes (représentés par des entiers). Pour une requête Q, notre simulateur de moteur de recherche retourne une liste de au plus 10 documents. La pertinence d'un document d pour la requête Q est tout simplement mesuré par : Pertinence (d,Q) = $||d \cap Q||/||d \cup Q||$. Par contre au lieu de renvoyer les listes de réponses dans l'ordre de pertinence,

on envoie une liste de 10 documents dont les 5 meilleurs sont placés dans la deuxième moitié de la liste. On simule que l'utilisateur sélectionne systématiquement trois des cinq meilleurs documents. Nous avons calculé combien de requêtes similaires faut-il avoir pour retrouver les cinq meilleurs documents en tête de la liste. Nous avons varié le seuil de similarité de 1 à 0.7. L'expérimentation est répétée dix fois et les valeurs moyennes sont relevées dans le tableau suivant :

Similarité	1	0.9	0.8	0.
Nombre de requêtes	4	5	9	12

Les résultats montrent qu'il suffit de poser la même requête 4 fois pour retrouver les meilleurs documents en tête du classement. Si la similarité des requêtes posées ne dépassent pas 0.7 il faut environ 12 requêtes pour avoir le même résultat. Ces résultats sont encourageants mais il faut évidemment valider le système dans des situations d'utilisation réelles où les utilisateurs ne sont pas toujours capables de sélectionner systématiquement les bons documents.

5 Conclusion

Ce papier décrit une approche d'aide au tri des résultats de moteurs de recherche sur le Web. L'approche présentée combine l'exploration, ou le fouille des données d'usage avec une approche coopérative. Peu de travaux dans la littérature ont abordé ce problème avec une approche similaire. Une exception est le système I-SPY (Freyen et. al., 2004). La différence principale entre notre système et I-SPY est que ce dernier est construit selon une architecture centralisée où les données d'usage de tous les utilisateurs sont groupés et traités sur une même machine. Des problèmes de performances, de et de protection de la confidentialité ne sont pas abordés. En outre, le système I-SPY ne modifie pas le classement des résultats que pour les requêtes qui sont déjà soumises telle qu'elles (similarité égale à 1). Outre la validation dans des situations réelles d'usage, plusieurs problèmes intéressants restent à résoudre. Nous citons en particulier le problème de formation de communauté, et le problème de prise en compte de la co-existence au sein du groupe des utilisateurs qui ont des comportements et des objectifs différents tout en ayant les mêmes centres d'intérêts. Ces deux problèmes sont sur notre agenda de travaux futurs

Références

AAMODT A., PLAZA E. (1994), Case-based Reasoning: Foundational issues, Methodological variations and system approaches. AI communications 7(1):39-59

AREZKI, R. PONCELET P., DRAY G. AND PEARSON D.W. (2004). PAWebSearch: An Intelligent Agent for Web Information Retrieval. In proceedings of the International Conference on

Advances in Intelligent Systems: Theory and Applications (AISTA 2004), Luxembourg, November 15-18, 2004.

BRIN S. AND PAGE L. (1998) The anatomy of large scale hypertextual web search engine. In proceedings of th 7th International conference on the world wide web. Brisbane, Australia.

BORODIN A. ET. AL., (2002) Finding authorities and hubs from link structures on the world wide web. In proceedings of the 10th International conference on the word wide web, Hong Kong

CHEN Z AND MENG X. (2000) Yarrow: real-time client side Meta-search learner. In proceedings of the AAAI workshop on artificial intelligence for web search (AAAI'00). 12-17 July Austin. AAAI press pp. 12-27

CHIDLOVSKI B. ET; AL. (2000) Collaborative Re-Ranking of search results. In proceedings of the AAAI workshop on artificial intelligence for web search (AAAI'00). 12-17 July Austin. AAAI press pp. 18-22

DING C. (2002) PageRank, HITS and a unified framework for link analysis. In proceedings of the 25th AM SIGIR conference, Tampere, Finland

DWORK C. ET. AL. (2001) Rank aggregation methods for the web. In proceedings of the 10th International conference on the word wide web, Hong Kong pp. 613-622

FREYEN J., SMYTH B. COLE M. BALLE EVELN AND BRIGGS P.(2004) Further Experiments on Collaboration Ranking in Community based Web search. Artificial Intelligence Review, 21(3-4), pp. 229-252.

KANAWATI R., JACZYNSKI, M., TROUSSE B., ANDREOLI J.M. Applying the Broadway Recommandation Computation Approach for Implementing a Query Refinement Service in the CBKB Meta-search Engine, Conférence française sur le raisonnement à partir de cas (Ra`PC'99), Palaiseau, France, juin, 1999.

KEINBERG J. (1999) Authoritative links in a hyperlink environment. Iin journal of the ACM (JACM)44.

LEAKE D.,WILSON D.C. (2002) Maintaining case-based reasoners: diemensions and dierections Computational Intelligence 27(2)

LEMPLE R., MORRIN S. (2000) The stochastic for link structure analysis SALSA and the TKC effect. In proceedings of the 9th International conference on the word wide web

MCCABE M. ET. AL. (1999). A unified environment for fusion of information retrieval approaches. In ACM CIKM conference pp. 330-334

TROUSSE B., JACZYNSKI M., KANAWATI R., Using User Behavior Similarity for Recommandation Computation: The Broadway Approach, 8th international conference on human computer interactions (HCI'99), Munich, August, 1999.