

# Réflexions sur l'apport de l'exploration des traces d'usage pour améliorer le tri des résultats des moteurs de recherche sur le Web

Rushed Kanawati

LIPN – CNRS UMR 7030  
99 Av. J.B. Clément 93430 Villetaneuse  
rushed.kanawati@lipn.univ-paris13.fr  
<http://www-lipn.univ-paris13.fr/~kanawati>

**Résumé.** Nous présentons dans ce papier un système de fouille coopérative de données d'usage de moteurs de recherche sur le Web dont l'objectif est d'améliorer le tri des résultats rendus par un moteur de recherche. Le système est construit selon une architecture multi-agents où chaque utilisateur est assisté par un agent personnel. Les agents coopèrent entre-eux et utilisent la méthodologie du raisonnement à partir de cas pour re-trier les résultats rendus par un moteur de recherche. Nous nous servons de ce système pour 1) présenter notre analyse des choix de conception d'un système d'exploration coopérative de données d'usage du Web et 2) montrer les problèmes qui restent à résoudre et l'apport attendu des techniques de fouille de données d'usage pour les résoudre.

## 1 Introduction

La présentation et le tri des résultats des moteurs de recherche 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. La contre performance des moteurs existant est le meilleur témoin de la nécessité de nouvelles approches de tri de résultats. 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 (Brin & Page, 1998) (Ding, 2002).
2. Approches fondées sur l'exploration des données d'usage (Chen & Meng, 2000) (Arezki et al., 2004).
3. Approches coopératives ou approches orientées communauté d'utilisateurs (Chidloveski et. al., 2000).

Nous nous intéressons à une approche hybride qu'on qualifie de fouille coopérative de données d'usage. L'idée de base est de permettre à un groupe d'utilisateurs de partager implicitement leurs expériences en recherche d'information (Trousse et. al., 1999), (Kanawati, 2003), (Freyne, et. al, 2004). Un premier prototype d'un système d'agents assistants d'aide au tri des résultat est proposé. Ce système est construit selon une architecture égal-à-égal (Peer to Peer). Les agents assistants utilisent la méthodologie du

raisonnement à partir de cas pour calculer les recommandations. Nous nous servons de ce système pour :

1. Présenter notre analyse des choix de conception d'un système d'exploration coopérative de données d'usage du Web.
2. Montrer les problèmes qui restent à résoudre et l'apport attendu des techniques de fouille de données d'usage pour les résoudre.

La suite de l'article est organisée comme suit : Dans la section 2, nous présentons l'idée générale de l'approche proposée et nous discutons les principaux choix de conception. Une première évaluation du système proposé est donnée dans la section 3. Nos réflexions sur l'apport possible des techniques d'exploration des données d'usage pour améliorer le système sont rapportées dans la section 4. Une conclusion est donnée dans la section 5.

## 2 Tri coopératif

### 2.1 Principe

Etant donné une requête  $Q$  et une liste de résultats  $R$  retournée par un moteur de recherche, notre objectif est de calculer 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'idée que nous proposons est de garder en mémoire les traces d'usage du moteur de recherche par l'utilisateur, notamment les requêtes posées dans le passé, et l'ordre des documents consultés par l'utilisateur parmi l'ensemble des résultats retournés par le moteur de recherche. En fait, comme la plupart des moteurs de recherche existant retourne une liste de résumés de documents, nous faisons l'hypothèse que les documents sélectionnés par l'utilisateur sont jugés pertinents par rapport à la requête posée. Donc pour un couple  $\langle Q, R \rangle$ , nous cherchons dans la mémoire du système (les données de trace) les couples similaires et nous trions la liste  $R$  en fonction des documents sélectionnés enregistrés dans les couples  $\langle Q_i, R_i \rangle$  remémorés. La méthodologie du raisonnement à partir de cas est utilisée à cet effet (Aamodt et Plaza, 1994). 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  $\langle \text{problème}, \text{solution} \rangle$ . Dans notre application un cas cible est tout naturellement composé du couple  $\langle Q, R \rangle$  : la requête soumise par l'utilisateur et la liste des réponses retournées par le moteur de recherche. Un cas source  $c$  est donné par un couple  $\langle c.Q, c.K \rangle$  ou  $c.Q$  est une requête et  $c.K$  est la liste de résultats retournée en réponse à  $c.Q$  après avoir placé les documents sélectionnés par l'utilisateur en tête.

Lorsqu'un nouveau cas cible se présente la première phase du cycle RàPC a pour rôle de remémorer les cas sources les plus similaires au cas cible. Faute de place nous ne détaillons pas ici les mesures de similarités utilisées ni les schémas d'indexation des cas. Une description détaillée du cycle RàPC employé est donnée dans (Kanawati, 2005). Soit  $\Gamma R$ , l'ensemble des cas sources remémorés à l'issue de la phase de recherche. L'objectif de la phase suivante, la phase de réutilisation, est de calculer une permutation  $R^*$  de  $R$  en fonction de  $\Gamma R$ . 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,

considérons la situation idéale où un couple  $d_i, d_j$  figure aussi dans la liste  $c.K$ . Le vote du cas  $c$  est donné par :  $vote_c(d_i, d_j) = rang(d_i, c.K) - rang(d_j, c.K)$  où  $rang(d, c.K)$  est le rang du document  $d$  dans la liste ordonnée  $c.K$ .

Dans le cas général, 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, et avant de calculer le vote d'un cas  $c$  concernant 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  $c.K$  selon la mesure de similarité employé dans la phase de recherche. Un seuil de similarité minimal  $\sigma_h$  doit être satisfait par les documents homologues. 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 tout couple de documents impliquant  $d_i$ .

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

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

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  $R = \{d1, d2, \dots, d5\}$ . Si aucun cas similaire au cas cible  $\langle Q, R \rangle$  n'est trouvé, la permutation  $R^*$  sera tout simplement la liste  $R$  elle-même. Autrement dit l'ordre donnée par le moteur de recherche ne sera pas changée. 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 :  $c.K = \{d5, d1, d2, d3, d4\}$ . 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. 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 moyenne des votes est égale à :

$$Votes(d5, d1) = [rang(R, d5) - rang(R, d1) + rang(c.K, d5) - rang(c.K, d1)] / 2 = 1.5 > 0$$

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

## 2. 2 Choix de conception

Les performances de l'approche proposée ci-haut dépend largement de la qualité de la base de cas employée. En effet, pour pouvoir trier les résultats d'une requête  $Q$  il faut disposer dans la mémoire du système des requêtes similaires à  $Q$ . Pour augmenter la chance

d'avoir des requêtes similaires aux requêtes posées par un utilisateur, un choix évident est de permettre à un groupe d'utilisateurs, qui ont des centres d'intérêts communs, de partager leurs expériences de recherche d'information. Or, parlant de partage de traces d'usage du Web, des problèmes évidents de confidentialité et de protection de la vie privée se posent. Plusieurs solutions sont proposées dans la littérature pour le partage de données d'usage du Web. Une première solution classique est de passer par un *proxy* Web auquel se connectent les utilisateurs voulant partager les traces d'usage (Trousse et. al., 1999). Les utilisateurs ont le choix de se connecter au Web via ce proxy ou non. Cette solution est simple à mettre en oeuvre mais le schéma de partage de données est aussi simpliste. L'utilisateur a le choix de rendre ses données complètement privées, ou alors publiques. Or dans beaucoup de scénarios coopératifs on a besoin d'avoir des règles d'accès aux données partagées de granularité fin. En plus, avoir les données de traces sur une machine autre que celle de l'utilisateur complique la modification du schéma de partage.

Une deuxième solution proposée dans (Freyne et. al., 2004) consiste à créer des communautés thématiques, à l'instar des forums de discussions. Pour chaque communauté une page Web spécifique est créée permettant de poser des requêtes. Chaque page est connectée à un proxy spécifique à la communauté qui garde la trace d'usage des participants. Cette solution est certes plus souple que la première mais présente l'inconvénient de demander à l'utilisateur une surcharge additionnelle qui consiste à chercher la bonne communauté pour y poser sa requête. En plus il faut prévoir des mécanismes de création et de gestion des communautés thématiques.

La solution que nous proposons consiste à permettre à chaque utilisateur de garder ses propres traces d'usage localement. Un agent logiciel personnel se charge de cette tâche. Si l'utilisateur a besoin d'aide l'agent personnel pourra demander cette aide des autres agents. Chaque agent est maître et pourra accepter ou décliner les requêtes reçues. Chaque agent assistant exécute localement le cycle RàPC décrit sommairement en section 2.1 (voir détails en (Kanawati, 2005)) et pourra demander l'aide des autres agents. Plusieurs questions se posent ici : Quand demander de l'aide ? De qui ? Et quelle est la nature de l'aide à fournir ?

En ce qui concerne la dernière question, trois types d'aide sont envisageables (Malek & Kanawati, 2001) : 1) envoyer à l'agent demandeur des cas sources similaires au problème posé (i.e. cas cible), 2) envoyer des données brutes et 3) envoyer le résultat de la résolution du cas cible reçu. D'un point de vue de confidentialité, la troisième solution est la plus sûre. Mais une étude détaillée doit être menée pour comparer ces choix en fonction de leur influence sur la qualité des résultats rendus par le système (Plaza & Ontanon, 2002).

Dans l'actuelle version du prototype, nous avons opté pour la troisième solution mais nous travaillons sur une étude comparative des trois solutions proposées. Les deux autres questions (quand ? Et qui?) ne sont pas traitées pour le moment. Nous avons opté pour une demande systématique de l'aide de tout agent disponible (diffusion de la requête). Mais nous sommes conscients de l'importance de traitement de ces deux points.

### 3 Expérimentation

Afin de valider notre proposition nous avons exécuté l'algorithme du tri proposé sur  $n$  jeux de données fictives. Nous avons généré un ensemble  $D$  de 5000 documents. Chaque document est représenté par un identificateur qui joue le rôle de l'adresse et 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 mesurée par :

$$Pertinence(d, Q) = ||d \cap Q|| / ||d \cup Q||$$

Par contre au lieu de renvoyer la liste 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	0.7
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.

## 4 Discussion

L'approche proposée repose sur l'hypothèse qu'un document sélectionné par un utilisateur parmi une liste fournie en réponse à une requête  $Q$  est un document pertinent pour  $Q$ . Cette hypothèse sous-entend que les documents non sélectionnés seront moins ou non pertinents. Or en réalité, le comportement de l'utilisateur dépend de son objectif de recherche. Le même outil de recherche est utilisé différemment si 1) on cherche une ressource particulière, 2) on cherche des documents dans un thème 3) on effectue une tâche de veille 4) on effectue une recherche exploratoire, etc. Or l'objectif de l'utilisateur n'est pas explicité dans les fichiers des traces. Un problème intéressant à étudier est d'essayer de reconnaître l'objectif de recherche en explorant les traces d'usage. L'interprétation des traces d'usage sera alors dépendante de la nature de l'objectif.

Une piste qui nous semble intéressante est d'explorer le dynamique du comportement de l'utilisateur, autrement dit d'essayer de reconstruire à partir des données d'usage de *sessions* de recherche d'informations faites par l'utilisateur. Une session peut être de durée variable, plusieurs jours voire plusieurs semaines (en cas de tâche de veille). Ceci rend l'identification de sessions une tâche difficile mais ça reste un défi intéressant.

Un autre point important concerne la formation automatique et dynamique des communautés d'intérêts. Comment à partir de traces d'usage mais aussi à partir des traces du raisonnement faits par les agents reconnaître les communautés existantes au sein d'un groupe d'utilisateurs ?

Enfin, un problème déjà abordé concerne le choix de niveaux de partage d'expérience entre les utilisateurs : on partage des données, des cas ou des raisonnements ? Autant de problèmes intéressants qui font appel à la communauté de recherche en exploration de données d'usage !

## 5 Conclusion

Un système de tri coopératif de résultat de moteur de recherche sur le Web a été présenté. Le système est construit selon une architecture multi-agents où chaque utilisateur est assisté par un agent personnel qui garde trace de l'activité de l'utilisateur et utilise ces données de trace pour le tri des résultats de nouvelles requêtes. Une stratégie de coopération entre les agents est présentée pour améliorer le tri. La description du système a servi comme support pour poser les principales questions que soulève la mise en oeuvre d'un système de fouille coopérative de données d'usage. Quelques axes de recherche qui nous semble intéressants sont aussi mentionnés.

## Références

- A. Aamodt, E. Plaza (1994), Case-based Reasoning: Foundational issues, Methodological variations and system approaches. *AI communications* 7(1):39-59
- R. Arezki, P. Poncelet, G. Dray and D.W. Pearson (2004). PAWebSearch: An Intelligent Agent for Web Information Retrieval. In proceedings of the International Conference on Advances in Intelligent Systems: Theory and Applications, Luxembourg, November 15-18, 2004.
- S. Brin and L. Page (1998) The anatomy of large scale hypertextual web search engine. In proceedings of the 7<sup>th</sup> International conference on the world wide web. Brisbane, Australia.
- 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 25<sup>th</sup> AM SIGIR conference, Tampere, Finland
- Freyen J., Smyth B. Cole M. Balle E. 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. (2003). A multi-agent system for improving result ranking service of web search engines. In Proceedings of CE'03: International conference on concurrent engineering, R. Jardim-Gonçalves et. al. (Eds) Madeira, July 2003.
- Kanawati R. (2005). Un système coopératif pour le tri des résultats de moteurs de recherche. Atelier RàPC Plate-forme AFIA Nice Mai-juin 2005.
- Malek, M. Kanawati R. (2001) CBR-based Collaborative Interface agents: Architectural issues. In proceedings of IEEE international conference on computer systems and applications AICCSA'01 ACS/IEEE, 26-29 June, Beirut.
- Plaza E., Ontanon S. (2002) Cooperative Multi-agent Learning. *Adaptive Agents and Multi-Agent Systems: Adaptation and Multi-Agent Learning*. LNCS 2636 Springer 2003,
- Trousse B., Jaczynski M., Kanawati R. (1999), Using User Behavior Similarity for Recommendation Computation: The Broadway Approach, 8th international conference on human computer interactions (HCI'99), Munich, August.