Recommandation Diversifiée et Distribuée pour les Données Scientifiques

Maximilien Servajean

Collège Doctoral du Languedoc-Roussillon

THÈSE

Pour obtenir le grade de

Docteur

Délivré par l'**Université Montpellier II**

Préparée au sein de l'école doctorale **I2S***
Et de l'unité de recherche **UMR 5506**

Spécialité : **Informatique**

Présentée par **Maximilien Servajean**
servajean@lirmm.fr

Recommandation Diversifiée et Distribuée pour les Données Scientifiques

Doit être soutenue le 16/12/2014 devant le jury composé de :

Pierre Sens, PR, Université de Paris 6	Rapporteur
Mohand Boughanem, PR, Université de Toulouse	Rapporteur
Sihem Amer-Yahia, DR, CNRS, Université de Grenoble	Examinatrice
Anne Laurent, PR, Université de Montpellier 2	Examinatrice
Reza Akbarinia, CR, INRIA, Université de Montpellier 2	Examinateur
Pierre Larmande, IR, CIRAD, Montpellier	Invité
Pascal Neveu, IR, INRA, Montpellier	Invité
Esther Pacitti, PR, Université de Montpellier 2	Directrice

UNIVERSITÉ MONTPELLIER 2
SCIENCES ET TECHNIQUES

* **I2S :** École doctorale Information Structures Systèmes

Résumé

Dans de nombreux domaines, les nouvelles technologies d'acquisition de l'information ou encore de mesure (*e.g.* serres de phénotypage robotisées) ont engendré une création phénoménale de données. Nous nous appuyons en particulier sur deux cas d'application réels : les observations de plantes en botanique et les données de phénotypage en biologie. Cependant, nos contributions peuvent être généralisées aux données du Web. Par ailleurs, s'ajoute à la quantité des données leur distribution. Chaque utilisateur stocke en effet ses données sur divers sites hétérogènes (*e.g.* ordinateurs personnels, serveurs, *cloud*), données qu'il souhaite partager. Que ce soit pour les observations de botanique ou pour les données de phénotypage en biologie, des solutions collaboratives, comprenant des outils de recherche et de recommandation distribués, bénéficieraient aux utilisateurs.

L'objectif général de ce travail est donc de définir un ensemble de techniques permettant le partage et la découverte de données, via l'application d'approches de recherche et de recommandation, dans un environnement distribué (*e.g.* sites hétérogènes).

Pour cela, la recherche et la recommandation permettent aux utilisateurs de se voir présenter des résultats, ou des recommandations, à la fois pertinents par rapport à une requête qu'ils auraient soumise et par rapport à leur profil. Les techniques de diversification permettent de présenter aux utilisateurs des résultats offrant une meilleure nouveauté tout en évitant de les lasser par des contenus redondants et répétitifs. Grâce à la diversité, une distance entre toutes les recommandations est en effet introduite afin que celles-ci soient les plus représentatives possibles de l'ensemble des résultats pertinents.

Peu de travaux exploitent la diversité des profils des utilisateurs partageant les données. Dans ce travail de thèse, nous montrons notamment que dans certains scénarios, diversifier les profils des utilisateurs apporte une nette amélioration en ce qui concerne la qualité des résultats : des sondages montrent que dans plus de 75% des cas, les utilisateurs préfèrent la diversité des profils à celle des contenus.

Par ailleurs, afin d'aborder les problèmes de distribution des données sur des sites hétérogènes, deux approches sont possibles. La première, les réseaux *P2P*, consiste à établir des liens entre chaque pair (*i.e.* nœud du réseau) : étant donné un pair p, ceux avec lesquels il a établi un lien représentent son voisinage. Celui-ci est utilisé lorsque p soumet une requête q, pour y répondre. Cependant, dans

les solutions de l'état de l'art, la redondance des profils des pairs présents dans les différents voisinages limitent la capacité du système à retrouver des résultats pertinents sur le réseau, étant donné les requêtes soumises par les utilisateurs. Nous montrons, dans ce travail, qu'introduire de la diversité dans le calcul du voisinage, en augmentant la couverture, permet un net gain en termes de qualité. En effet, en tenant compte de la diversité, chaque pair du voisinage a une plus forte probabilité de retourner des résultats nouveaux à l'utilisateur courant : lorsqu'une requête est soumise par un pair, notre approche permet de retrouver jusqu'à trois fois plus de bons résultats sur le réseau.

La seconde approche de la distribution est le multisite. Généralement, dans les solutions de l'état de l'art, les sites sont homogènes et représentés par de gros centres de données. Dans notre contexte, nous proposons une approche permettant la collaboration de sites hétérogènes, tels que de petits serveurs d'équipe, des ordinateurs personnels ou de gros sites dans le *cloud*. Un prototype est issu de cette contribution. Deux versions du prototype ont été réalisées afin de répondre aux deux cas d'application, en s'adaptant notamment aux types des données.

Titre en français

Recommandation Diversifiée et Distribuée pour les Données Scientifiques

Mots-clés

- recherche et recommandation
- diversité des profils
- *top-k*
- pair-à-pair
- multisite

Abstract

In many fields, novel technologies employed in information acquisition and measurement (*e.g.* phenotyping automated greenhouses) are at the basis of a phenomenal creation of data. In particular, we focus on two real use cases: plants observations in botany and phenotyping data in biology. Our contributions can be, however, generalized to Web data. In addition to their huge volume, data are also distributed. Indeed, each user stores their data in many heterogeneous sites (*e.g.* personal computers, servers, cloud); yet he wants to be able to share them. In both use cases, collaborative solutions, including distributed search and recommendation techniques, could benefit to the user.

Thus, the global objective of this work is to define a set of techniques enabling sharing and discovery of data in heterogeneous distributed environment, through the use of search and recommandation approaches.

For this purpose, search and recommendation allow users to be presented sets of results, or recommendations, that are both relevant to the queries submitted by the users and with respect to their profiles. Diversification techniques allow users to receive results with better novelty while avoiding redundant and repetitive content. By introducing a distance between each result presented to the user, diversity enables to return a broader set of relevant items.

However, few works exploit profile diversity, which takes into account the users that share each item. In this work, we show that in some scenarios, considering profile diversity enables a consequent increase in results quality: surveys show that in more than 75% of the cases, users would prefer profile diversity to content diversity.

Additionally, in order to address the problems related to data distribution among heterogeneous sites, two approaches are possible. First, *P2P* networks aim at establishing links between peers (*i.e.* nodes of the network): creating in this way an overlay network, where peers directly connected to a given peer p are known as his neighbors. This overlay is used to process queries submitted by each peer. However, in state of the art solutions, the redundancy of the peers in the various neighborhoods limits the capacity of the system to retrieve relevant items on the network, given the queries submitted by the users. In this work, we show that introducing diversity in the computation of the neighborhood, by increasing the coverage, enables a huge gain in terms of quality. By taking into account

diversity, each peer in a given neighborhood has indeed, a higher probability to return different results given a keywords query compared to the other peers in the neighborhood. Whenever a query is submitted by a peer, our approach can retrieve up to three times more relevant items than state of the art solutions.

The second category of approaches is called multi-site. Generally, in state of the art multi-sites solutions, the sites are homogeneous and consist in big data centers. In our context, we propose an approach enabling sharing among heterogeneous sites, such as small research teams servers, personal computers or big sites in the cloud. A prototype regrouping all contributions have been developed, with two versions addressing each of the use cases considered in this thesis.

Title in English

Diversified and Distributed Recommendation for Scientific Data

Keywords

- search and recommendation
- profile diversity
- top-k
- *P2P*
- multi-sites

Équipe de Recherche

Zenith, INRIA & LIRMM

Laboratoire

LIRMM - Laboratoire d'Informatique, Robotique et Micro-Électronique de Montpellier

Adresse

Université Montpellier 2
UMR 5506 - LIRMM
CC477
161 rue Ada
34095 Montpellier Cedex 5 - France

Table des matières

CHAPITRE 1

Introduction

1.1 Motivations et cas d'application

Les nouvelles technologies d'acquisition de l'information, en biologie ou en botanique par exemple, ou encore de mesure (*e.g.* serres de phénotypages robotisées) ont engendré une création phénoménale de données. Par ailleurs, l'arrivée du Web 2.0 a permis l'élaboration de plateformes de collaboration et de partage dynamique, favorisant le développement de communautés à grande échelle. Nous nous appuyons en particulier sur deux cas d'application réels :

- **Données de botanique :** l'arrivée des réseaux sociaux a permis la création de communautés en ligne structurées d'observateurs de la nature (*e.g.* *e-bird* [180], *xeno-canto* [181], *Tela Botanica* [162], *Pl@ntNet* [77, 130]) qui ont commencé à produire de très grandes collections d'images. Celles-ci sont stockées sur différents sites hétérogènes (*e.g.* PCs, *smartphones*, *clouds*, serveurs). Construire une base de connaissances précises de l'identité, de la distribution géographique et de l'évolution des espèces est essentiel pour un développement pérenne de l'humanité, tout autant que pour la conservation de la biodiversité. Chaque participant à cette création de connaissance partage ainsi bénévolement ses observations de plantes. Cependant, afin de créer réellement de l'information, trois étapes, décrites en Figure 1.1, sont nécessaires :
  1. **Partage :** tout d'abord l'utilisateur doit partager ses observations, incluant la photographie, la position GPS, la date, etc. ;
  2. **Identification :** l'observation passe ensuite par une phase collaborative d'identification, où elle est associée à plusieurs familles, espèces et genres possibles par les utilisateurs de la plateforme de partage ;
  3. **Validation :** une étape de validation fait enfin ressortir son appartenance correcte, grâce aux votes d'utilisateurs. Le tuple *famille, espèce et genre* ayant reçu le plus de votes est sélectionné. Cette phase finale

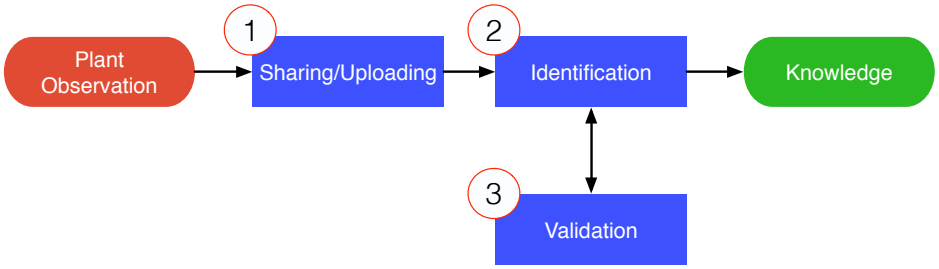


Figure 1.1: Les étapes dans la création de connaissances en botanique.

permet la création d'une connaissance : la plante *xy* a été observée (*i.e.* photographiée) à tel endroit avec telles caractéristiques.

Toutefois, les observations, dans la perspective de leur identification et de leur validation, doivent être re-dirigées vers les bons utilisateurs. Il est nécessaire que ces derniers possèdent la connaissance nécessaire pour achever correctement cette étape. Cependant, ils n'accepteraient pas de devoir uniquement identifier la même plante – ou les mêmes types de plantes. Il s'agit donc de leur retourner un ensemble représentatif et varié des plantes qu'ils peuvent identifier et valider. La Figure 1.2 présente bien ce problème où peu de plantes sont très populaires – et, en conséquence, très souvent recommandées afin d'être identifiées – et nombre d'entre elles le sont moins. À cela s'ajoute un second défi. Un nombre important d'herbiers ou d'observations quelconques est stocké sur les machines propres ou sur des serveurs privés appartenant à divers utilisateurs. Il s'agit donc de leur permettre de rechercher et de se faire recommander des données quels que soient les endroits où elles sont stockées.

- **Données de phénotypage :** chaque être vivant est décrit par son génotype, c'est-à-dire par les caractéristiques *génétiques* qui lui sont propres. Cependant, chacun se comporte différemment en fonction des stress auxquels il est soumis (*e.g.* le soleil ou l'absence de soleil est un stress, et une personne qui y sera soumise bronzera ou pas). L'ensemble des caractéristiques ainsi exprimées par un individu s'appelle *phénotype*. Nous nous intéressons en particulier au phénotypage des plantes. L'objectif de ces recherches est de trouver celles qui résistent plus ou moins bien à certains stress (*i.e.* climats) afin d'adapter les cultures en conséquence. Les recherches en phénotypage se décomposent en trois étapes principales :
  1. **Expérimentation :** les scientifiques établissent un protocole expérimental dans lequel ils définissent un ensemble de stress auxquels doivent être soumises les plantes. Ces dernières sont disposées au sein de serres et des automates mesurent leurs caractéristiques – exprimées en fonction des stress – de phénotypage (cf. Figure 1.3). L'ensemble de ces mesures, ou *données brutes*, est stocké dans des bases de données,

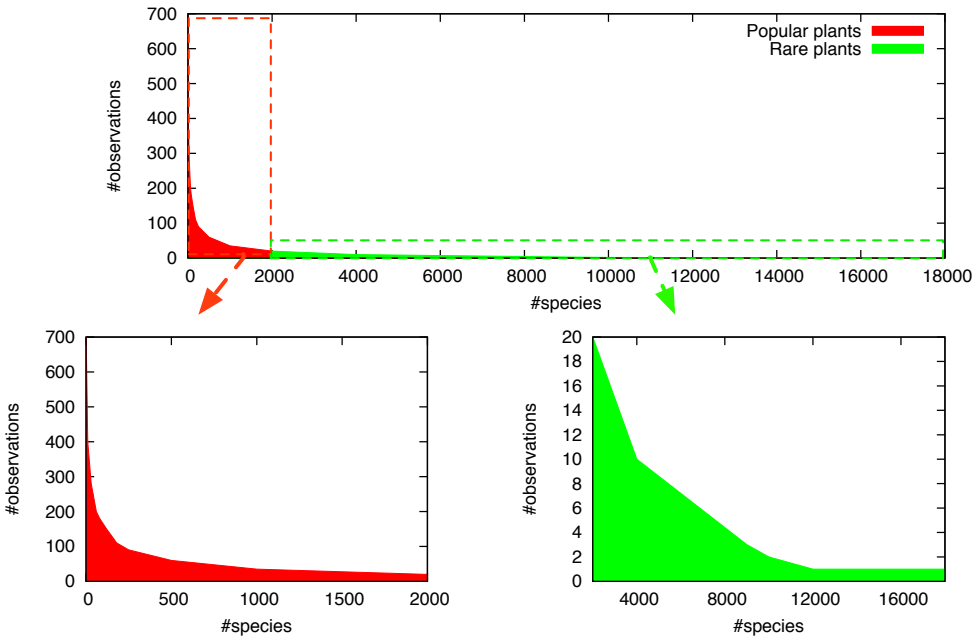


Figure 1.2: Distribution du nombre d'observations de plantes en fonction de leur espèce sur la plateforme *Pl@ntNet*.

mais reste inexploitable : il est trop complexe, taché de bruits, etc. Ce dernier point nous introduit à la seconde étape ;

2. **Transformation des données :** les données brutes, générées précédemment, sont modifiées par un ensemble d'outils mathématiques (*e.g.* régression linéaire, filtrage, moyenne), afin de les rendre compréhensibles et donc utilisables à des fins de recherche. Le résultat de ces opérations, appelé *donnée transformée*, peut être un graphique, un tableau ou toutes autres figures ;
3. **Publication :** enfin, et c'est l'objectif d'un travail de recherche, les résultats scientifiques sont publiés. Ces articles sont textuels mais font référence à toutes les données transformées précédemment créées. Elles y sont directement intégrées au sein de *figures* ou de *tables*, ou référencées par des *URI* ou des *URL*.

Ces trois niveaux sont présentés en Figure 1.4. Lors de travaux de recherche, les scientifiques effectuent presque systématiquement ces trois étapes, alors même qu'un jeu de données répondant à leur problématique peut déjà exister. Ils gagneraient donc à bénéficier d'une méthode collaborative permettant de retrouver les jeux de données transformées pertinents. Puisque ces derniers sont décrits et intégrés dans un contexte plus général au sein d'articles et de documents scientifiques, l'objectif revient à fournir une méthode

Figure 1.3: Serre de phénotypage *phenoarch*.

permettant à chaque chercheur de retrouver les documents scientifiques qui lui conviennent. En d'autres termes, ces derniers doivent à la fois provenir (*i.e.* être partagés) de communautés proches du profil du chercheur, mais également introduire une certaine diversité, afin de rendre possible la mise en relation de jeux de données issus de différentes communautés, toutes pertinentes pour une problématique donnée. Ainsi, un jeu de données utilisé en génétique pourraient intéresser des éco-physiologistes. À cela s'ajoute également le fait que ces données se retrouvent stockées sur divers serveurs d'équipe ou sur les machines propres des chercheurs. Il s'agit donc, comme pour les données de botanique, de leur permettre de rechercher et de se faire recommander des données quels que soient les endroits où elles sont stockées.

1.2 Problématiques

Afin d'aborder nos cas d'application, il est nécessaire d'en définir les points bloquants, ou problématiques. Un premier découle directement des descriptifs précédemment réalisés et peut se formuler de la manière suivante :

Problème 1 : *comment exploiter les techniques de recherche et de recommandation afin que les recommandations ne s'adressent qu'aux utilisateurs pertinents, et qu'elles offrent une nouveauté (i.e. diversité, représentativité) suffisante pour être véritablement intéressantes ?*

En second lieu, et c'est une caractéristique commune aux deux cas d'application, les données ne sont pas disponibles sur un site centralisé. Elles sont souvent stockées sur les machines locales des utilisateurs ou sur divers serveurs centralisés.

Plusieurs raisons peuvent expliquer ce phénomène :

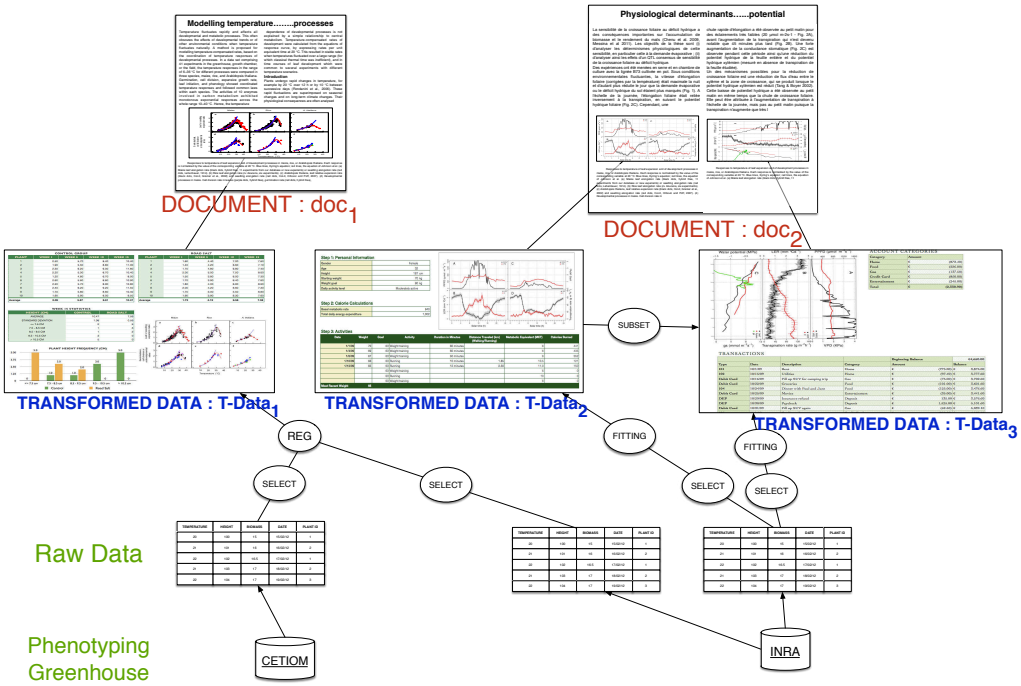


Figure 1.4: Les trois niveaux du contexte applicatif de phénotypage.

1. Le coût que peut avoir la création d'une plateforme centralisée ;
2. Une volonté de contrôle des utilisateurs sur leurs données non permise par les *copyrights* de certains sites centralisés ;
3. La multiplicité même de toutes ces applications ou plateformes en ligne (*e.g.* Flickr, Picasa, 500px et autres pour la photographie grand public), obligeant chaque utilisateur à en choisir une. Les données des utilisateurs se retrouvent ainsi distribuées sur plusieurs sites.

Cela conduit au second problème général :

Problème 2 : *comment exploiter les techniques distribuées afin de permettre aux chercheurs et bénévoles botanistes de collaborer, où que se trouvent leurs données ?*

1.3 Aperçu de l'état de l'art

Au milieu des années 90, les systèmes de recommandation (*i.e. RS*) ont été proposés afin de délivrer les bons contenus aux bons utilisateurs au bon moment, de manière pro-active [3]. Deux grandes catégories de systèmes de recommandation existent. La première consiste à analyser les comportements des utilisateurs et à en détecter des corrélations, des motifs, etc. afin d'effectuer des prédictions quant à celui de l'utilisateur courant : il s'agit du *filtrage collaboratif* [61, 69,

135]. La seconde méthode exploite les contenus des objets afin d'en extraire des corrélations, des motifs, etc. ; ces derniers sont ensuite utilisés pour recommander à l'utilisateur courant des objets qui sont proches de ceux qu'il a déjà *aimé* (*e.g.* acheté, fourni une bonne note). Il s'agit ici du *filtrage basé sur les contenus* [22, 34, 141].

Ces méthodes sont parfois combinées afin de n'en garder que les meilleurs aspects au sein de techniques dites de *filtrage hybride* [5, 56].

Ces systèmes s'appuient traditionnellement sur une matrice, dite « *Users* × *Items* », où chaque case représente la note qu'un utilisateur u a associé à un objet i – il est possible de représenter l'idée de partage d'un objet par un utilisateur en remplissant la matrice avec des 1 et des 0 s'il ne partage pas l'objet correspondant. Les prédictions issues de ces modèles de recommandation sont malheureusement statiques ; l'utilisateur ne peut généralement pas interagir avec le système pour mettre à jour ces recommandations, en soumettant par exemple des requêtes à mots clés, ou en commençant à remplir un panier d'achat comme sur *Amazon* où les recommandations pourraient se faire dynamiquement en fonction de ce dernier (*e.g.* vous avez acheté un iPhone, peut-être souhaiteriez-vous acheter une housse).

L'application de techniques issues de la recherche d'information, comme les *top-k*, permet à certains sites collaboratifs cette fonctionnalité [9, 7]. Étant donné un index, une requête q généralement à mots clés, et une fonction de score s, le *top-k* est un algorithme qui va calculer les k résultats maximisant ce score $s(q, index)$. Ces solutions, combinées avec des techniques de recommandations [9, 12], permettent de retourner à l'utilisateur des résultats à la fois pertinents par rapport à son profil et à sa requête. En botanique, cela consisterait à retourner à un utilisateur u des observations issues de plantes proches de son profil et satisfaisant son besoin immédiat, exprimé par une requête à mots clés.

Cependant, les algorithmes de recommandation ou de recherche d'information ont généralement tendance à retourner des objets déjà populaires – et donc ne nécessitant pas, par définition, de recommandation –, très similaires entre eux et n'apportant pas de nouveauté à l'utilisateur [11, 13, 15]. En effet, puisque les méthodes de recommandation s'appuient sur les notes soumises par les utilisateurs à des objets ou directement sur les objets partagés par ces derniers, les éléments populaires représentent la majorité des notes et donc la majorité des recommandations. De plus, lorsque le contenu des objets est exploité dans le processus de recommandation, les objets retournés aux utilisateurs se retrouvent très similaires entre eux.

La diversification [13, 15, 17] consiste à construire des listes de résultats (ou de recommandations) en introduisant une distance entre les objets qui s'y trouvent de sorte que chaque objet recommandé soit différent des autres (*i.e.* divers). Cette distance est généralement calculée comme l'inverse de la similarité entre deux objets, par exemple en exploitant le contenu de ces derniers. Deux points peuvent

limiter l'efficacité de cette diversification. Tout d'abord, les contenus décrits de manière incomplète, voire erronée réduiraient, par exemple, considérablement l'effet de la diversification. En second lieu, quand bien même la description serait correcte, si celle-ci emploie un vocabulaire ambigu, certaines méthodes de similarité pourraient évaluer deux objets comme similaires ou pas du tout, alors qu'en réalité, c'est l'inverse. La sérandipité [11] aborde un problème similaire. La distance est ici introduite entre les objets recommandés et ceux rattachés au profil de l'utilisateur.

Concernant la problématique de gestion des données distribuées, les systèmes *P2P* (*i.e.* pair-à-pair) offrent de nombreux avantages comme le passage à l'échelle, la dynamicité, la gestion des pannes et le contrôle décentralisé. Dans un réseau de ce type, chaque nœud peut représenter un utilisateur désireux de collaborer avec d'autres en partageant ses contenus. Cette opération se fait ici directement entre les pairs, sans passer par des entités intermédiaires. Les protocoles distribués permettent ainsi les opérations suivantes : localisation d'autres nœuds ou de contenu, réplication et récupération de ces contenus, etc.

Les réseaux pair-à-pair peuvent être classés au sein de plusieurs catégories [123, 124, 138] en fonction de leur type de topologie : réseaux structurés ou non, avec super-pairs ou dynamiques.

Deux approches peuvent être mises en avant pour la recherche et la recommandation. Tout d'abord, plus en rapport avec la recherche, les techniques pour la gestion de données s'occupent de regrouper les données [161] ou les pairs [24, 155] proches afin d'en faciliter l'accès et la recherche (*i.e.* classification ou *clustering*) ou en créant des raccourcis entre les nœuds. En second lieu, les techniques de prédictions [115], quant à elles, consistent à construire un recouvrement (*i.e.* ensemble des liens entre les pairs d'un réseau *P2P*) pair-à-pair tel que le voisinage de chaque pair permet, en agrégeant les informations que les pairs du voisinage possèdent (*e.g.* données qu'ils ont aimées), de calculer les recommandations (*i.e.* prédictions).

La majeure partie des solutions *P2P* précédemment introduites, construit donc le voisinage de chaque pair en utilisant des méthodes de similarité, ce qui y introduit de la redondance. Ainsi, lorsqu'une requête est soumise, chacun d'entre eux a une forte probabilité de retourner le même ensemble de résultats que ses « voisins », ce qui limite la capacité du système à retrouver des résultats pertinents.

Parallèlement au pair-à-pair, les techniques multisites définissent un type d'architecture où chaque nœud du réseau, appelé site, regroupe un certain nombre d'utilisateurs. Par exemple, en botanique, le site *TelaBotanica* regroupe un grand nombre d'utilisateurs, principalement francophones, mais d'autres sites permettent de partager des observations de plantes. Ces sites sont donc moins nombreux que les nœuds d'un réseau *P2P*, mais également plus fiables (*i.e.* la probabilité qu'ils tombent en panne ou quittent simplement le réseau est beaucoup plus faible que

pour les nœuds d'un réseau *P2P*). L'objectif des contributions relatives à ces architectures est d'optimiser le traitement des requêtes en minimisant la charge réseau (*i.e.* le nombre de fois qu'une requête passe par le réseau), tout en maximisant le rappel. Pour cela, des conditions de propagation de la requête (*i.e.* condition vraie si la requête doit être transmise à un autre site, fausse sinon) sont définies [19, 35, 58], la réplication est utilisée pour maximiser le traitement des requêtes localement à chaque site [80]. Le choix du placement des données est également réfléchi [28], etc. Malheureusement, ces solutions ne s'appliquent pas facilement à nos scénarios où le nombre de sites est beaucoup plus élevé (*e.g.* un site par équipe de recherche) et où ces derniers forment un ensemble hétérogène (*e.g.* un utilisateur, observateur des plantes rejoignant avec son serveur personnel une plateforme distribuée de partage). Par ailleurs, les solutions présentées dans l'état de l'art ne prennent généralement pas en compte le profil des utilisateurs dans le traitement des requêtes.

1.4 Contributions

Nous présentons maintenant les différentes contributions issues de cette thèse. Celles-ci sont à la croisée de plusieurs domaines : la recherche d'information et la recommandation imbriquées au sein d'architectures distribuées de type *P2P* et multisites.

Diversité des profils : afin d'aborder les problématiques de diversification, la première contribution se situe dans un contexte centralisé. Nous introduisons tout d'abord l'idée de diversité des profils qui permet d'aborder les limitations précédemment décrites (*i.e.* la mauvaise description des contenus et l'ambiguïté des mots clés utilisés dans ces descriptions). Nous adoptons ici l'idée de recherche de recommandations : les utilisateurs peuvent soumettre des requêtes à mots clés pour rechercher des objets partagés par des utilisateurs pertinents. La pertinence d'un utilisateur est évaluée étant donné l'idée de filtrage hybride (*i.e.* filtrage collaboratif combiné à celui basé sur les contenus). Les résultats doivent par ailleurs être divers à la fois en termes de contenu, mais également en fonction des profils des utilisateurs partageant les objets. Cette diversité des profils permet de limiter la redondance due aux descriptions incomplètes ou ambigües. Notre approche s'appuie sur un modèle probabiliste [17, 39] et permet d'adapter le niveau de diversification en fonction de *feedbacks* utilisateurs (*e.g.* si un utilisateur n'est pas satisfait des résultats présentés, les calculs futurs adapteront la diversité en conséquence). Nous avons proposé un ensemble d'algorithmes centralisés, s'appuyant sur des listes inversées (*i.e.* ces dernières associent chaque mot clé aux documents le contenant, triés par score décroissant, par exemple en utilisant la similarité avec le mot clé donné) et sur la notion de liste de candidats afin d'être le plus efficace possible. En particulier, nous avons limité le nombre d'accès dans les

listes inversées, le nombre de candidats à considérer et amélioré le calcul du score de chaque objet. Cette contribution est accompagnée de plusieurs publications :

- M. Servajean, E. Pacitti, S. Amer-Yahia et P. Neveu, « Profile Diversity in Search and Recommendation », dans *Proceedings of the 22nd International Conference on World Wide Web Companion*, 2013, p. 973–980 ;
- M. Servajean, E. Pacitti, S. Amer-Yahia et P. Neveu, « Phenotyping Data Search and Recommendation », dans *Bases de Données Avancées*, 2013 ;
- M. Servajean, R. Akbarinia, E. Pacitti et S. Amer-Yahia, « Profile Diversity for Query Processing using User Recommendations », *Information Systems (à paraître), 20 pages*, 2014.

Diversité des profils en *P2P* : en second lieu, nous exploitons la diversité dans les réseaux *P2P*. Nous proposons une nouvelle méthode de regroupement des utilisateurs (*i.e.* calcul du voisinage de chaque pair) basée sur un score combinant pertinence et diversité, nommé *usefulness*. Cette notion de diversité est une application différente de celle présentée précédemment, en centralisé, et peut induire une certaine confusion. Il est donc nécessaire de la clarifier. Alors que la diversité est traditionnellement utilisée dans les modèles de recherche et de recommandation – et c'est le cas pour notre contribution précédente –, nous appliquons ici cette notion dans l'étape de regroupement des pairs (*i.e.* calcul du voisinage de chaque pair) du réseau *P2P*. En diversifiant le voisinage de chaque pair, la probabilité qu'ils retournent les mêmes résultats à chaque requête soumise se réduit, et la capacité du système à retrouver des données pertinentes, étant donné une requête q, augmente. Nous montrons une stratégie de réplication accroissant encore plus la probabilité de retrouver des objets pertinents. Il en résulte que le score de regroupement diversifié permet d'obtenir des gains majeurs en termes de rappel (*i.e.* fraction des résultats pertinents que le système est capable de récupérer) tout en limitant les valeurs de TTL (*i.e. Time-To-Live*). La réplication permet rapidement d'obtenir des gains encore plus élevés. Les niveaux de rappel atteignent alors des valeurs quasi centralisées. Alors que le rappel (*i.e.* capacité du système à retourner tous les bons résultats) est généralement confronté à la précision (*i.e.* capacité du système à ne retourner que de bon résultats), en recherche ou en recommandation distribuée, il est contrebalancé par les contraintes du réseau (*i.e.* taille du voisinage de chaque pair et TTL). Cette contribution est accompagnée de plusieurs publications :

- M. Servajean, E. Pacitti, M. Liroz-Gistau, S. Amer-Yahia et A. El Abbadi, « Exploiting Diversification in Gossip-Based Recommendation », dans *Proceeding of the 7th International Conference on Data Management in Cloud, Grid and P2P Systems*, Springer, 2014, p. 25–36 ;

- M. Servajean, E. Pacitti, M. Liroz-Gistau, S. Amer-Yahia et A. El Abbadi, « Exploiting Diversity in Distributed Recommendation », dans *Bases de Données Avancées*, 2014 ;
- M. Servajean, E. Pacitti, S. Amer-Yahia et A. El Abbadi, « Increasing Coverage in Distributed Search and Recommendation with Profile Diversity », à soumettre.

Plateformes multisites pour la recherche et la recommandation : en combinant les deux contributions précédentes, nous proposons une approche multisites pour sites hétérogènes, permettant la recherche et la recommandation diversifiées. Nous introduisons l'idée de nœud virtuel ; il s'agit d'un regroupement d'utilisateurs similaires dans un site. Le voisinage de chaque site est réalisé via l'utilisation des nœuds virtuels en utilisant le score de *usefulness* précédemment introduit. Lorsqu'une requête est soumise, elle est propagée au travers de ce voisinage et les résultats retournés sont choisis via notre score incluant la diversité des profils. Nous avons également proposé un algorithme entièrement distribué afin d'indexer correctement les objets dans des listes inversées, en tenant compte de statistiques globales à tous les sites (*e.g.* fréquence d'un mot dans l'ensemble des documents). Cette dernière contribution est accompagnée d'une publication :

- M. Servajean, E. Pacitti, M. Liroz-Gistau, A. Joly et J. Champ, « PlantRT : Multi-Site Diversified Search and Recommendation for Citizen Sciences », dans *Bases de Données Avancées*, 2014.

Un prototype a été réalisé pour cette contribution. Deux versions différentes, abordant nos deux cas d'application en s'adaptant par exemple au type des données, en sont issues :

- PlantRT : `http://www2.lirmm.fr/~servajean/prototypes/plant-sharing/plant-rt.html` ;
- DocRT : `http://www2.lirmm.fr/~servajean/prototypes/documents-sharing/doc-rt.html`.

1.5 Contexte de la thèse

Cette thèse est financée partiellement par le *labex Numev*[1], et a été réalisée au sein du projet *Mastodons*[2].

Il en résulte qu'au-delà des publications, ce travail s'est accompagné de multiples interventions lors de manifestations scientifiques, comprenant notamment deux posters :

1. `http://www.lirmm.fr/numev/`
2. `http://www.cnrs.fr/mi/spip.php?article53`

- *A Personalized and Diversified Model for Decentralized Scientific Search and Recommendation*[3] ;
- *Multi-Site Diversified Search and Recommendation*[4].

1.6 Organisation de la thèse

Le Chapitre 2 présente l'état de l'art des techniques importantes pour la compréhension de notre travail. Nous y abordons, en particulier, les techniques de recommandation ainsi que l'idée de recherche de recommandations via l'application de techniques comme les *top-k*. En second lieu, sont introduites les solutions de diversification et de sérandipité. Enfin, les solutions de recherche et de recommandation *P2P* et multisites sont étudiées.

Notre première contribution, traitant de la diversité des profils, est présentée au Chapitre 3. En particulier, nous définissons notre nouvelle fonction de score en tenant compte de la diversité des profils, et nous introduisons les algorithmes nécessaires à son calcul. Nous validons notre fonction de score par une évaluation expérimentale importante sur deux jeux de données issus du Web (*i.e. Delicious* et *Flickr*), dont les résultats sont validés sur des données de phénotypage et de botanique. Les algorithmes proposés offrent cependant un temps de réponse trop élevé pour offrir une satisfaction maximale à l'utilisateur.

Toujours dans le cadre de cette première contribution, le Chapitre 4 se focalise sur les aspects d'optimisation afin de minimiser le temps de réponse lors du calcul d'un ensemble de résultats pertinents et divers. En utilisant les deux mêmes jeux de données que précédemment, accompagnés d'un second jeu de données également issu de *Delicious*, nous montrons les gains majeurs en temps de réponse engendré par nos optimisations.

Le Chapitre 5 présente une nouvelle utilisation de la diversité, différente de son application traditionnelle. Ce chapitre se concentre sur la recherche et la recommandation déployées sur une architecture *P2P*. Nous montrons, plus précisément, comment l'introduction de la diversité dans le processus de construction du voisinage de chaque pair permet d'augmenter le rappel tout en limitant le *TTL* (*i.e.* le nombre de sauts que fait la requête sur le réseau, et donc le nombre de pairs impliqués dans le calcul des résultats). Nous introduisons également une méthode de réplication qui tire parti du voisinage diversifié de chaque pair, et nous validons expérimentalement ces deux points via l'utilisation de trois jeux de données : *MovieLens*, *Flickr* et *LastFM*.

Le Chapitre 6 présente les deux versions du prototype issu de cette thèse. En particulier, nous avons combiné les contributions propres aux trois précédents chapitres pour proposer une adaptation multisite. Nous présentons la manière

3. `http://www2.lirmm.fr/~servajean/new/download/Servajean-NUMEV13.pdf`
4. `http://www2.lirmm.fr/~servajean/new/download/Servajean-NUMEV14.pdf`

dont les deux versions du prototype répondent à nos cas d'utilisation. Nous introduisons son architecture et nous validons expérimentalement sa capacité à être déployé réellement en utilisant des données de botanique issues du projet *Pl@ntNet*, et à petite échelle via le partage de données de phénotypage.

Enfin, en conclusion, dans le Chapitre 7, nous synthétisons les différentes contributions et proposons des pistes de recherche.

CHAPITRE 2

État de l'art

Résumé. Ce chapitre présente les contributions fondamentales nécessaires à la compréhension de cette thèse. Il est composé de quatre sections. La première introduit les techniques de recommandation et de recherche de l'information, comprenant principalement les solutions de prédiction (i.e. les modèles de recommandation), de recommandation (i.e. top-N) et les top-k. Seront également abordées les principales approches de diversification et de sérendipité qui permettent d'améliorer la qualité des recommandations en réduisant leur redondance. La seconde section aborde les topologies et les techniques P2P pour la gestion de données distribuées. La troisième section introduit à la recherche d'information et aux techniques de prédictions distribuées ; une présentation de la recherche multisite est également réalisée. Enfin, une synthèse et une discussion concluent ce chapitre.

2.1 Systèmes de recommandation et de recherche de l'information

Cette section introduit aux problèmes de recommandation et de recherche de l'information. Elle est structurée de la manière suivante. Une présentation générale accompagnée de définitions sont proposées en Section 2.1.1. Ensuite, en Section 2.1.2 sont détaillés les modèles de recommandation, ou de prédiction, suivis par les algorithmes de recherche et de recommandation (*i.e. top-k* et *top-N*) en Section 2.1.3. Les méthodes de diversification et de sérendipité sont également traitées en Section 2.1.4.

2.1.1 Présentation et définitions

Les problèmes de recherche d'information se sont posés à chaque fois que l'être humain fut confronté à de grands volumes de données. Ainsi, au cours de l'An-

Table 2.1: Bénéfice de la diversité sur les données de botanique avec la requête « asteraceae ».

Methodes	Without Diversity	With Diversity
Profiles	Leucanthemum adustum (*i.e.* marguerites), ...	Cirsium vulgare, ...
Results		

tiquité, dans la bibliothèque d'Alexandrie, les 500 000 à 800 000 volumes étaient étiquetés, puis rangés par thèmes, et finalement ordonnés par auteurs [157].

En informatique, cette bataille pour retrouver les informations aboutit à une structuration en deux principaux domaines. Il s'agit, tout d'abord, de la recherche à proprement parler qui, à partir d'une requête q – généralement à mots clés, mais pas uniquement – et d'une fonction de score s, s'emploiera à retrouver chaque document d satisfaisant q (*i.e.* $s(q, d) > 0$). Ces documents sont généralement présentés à l'utilisateur triés par score décroissant [108, 141, 170]. C'est aussi ce que l'on appelle le « problème de la bibliothèque ».

En second lieu, la recommandation (cf. Définition 1) permet de retourner automatiquement (*i.e.* sans soumission de requête) des contenus pertinents à l'utilisateur.

Définition 1 (Recommandation)**.**

En exploitant l'ensemble des objets $it \in I$ d'une plateforme et les profils des utilisateurs, il s'agit de leur présenter automatiquement un sous-ensemble de I susceptible de les intéresser par l'application de techniques spécifiques, dites de recommandation. On appelle liste de recommandations ce sous-ensemble qui est retourné à l'utilisateur.

Deux étapes sont nécessaires à la réalisation d'une recommandation. Il s'agit,

tout d'abord, de la prédiction – ou filtrage –, qui correspond à la capacité du système à prédire la satisfaction d'un utilisateur par rapport à un produit. Deux approches principales sont ici confrontées :

1. **Le filtrage collaboratif :** il s'appuie sur l'idée qu'une recommandation sera de meilleure qualité si elle provient d'une personne de confiance. Ces méthodes exploitent généralement une matrice « Users × Items », notée S, qui associe à chaque utilisateur les notes qu'il a déjà données à certains objets (*e.g.* l'utilisateur u a mis la note de 5/5 à l'article *Phenotyping...*, ou 0/1, si l'utilisateur a consulté ou partagé un contenu ou non). Le but devient alors, en exploitant la matrice S, de prédire le comportement de u (*i.e.* les notes qu'il attribuerait) pour des objets qu'il n'a pas encore évalués, en exploitant les notes soumises par des utilisateurs qui ont eu le même comportement par le passé.
2. **Le filtrage basé sur les contenus :** une seconde technique de recommandation s'appuie sur le contenu des objets. Il s'agit alors de recommander à un utilisateur u les objets les plus similaires à ceux qu'il a déjà noté (*e.g.* u a partagé des observations de plantes de la famille *asteraceae* : il lui en sera donc recommandé d'autres de cette même famille).

La deuxième étape, qui suit celle de prédiction, est la recommandation en soi qui est l'opération permettant de retourner à un utilisateur donné les meilleures prédictions qui le concernent. Nous exploitons dans ce travail la notion de « recherche et recommandation » ou de « recherche de recommandations ». Cette dernière est définie ci-après.

Définition 2 (Recherche de recommandations)**.**

Étant donné une requête à mots clés q, un utilisateur u et l'ensemble des objets I, il s'agit de retourner à u des résultats $R^q \subset I$ pertinents étant donné q, mais aussi étant donné le profil de u, par l'application de techniques issues des systèmes de recommandation.

D'autres solutions découlent des deux points précédents, comme le filtrage hybride, combinaison des approches décrites. Il est ainsi possible de définir les profils des utilisateurs à partir du contenu des objets (*e.g.* contenu textuel) qu'ils partagent, et de réaliser le processus de recommandation de manière collaborative, par exemple : en botanique, cela reviendrait à recommander des observations provenant d'utilisateurs ayant observé des plantes au sein d'une même région géographique ou des mêmes familles que celles de l'utilisateur courant.

Les systèmes de recommandation souffrent malheureusement de deux principales limitations [2, 16, 176] :

1. **Diversification :** les recommandations faites à chaque utilisateur u ont tendance à être trop redondantes. Elles ont souvent un contenu très proches les une des autres. Ainsi, un utilisateur dont le profil indique, par exemple, qu'il partage des données de phénotypage traitant de la plante *arabidopsis* se verra recommander des objets similaires, soit un ensemble d'objets traitant tous de cette même plante et n'aura aucune ouverture. La Figure 2.1 illustre bien ce problème ; la première colonne, non diversifiée, regroupe un ensemble d'observations très similaires, correspondants au profil de l'utilisateur (*i.e.* marguerites). Les recommandations font également très souvent référence à des objets déjà populaires, car ils sont tout simplement les plus présents dans la matrice « *Users* × *Items* ». Yu *et al.* [176] ont montré que sur *Delicious*, au moment de la primaire démocrate de 2008, le thème *élection* ne permettait de retrouver que des liens relatifs à *Barack Obama*, qui en plus d'être très proches en termes de contenus étaient également les plus populaires sur la plateforme. En botanique, le problème est le même. La Figure 1.2, en page 3, présente la distribution des observations qu'il peut y avoir dans l'application *Pl@ntNet*. On observe que très peu de plantes sont très populaires (*e.g.* marguerites, roses) ; elles représenteront donc la majorité des recommandations. Les techniques de *diversification* permettent cependant de limiter ce problème de redondance en introduisant une notion de distance entre les objets recommandés : étant donné une liste de résultats R à recommander à l'utilisateur u, un objet (*i.e. item*) $it \in R$ doit être pertinent pour u mais également distant des autres objets présents dans R.
2. **Sérendipité :** les recommandations faites aux utilisateurs sont généralement très proches de ce que l'utilisateur a déjà évalué ou vu, ce qui risque de rendre la plateforme ennuyante [2]. En effet, puisque le profil de l'utilisateur courant u est utilisé pour produire les recommandations, que ce soit les méthodes collaboratives ou basées sur les contenus, elles recommanderont des contenus provenant soit d'utilisateurs très similaires à u (*i.e.* qui sont donc associés à des objets très proches de ceux partagés par u), soit directement très similaires aux contenus partagés ou évalués par u. La Figure 2.1, encore une fois, illustre bien ce problème ; la première colonne ne contient que des observations très proches du profil de l'utilisateur (*i.e.* marguerites). Les techniques évoquées ici sont dites de *sérendipité* : l'objectif est d'introduire de la distance entre les objets déjà évalués par l'utilisateur et ceux retournés par le moteur de recommandation.

2.1.2 Modèles de recommandation ou prédiction

Cette section se propose d'introduire aux méthodes de prédictions ; ces dernières, étant donné l'ensemble des utilisateurs, des objets et diverses informations (*e.g.* notes des utilisateurs sur les objets, liens d'amitiés), vont calculer des prédic-

tions (*i.e.* quelle note attribuerait l'utilisateur u à l'objet it, qu'il n'a pas encore évalué ou vu).

Quatre approches sont présentées. En premier lieu, le filtrage collaboratif exploite les liens (*e.g.* corrélation, similarité) entre utilisateurs afin d'effectuer les prédictions. Ensuite, le filtrage basé sur les contenus exploite les liens entre objets. Le filtrage hybride combine les deux précédentes stratégies dans le processus de prédiction. Un exemple caractéristique est de construire les profils des utilisateurs à partir des contenus des objets qu'ils partagent ou évaluent, et de réaliser le processus de recommandation de manière collaborative. Enfin, le filtrage social, qui est une sous-catégorie du filtrage collaboratif, exploite les données sociales dans le processus de prédiction. Ainsi, si deux utilisateurs sont amis sur un réseau social spécialisé dans la botanique, les observations de l'un des deux seront par exemple recommandées au second.

Filtrage Collaboratif

Deux utilisateurs u et v qui ont par le passé eu un comportement similaire, voire identique, ont une forte probabilité d'avoir cette même tendance dans le futur. Ainsi, exploiter l'ensemble des utilisateurs – ou seulement les plus proches de u, notés *neighbor(u)* – permet de prédire les notes que u pourrait attribuer à des objets qu'il n'a pas encore évalués [27, 61, 101, 136, 153]. Nous disposons donc de deux ensembles : le premier, composé de n utilisateurs (*i.e. users*), est noté U et le second, composé de m objets (*i.e. items*), est noté I : ces deux ensembles forment la matrice « *Users* $\times$ *Items* », notée S. Chaque couple $\{utilisateur, objet\}$ est associé à la note que l'utilisateur $S_{i,\star}$ (*i.e.* ligne de la matrice S correspondant à l'utilisateur) a mis à l'objet $S_{\star,j}$ (*i.e.* colonne de la matrice S correspondant à l'objet) : $S_{i,j}$ qui prend la valeur 0 ou « $-$ » si l'utilisateur n'a pas encore évalué l'objet et la note qu'il a attribué dans le cas contraire (*e.g.* une note entre 1 et 5, ou juste 1 si on utilise la notion de partage). L'ensemble des couples forment la matrice *Users*$\times$*Items* notée S.

La Figure 2.1 illustre les étapes d'un système de filtrage collaboratif.

Les techniques de filtrage collaboratif peuvent être structurées en deux catégories : (1) celles à base de mémoire qui utilisent les notes précédentes pour calculer des corrélations (*e.g.* utilisateurs mettant les mêmes notes aux mêmes objets ou objets recevant des notes identiques des mêmes utilisateurs) afin d'en déduire les prédictions ; (2) et celles à base de modèles qui exploitent des techniques comme le *machine learning* afin de déterminer des motifs et prédire les notes manquantes de la matrice S. Bien que les chapitres suivants se focalisent sur les méthodes à base de mémoire, celles à base de modèles seront malgré tout introduites brièvement par la suite.

Tout d'abord, les techniques à base de mémoire qui établissent les corrélations nécessaires pour le calcul des prédictions peuvent être établies de deux manières : (1) les utilisateurs ayant un comportement similaire, ou (2) les objets étant notés

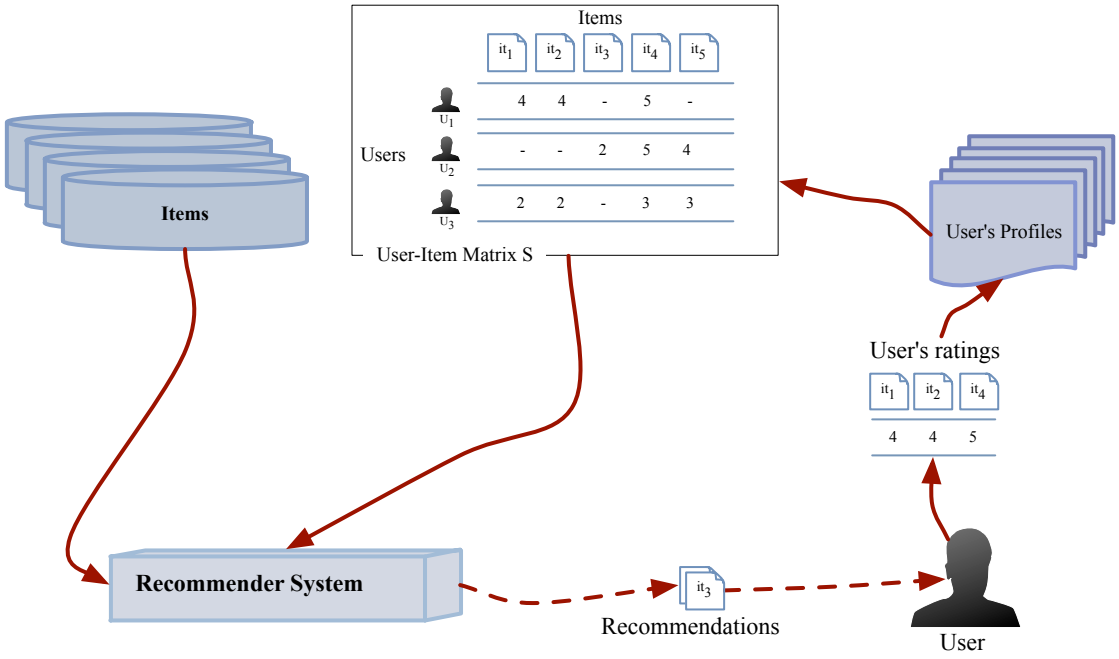


Figure 2.1: Étapes générales d'un processus de filtrage collaboratif.

de manière corrélée :

- **Corrélation Users×Users :** une manière simple d'effectuer des prédictions consiste à (1) définir une fonction de similarité entre les utilisateurs, (2) calculer pour un utilisateur u donné, ceux qui en sont le plus proches, notés $neighbors(u)$, (3) et finalement prédire à partir de ces derniers des notes sur des objets que l'utilisateur u n'a pas encore évalués.

  Rappelons que chaque utilisateur $S_{i,\star}$ est représenté par un vecteur creux de notes – l'utilisateur n'a généralement évalué qu'une petite partie des objets disponibles. Plusieurs méthodes permettent d'évaluer la similarité entre deux vecteurs. Tout d'abord, le cosinus calcule l'angle :

$$cos(\theta) = \frac{S_{i,\star} \times S_{j,\star}}{||S_{i,\star}|| \times ||S_{j,\star}||} = \frac{\sum_{k=1}^{n} S_{i,k} \times S_{j,k}}{\sqrt{\sum_{k=1}^{n} S_{i,k}^2} \times \sqrt{\sum_{k=1}^{n} S_{j,k}^2}} \tag{2.1}$$

  Cette méthode établit la similarité du comportement de deux utilisateurs (*e.g.* ils mettent toujours les mêmes notes à certains films).

  Dans une optique similaire, le coefficient de corrélation linéaire de Bravais-Pearson [61, 153] permet de calculer une corrélation entre deux utilisateurs (*i.e.* deux comportements opposés, identiques ou orthogonaux). Le coefficient tendra vers 1 s'ils mettent les mêmes notes, et vers -1 si les deux utilisateurs ont un comportement opposé. Le 0 indique l'absence totale de

corrélation. Ce coefficient se définit de la manière suivante :

$$corr(S_{k,\star}, S_{m,\star}) = \frac{\mathbb{E}[S_{k,\star}S_{m,\star}] - \mathbb{E}[S_{k,\star}] \times \mathbb{E}[S_{m,\star}]}{\sigma_{S_{k,\star}} \times \sigma_{S_{m,\star}}} \tag{2.2}$$

Où la fonction $\mathbb{E}$ indique l'espérance d'une valeur et σ son écart type. La formule peut s'évaluer de manière géométrique comme un cosinus tel que présenté ci-dessous :

$$corr(S_{k,\star}, S_{m,\star}) = \frac{\sum_{i=1}^{n}(S_{k,i} - \overline{S_{k,\star}}) \times (S_{m,i} - \overline{S_{m,\star}})}{\sqrt{\sum_{i=1}^{n}(S_{k,i} - \overline{S_{k,\star}})^2} \times \sqrt{\sum_{i=1}^{n}(S_{m,i} - \overline{S_{m,\star}})^2}} \tag{2.3}$$

Ainsi, les candidats à faire partie du voisinage de u sont plus nombreux que pour le cosinus puisque les comportements opposés sont aussi pris en compte.

Finalement, étant donné un utilisateur u et son voisinage *neighbors(u)* (*i.e.* les k utilisateurs avec la plus forte corrélation), la dernière étape consiste à calculer un score pour chaque objet $it_i \in I$, non encore évalué par u ; ce score est noté $S^*_{u,i}$. Ce but peut être atteint en appliquant simplement une moyenne pondérée :

$$S^*_{u,i} = \frac{\sum_{v \in neighbors(u)} S_{v,i} \times sim(u,v)}{\sum_{v \in neighbors(u)} sim(u,v)} \tag{2.4}$$

Ici, $S_{v,i}$ représente la note qu'un utilisateur v a donné à un objet k et *sim* est l'une des méthodes de corrélation précédemment décrite. Il s'agit de prendre en compte l'avis de chaque utilisateur présent dans le voisinage de u, pondéré par sa corrélation par rapport à u.

- **Corrélation *Item-based* :** les prédictions peuvent aussi être effectuées en prenant le problème du point de vue des objets [95, 144] : (1) on évalue une similarité entre objets, (2) on calcule le voisinage d'un objet et finalement (3) on prédit sa note pour un utilisateur. Les mêmes méthodes que celles décrites précédemment sont utilisables puisque les objets sont également représentables par des vecteurs creux.

D'autres approches ont également été proposées comme une vision hybride entre les deux types de corrélation [172].

Les techniques à base de modèles, quant à elles, permettent de détecter des motifs complexes sur les données. Ces motifs sont ensuite exploités afin de prédire les scores manquants de la matrice S, comme précédemment. Ce genre de techniques est souvent proposé afin d'outrepasser les limitations des techniques à base de mémoire (*e.g.* utilisateur avec un voisinage réduit ou inexistant, objets jamais évalués). Plusieurs approches ont été proposées :

- **Réseau Bayésien [68, 117, 129] :** les réseaux Bayésiens définissent la probabilité que l'objet *it* atteigne un score particulier, étant donné un utilisateur u. Chaque score – on peut supposer que le score d'un objet puisse prendre une valeur entre 1 et 5 – définit une classe et il s'agit finalement d'un problème de classification où chaque objet se retrouve dans la classe qui lui convient le mieux.
- **Factorisation de matrices :** Une des méthodes de filtrage collaboratif ayant parmi les meilleurs résultats – technique utilisée par les vainqueurs du prix *Netflix*[1] – est la factorisation de matrice [135]. La matrice « Users × Items », notée S, est toujours au cœur de cette solution. Il s'agit de la décomposer en une factorisation de deux sous-matrices :

$$S^\star = U^T \times V \tag{2.5}$$

Où U est une matrice décrivant les utilisateurs au travers de caractéristiques latentes et la matrice V décrit les objets de la même manière. Le but de ce problème est de calculer les matrices U et V de telle manière que leur factorisation $S^\star$ soit le plus similaire possible de S.

D'autres modèles ont été développés afin de résoudre les limitations du filtrage collaboratif comme le *clustering* où les objets similaires sont regroupés [38, 65, 121, 145] ou encore des techniques de régression linéaire [69].

Limitations relatives au filtrage collaboratif : Comme cela a été dit, deux principales limitations existent. Un utilisateur n'ayant jamais soumis de vote ne pourra pas se faire recommander des objets correspondants à son profil. De la même manière, un objet n'ayant jamais été évalué ne pourra pas être correctement recommandé. Il s'agit du problème de démarrage à froid [6, 92, 30].

À cela s'ajoute le fait que même les utilisateurs les plus actifs des plateformes de *e-commerce* n'accèdent qu'à une part minime, souvent inférieure à 1% des objets. Sur les 20 millions de livres d'Amazon, un utilisateur accédant à 1% d'entre eux, serait confronté à 20 000 livres. Il s'agit des problèmes de matrices creuses [70, 125] : la matrice existe mais est presque complètement vide. Ainsi, même un utilisateur ayant évalué des objets peut se retrouver, malgré tout, sans voisinage.

Les approches à base de modèle permettent d'atténuer les précédentes limitations, mais doivent être recalculées très régulièrement afin de prendre en compte les nouvelles notes ajoutées au système [31].

Filtrage Basé sur les Contenus

Ces techniques prennent une approche orthogonale à celles du filtrage collaboratif. Il s'agit, cette fois, de définir des méthodes de similarité ou des modèles

1. `http://www.netflixprize.com`

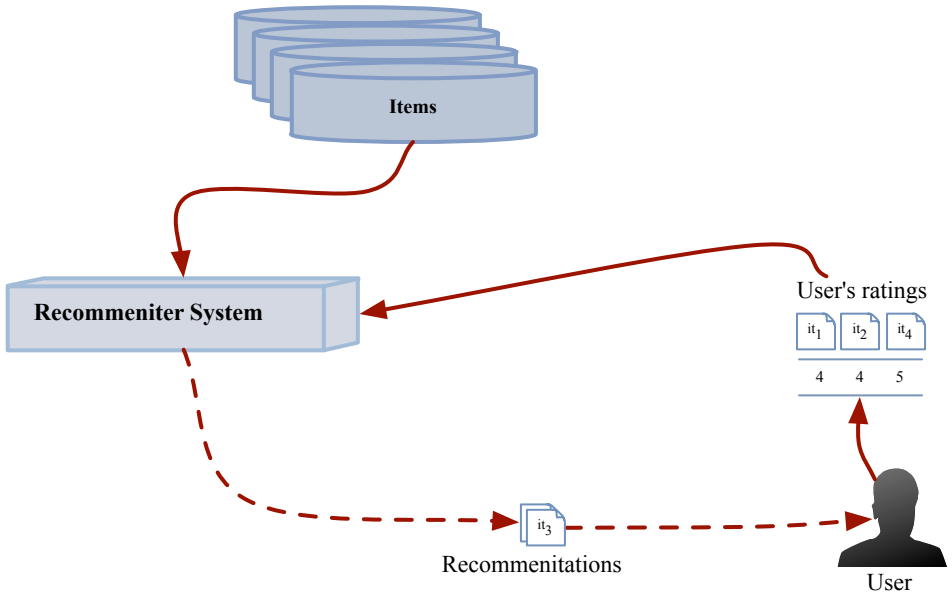


Figure 2.2: Étapes générales d'un processus de filtrage basé sur les contenus.

exploitant le contenu des objets pour effectuer des recommandations. Les étapes générales propres au filtrage à base de contenus sont présentées dans la figure 2.2.

Tout d'abord, l'objectif des méthodes de filtrage basé sur les contenus à base de mémoire est d'évaluer la similarité entre chaque vecteur $j \in I \backslash I_u$ et $i \in I_u$ où $I_u \subset I$ représente l'ensemble des objets déjà évalués par u. Cette similarité permet ensuite de calculer des prédictions en exploitant les notes déjà soumises par u ou les objets qu'il partage, par exemple en utilisant une moyenne pondérée (cf. Équation 2.4). Ainsi, un botaniste partageant des observations de marguerites, s'en verrait recommander d'autres qu'il pourra identifier ou valider. Plusieurs techniques permettent d'atteindre ce but :

- **Espace vectoriel [141] :** le modèle vectoriel reste l'un des plus populaires et des plus utilisés aujourd'hui en recherche d'information. Ici, les objets à recommander sont représentés par des vecteurs de hautes dimensions. Chaque dimension représente ainsi un mot clé existant ou un descripteur. Dans un univers booléen, cette valeur serait de 1 ou 0 en fonction de la présence ou de l'absence du mot dans la description de l'objet – par la suite le mot *document* fera référence à un objet et à sa description textuelle. De manière plus générale, il s'agit de l'importance du mot pour un document donné. Ainsi, chacun d'entre eux peut se définir de la manière suivante :

$$\begin{aligned} d_i &= \{\langle k_1 : w_{i,1}\rangle, \langle k_2 : w_{i,2}\rangle, ..., \langle k_n : w_{i,z}\rangle\} \\ d_j &= \{\langle k_1 : w_{j,1}\rangle, \langle k_2 : w_{j,2}\rangle, ..., \langle k_n : w_{j,z}\rangle\} \end{aligned} \tag{2.6}$$

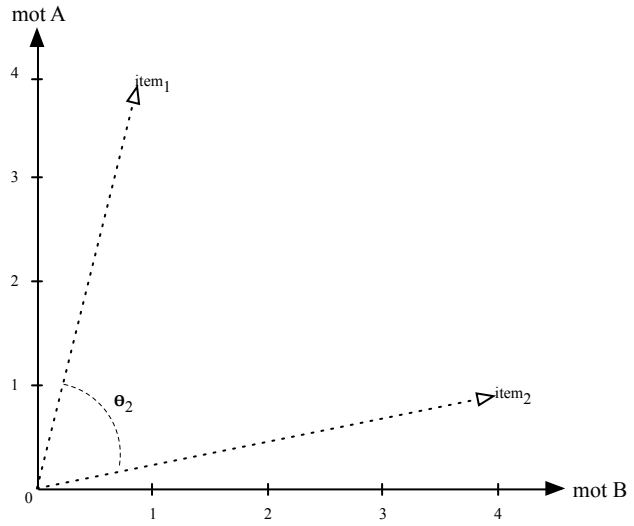


Figure 2.3: Espace vectoriel à deux dimensions représentant deux documents.

Où k_1 représente le premier mot clé (ou descripteur) du corpus et $w_{i,1}$ est son poids pour le document i. Ce poids prend sa valeur dans le champ suivant : $w_{i,l} \in [0, \infty[$. Remarquons tout de même qu'il s'agit encore une fois de vecteurs creux qui doivent donc être implémentés en conséquence : seuls les mots présents dans le document sont conservés, un mot absent aura donc la valeur 0. Calculer le poids d'un mot par sa fréquence dans le document présente cependant une grande limitation : certains termes comme « le » sont tellement courants qu'ils risquent de « cacher » les autres mots. En botanique ou en biologie, le mot « plante » est tellement commun que sa présence ou son absence d'une requête ne devrait pas influencer le résultat. G. Salton [141] a ainsi proposé la technique $TF \times IDF$, signifiant « la fréquence du terme (dans le document) multipliée par l'inverse de sa fréquence globale », permettant d'atténuer l'importance des mots très fréquents dans la majeure partie du corpus et, à l'inverse, d'augmenter la fréquence de ceux qui sont rares et présents dans peu de documents. De nombreuses formules satisfont cette idée. Une des plus récurrentes est la suivante :

$$tf(k,d) \times idf(k,D) = (\log(f(k,d)) + 1) \times \log\left(\frac{|D|}{\{d \in D : k \in d\}}\right) \quad (2.7)$$

Où D représente l'ensemble des documents. La fonction logarithmique sert ici à atténuer les écarts de scores entre les mots très fréquents et ceux qui le sont moins.

Finalement, plusieurs opérations permettent de calculer la similarité entre deux vecteurs. Une des plus connues reste le *cosinus* (cf. Equation 2.1).

La Figure 2.3 présente un exemple dans un environnement vectoriel à deux dimensions, ce qui signifie qu'il n'existe que deux mots (ou descripteurs) différents [34]. Ainsi, plus l'angle entre deux vecteurs est réduit, plus les vecteurs sont similaires et plus la valeur du cosinus tendra vers 1.
Le vecteur peut être simplifié en ne conservant que les 100 ou 120 mots les plus significatifs (*i.e.* avec le plus grand poids) [22, 128].
L'équation 2.4 peut être appliquée pour calculer les prédictions.

D'autres techniques comme les ontologies [105, 163] sont également utilisées.

De la même manière que pour le filtrage collaboratif, des méthodes à base de modèles ont été proposé dans le cadre de celui basé sur les contenus, comme des arbres de décision [126, 127, 132], des réseaux bayésiens [128], des réseaux de neurones [85, 173], ou encore des modèles basés sur les ensembles flous [122].

Limitations : deux principales catégories de limitations existent pour les techniques à base de contenus : les recommandations sont à la fois très similaires à ce que l'utilisateur a déjà évalué ou vu (puisque l'objectif est de recommander les objets les plus proches de ceux que l'utilisateur a déjà évalué), mais aussi entre elles (en étant très proche du profil de l'utilisateur, les objets recommandés se retrouvent par définition très similaires), et l'utilisateur ne se verra recommander que des objets très ciblés, limitant par là, la découverte de nouveaux contenus ; ensuite, un utilisateur n'ayant jamais rien évalué ne pourra pas se faire recommander efficacement un contenu, puisque son profil ne peut pas être exploité.

Filtrage Hybride

Afin de réduire au maximum les limitations inhérentes à chaque méthode de prédiction (*i.e.* filtrage collaboratif et basé sur les contenus), celles-ci sont parfois combinées dans le but d'en exploiter les meilleurs aspects.

Le profil de chaque utilisateur peut ainsi être calculé en exploitant les contenus des objets (*i.e.* filtrage basé sur les contenus), mais au lieu de recommander les objets les plus proches de son profil, il s'agit de calculer un voisinage d'utilisateurs (*i.e.* filtrage collaboratif) et de les exploiter dans le processus de prédiction [22, 88].

Les deux méthodes de recommandation (*i.e.* filtrage collaboratif et basé sur les contenus) peuvent être également exécutées simultanément, et les deux listes de recommandations sont retournées à l'utilisateur automatiquement [5], ou à sa demande [56], voire combinées linéairement [116] :

$$pred(u, it) = \lambda \times pred_{cont}(u, it) + (1 - \lambda) \times pred_{coll}(u, it) \tag{2.8}$$

où $pred(u, it)$ est la prédiction finale du score de l'objet $it \in I$ pour l'utilisateur u, $pred_{cont}(u, it)$ la prédiction étant donné une méthode basée sur les contenus,

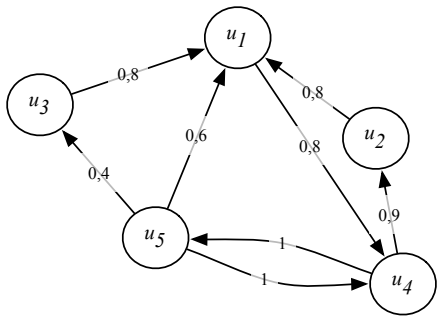


Figure 2.4: Exemple du graphe d'un réseau social.

$pred_{coll}(u, it)$ celle étant donné une méthode collaborative et $\lambda \in [0, 1]$ une variable définie par le système.

Remarquons malgré tout que ces solutions ne limitent pas les problèmes de redondance, et se focalisent principalement sur les objets déjà populaires et redondants. En effet, les deux méthodes souffrant déjà de ce problème, leur combinaison ne résout rien.

Filtrage Social

Afin d'améliorer la qualité des prédictions, il est possible d'exploiter les liens d'amitiés explicites ou implicites fournis par les réseaux sociaux [14, 90]. En effet, l'émergence du Web 2.0 a permis l'apparition de tels réseaux comme Facebook ou Twitter. Le filtrage social, en exploitant les utilisateurs, est donc une sous-catégorie du filtrage collaboratif.

Il est possible de distinguer deux approches. Dans la première, la plateforme dispose déjà d'une matrice « *Users* × *Items* », notée S, à laquelle s'ajoute le réseau social. Ce dernier se décrit comme un graphe $\mathcal{G} = (U, E)$, où U représente les utilisateurs et E les liens d'amitiés entre eux (cf. Figure 2.4). Un lien peut être pondéré par un indice de confiance. Les utilisateurs partagent ou évaluent donc des objets, ce qui permet de construire S et définissent également une liste d'amis, formant le graphe $\mathcal{G}$. Ce dernier, en calculant la confiance qu'un utilisateur peut avoir en un autre [25, 103, 102, 120], permet d'améliorer les prédictions de la matrice S. Un ensemble de techniques possible est présenté ci-dessous :

- **Factorisation de matrice [102, 103, 120, 135] :** le réseau social est représenté comme une matrice « Users × Users », notée E (*i.e.* la matrice représente les liens du graphe). Il suffit d'établir une factorisation supplémentaire $E^\star = U_1 \times U_2^T$ où U_1 est la même matrice de caractéristiques latentes des utilisateurs utilisée dans la méthode de factorisation des matrices présentée précédemment et U_2 d'autres caractéristiques latentes des utilisateurs. $E^\star$ est le résultat de la factorisation qui doit être le plus proche possible de E.

- **Basé sur la confiance [25, 93] :** le réseau social est construit comme un indicateur de la confiance qui peut exister entre deux utilisateurs. La confiance entre deux utilisateurs u et v est établie à partir du succès qu'a v à produire de bonnes recommandations pour u. Ce réseau est ensuite utilisé afin de calculer le voisinage de u pour prédire les notes.

- **Systèmes collaboratifs d'étiquetage [154, 164] :** les utilisateurs peuvent associer des étiquettes à des objets partagés sur la plateforme. Ces étiquettes et objets forment le profil de chaque utilisateur et les objets recevant des étiquettes similaires sont regroupés en *cluster*. Finalement, les proximités entre le profil d'un utilisateur et un filtre (*i.e.* une étiquette particulière que l'utilisateur a indiqué) avec les *clusters* sont utilisées pour effectuer la recommandation.

La seconde catégorie d'approches exploite directement le graphe social pour retrouver des contenus (*e.g.* sur un réseau social de botanique, un utilisateur peut vouloir récupérer les observations faites par ses amis). Le graphe est ici structuré de la manière suivante : $G = (U, I, E)$ où U représente les utilisateurs, I les objets et E les liens d'amitié entre utilisateurs et de partage entre utilisateurs et objets.

Un grand nombre de ces techniques s'appuyant sur les réseaux sociaux pour parcourir leur graphe utilisent l'hypothèse du petit monde. Cette hypothèse, vérifiée dans un premier temps en sociologie en 1967 [114] et confirmée plus tard [87], émet la prédiction suivante : si une personne ne peut envoyer un courrier qu'à ses connaissances et que celles-ci peuvent en faire autant jusqu'à ce que ce dernier arrive au destinataire, alors seul un petit nombre de sauts est nécessaire. La moyenne généralement avancée est six. En 2011, Facebook a montré que la longueur moyenne d'un lien entre deux utilisateurs était de $4,74$ [23]. À cela s'ajoute le fait que $99,91\%$ des utilisateurs sont inter-connectés. Ces dernières hypothèses permettent de garantir que les algorithmes qui parcourront le graphe social n'auront pas à faire plus de $4,74$ sauts en moyenne pour atteindre n'importe quel nœud. De manière similaire, elle permet de garantir que même si le nombre de sauts qu'un algorithme fait est réduit pour des raisons de performance, cela reste acceptable.

Ainsi, Amer-Yahia *et al.* [10, 12] proposent une architecture et un langage qui, à partir de ressources sociales (*i.e.* objets liés aux utilisateurs dans un réseau social), permet de recommander des contenus aux utilisateurs. *SocialScope* [12] agrège dans un premier temps les ressources de plusieurs sites sociaux qui pourront ensuite être requêtés par le langage Jelly [10]. Ainsi, il est possible de demander au système « lesquels de mes amis sont allés à Paris et ont visité la tour Eiffel ? » ou encore « où ont mangé mes amis qui sont allés à Paris visiter la tour Eiffel ? ».

Bilan des systèmes de recommandation et de recherche de l'information

Les méthodes décrites précédemment sont généralement décomposées en systèmes à base de mémoire qui exploitent l'historique des utilisateurs pour calculer un voisinage et effectuer les prédictions et, ceux à base de modèles qui, en découvrant des motifs généraux dans la matrice S, atteignent le même but. Quatre approches ont été mises en avant : les filtrages collaboratifs, basés sur les contenus, hybrides et basés sur les données sociales – sous-catégorie du premier point. Les premières, en exploitant les notes soumises par les utilisateurs à des objets, tentent de prédire celles qu'ils pourraient attribuer à des objets qu'ils n'ont pas encore évalués. Ces méthodes souffrent principalement de deux limitations, connues sous le nom de *démarrage à froid* : les utilisateurs n'ayant jamais évalué un objet et donc pour qui effectuer des prédictions devient compliqué, et les objets n'ayant été évalué par aucun utilisateur qui ne peuvent donc pas être recommandés. Le second groupe d'approches exploite le contenu des objets pour effectuer les mêmes prédictions. Cela permet de recommander des objets n'ayant jamais été évalués mais dont le contenu est proche de ceux qui sont attachés au profil d'un utilisateur par exemple. Le troisième point tente de combiner les deux méthodes afin d'en exploiter les meilleurs aspects. Enfin, le filtrage social permet d'aborder les scénarios où les utilisateurs n'ont jamais évalué d'objet. En effet, chaque utilisateur étant rattaché à une liste d'*amis*, ces derniers peuvent toujours servir à établir les prédictions.

Les méthodes suivantes, en exploitant les calculs résultants des précédentes contributions, vont retourner aux utilisateurs des listes de résultats, ou de recommandations, composées des meilleures prédictions.

2.1.3 Algorithmes de recherche et de recommandation

La plupart des contributions précédentes se focalisent sur la prédiction de scores entre un objet et un utilisateur. Généralement, la recommandation exploite cette information pour en retourner les meilleures prédictions, étant donné un utilisateur u.

Les solutions de *top-N* vont parcourir les structures de données déjà existantes, comme la matrice S afin de produire ces recommandations. À l'inverse, les solutions de type *top-k* vont s'appuyer sur des listes inversées (décrites plus loin) afin de calculer ces recommandations. L'utilisation de ces listes permet une plus grande interactivité avec l'utilisateur qui pourra par exemple soumettre des requêtes, ou se voir recommander des objets dynamiquement en fonction de ceux qu'il consulte à l'instant présent.

Algorithme 1: Calcul des *top-N* recommandations via une recommandation à base de contenus

Input:
M : la matrice $m \times m$ des k plus proches
U : un vecteur de taille m indiquant si un objet à été évalué par u
N : le nombre de recommandations souhaité

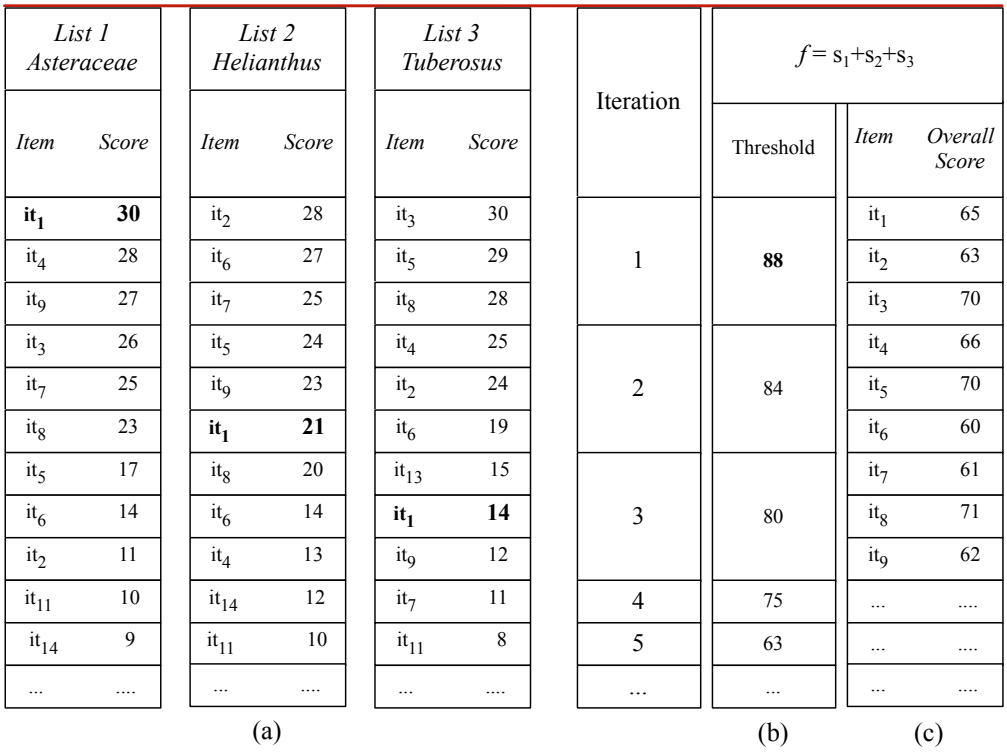
Output:
Les *top-N* recommandations.

```
1 begin
2     x ← MU;
3     for j ← 1 to m do
4         if U_j ≠ 0 then
5             x_i ← 0;
6     for j ← 1 to m do
7         if x_j ne fait pas partie des top-N recommandations dans x then
8             x_j ← 0;
9     return x;
```

Top-N

Étant donné l'ensemble des objets I et la matrice *Users* $\times$ *Items* S, le but d'une recommandation *top-N* est de recommander à l'utilisateur courant u un ensemble trié d'au maximum N objets qu'il n'a pas encore accédé [48, 79]. Ainsi, via une approche basée sur les contenus, la recommandation peut se produire en deux étapes, telles que décrites dans l'Algorithme 1 :

1. La première consiste à construire une matrice $M_{m \times m}$ telle que chaque ligne et colonne représentent un objet – m est donc le nombre d'objets. Toutes les cases indiquent dans un premier temps la similarité entre l'objet i et j si $i \neq j$. Puis, si j ne fait pas partie des k objets les plus similaires à i alors sa valeur est passée à 0. En définitive, on obtient une matrice M dont chaque colonne i contient les k objets le plus proches de i. Ramener à 0 tous les objets permet de réduire l'occupation mémoire de M et donc d'augmenter les performances.
2. La seconde étape consiste à appliquer ce modèle (*i.e.* utiliser la matrice) pour calculer les *top-N* recommandations. L'algorithme 1 présente une manière de calculer ces recommandations. L'algorithme prend en paramètre la matrice M et un vecteur de taille m indiquant par le chiffre 1, les objets évalués par l'utilisateur u, et par 0 ceux qui ne l'ont pas été. N est le nombre de recommandations souhaité. L'opération consiste à multiplier M par U pour obtenir un vecteur x. Chaque objet dans x aura une valeur dépendante



List 1 Asteraceae		*List 2 Helianthus*		*List 3 Tuberosus*		Iteration	$f = s_1+s_2+s_3$	
Item	*Score*	*Item*	*Score*	*Item*	*Score*		Threshold	*Item* *Overall Score*
it_1	**30**	it_2	28	it_3	30	1	**88**	it_1 65
it_4	28	it_6	27	it_5	29			it_2 63
it_9	27	it_7	25	it_8	28			it_3 70
it_3	26	it_5	24	it_4	25	2	84	it_4 66
it_7	25	it_9	23	it_2	24			it_5 70
it_8	23	**it_1**	**21**	it_6	19			it_6 60
it_5	17	it_8	20	it_{13}	15	3	80	it_7 61
it_6	14	it_6	14	**it_1**	**14**			it_8 71
it_2	11	it_4	13	it_9	12			it_9 62
it_{11}	10	it_{14}	12	it_7	11	4	75	... .
it_{14}	9	it_{11}	10	it_{11}	8	5	63	... .
...	....	...	....	...	....	...	...	... .

(a) (b) (c)

Figure 2.5: (a) listes inversées. (b) valeur du seuil en fonction de la position. (c) score global d'un objet.

de sa similarité avec les objets accédés par u. Ces derniers sont ensuite mis à 0 afin de ne pas les recommander. Puis, en dernière instance, les objets ne faisant pas partie du *top-N* sont mis à 0. Finalement, le vecteur ne contient que les *top-N* objets les plus similaires aux objets déjà évalués par u.

Top-k

Les algorithmes de *top-k* [7, 54, 9] offrent une plus grande interactivité avec l'utilisateur en lui permettant de soumettre des requêtes, directement (*e.g.* requête à mots clés) ou indirectement (*e.g.* les objets qu'il consulte constitueront la requête [96]). Il s'agit donc de lui retourner, étant donné une requête q, les k objets maximisant un score.

Afin de calculer un *top-k*, il est nécessaire de construire un ensemble de listes inversées [7]. Ces listes associent un attribut (*e.g.* un mot clé) aux objets qui le contiennent, accompagnés de leur score par rapport à cet attribut ; un objet dont le score serait 0 n'est tout simplement pas présent dans la liste. Enfin, les contenus de chaque liste sont triés par score décroissant. La Figure 2.5 présente des listes inversées, et la première liste contient les objets (*i.e. items*) contenant le mot « Asteraceae » associés à leur score par rapport à ce mot (*e.g.* similarité

entre l'objet et le mot).

Le *top-k*, décrit dans l'Algorithme 2 et illustré par la Figure 2.5 fonctionne de la manière suivante. Supposons que l'objectif soit de calculer les 5 meilleurs objets (*i.e. top-5*). Il s'agit dans un premier temps de calculer l'ensemble des listes inversées qui seront utilisées dans le calcul, et qui forment l'entrée de l'algorithme ; dans le cas où la requête est à mots clés, les listes sont celles associées à chacun des mots de la requête. Dans l'exemple de la Figure 2.5, la requête « Asteraceae Helianthus Tuberosus » (*i.e.* espèce de tournesol) a été soumise, et les listes inversées correspondants à chacun de ces mots sont donc choisies. L'algorithme

Algorithme 2: Calcul des *top-k* recommandations étant donné une requête q

Input:
L_q : les listes inversées pour la requête q
k : le nombre de résultats souhaités
Output:
R^q : La liste des k meilleurs résultats

```
1 begin
2     R^q ← ∅
3     repeat
4         1. sorted access in parallel in each list in L_q
5         2. given each item accessed in 1 in a list L ∈ L_q, perform a random
              access in the other lists in L_q to compute its full score
6         3. maintain in R^q the k best items seen until now
7         4. compute the threshold δ
8     until |R^q| = k and score(it) ≥ δ ∀it ∈ R^q
9     return R^q
```

va effectuer successivement des accès triés en parallèle dans chacune des listes inversées fournies en entrée (ligne 4). Un accès trié (*i.e. sorted access* ou *SA*), consiste à récupérer les objets d'une liste dans l'ordre (*i.e.* du premier vers le dernier). Dans la figure, les objets récupérés par les premiers accès triés sont it_1, it_2 et it_3. Lorsqu'un objet est récupéré, un accès aléatoire est réalisé dans chacune des autres listes fournies en entrée. Un accès aléatoire (*i.e. random access* ou *RA*) consiste à récupérer directement le score d'un objet dans une liste donnée. Enfin, les scores obtenus par l'accès triés et par les accès aléatoires sont combinés (*e.g.* somme, min, max) afin d'obtenir le score final de l'objet. Dans la figure, la combinaison est la somme et le score de it_1 est $30 + 21 + 14 = 65$. L'algorithme répète ces opérations jusqu'à obtenir k objets dont le score est supérieur à un seuil δ. Un seuil définit le score maximum potentiel que les objets non encore accédés peuvent obtenir ; il est calculé comme l'agrégation des scores (la méthode d'agrégation/combinaison doit être la même que celle utilisée pour le calcul du

score des objets) correspondant aux derniers accès triés faits dans chaque liste. Ainsi, dans la figure, lors du premier accès, le seuil vaut $\delta = 28 + 28 + 30 = 88$. Après 5 itérations, on peut observer que les 5 meilleurs objets (*i.e.* dont le score est le meilleur) ont un score supérieur au seuil $\delta = 63$. L'algorithme a donc calculé un *top-5* et peut s'arrêter.

Le *top-k* peut également s'appliquer aux approches collaboratives où les résultats dépendent des profils utilisateurs [9] (*e.g.* le score d'un objet est par exemple la note qu'un utilisateur lui a donné, ou une prédiction).

Bilan des algorithmes de recherche et de recommandation

Deux *points de vue* ont été présentés : les approches automatiques *top-N*, qui, en exploitant la matrice S, vont retourner les N meilleures prédictions à chaque utilisateur, et les approches interactives (*i.e.* nécessitant une requête soumise par un utilisateur) *top-k*, qui vont retourner les k objets maximisant un score – étant donné la requête, le profil utilisateur ou d'autres choses.

Malheureusement, les résultats issus de ces méthodes sont généralement redondants [176]. Premièrement, lorsque la matrice S est exploitée, la probabilité que les recommandations concernent des objets déjà populaires (et donc ne nécessitant que peu de recommandations) est forte ; ces objets représentent en effet la majeure partie des notes de la matrice S. Deuxièmement, lorsque leur contenu est exploité, les recommandations font référence à des objets très proches et très similaires. Enfin, puisque le profil de l'utilisateur est utilisé dans le calcul des recommandations (*i.e.* recommandation d'objets similaires à ceux partagés ou évalués par l'utilisateur courant, ou provenant d'utilisateurs très similaires), ces dernières sont finalement très proches des précédents partages ou évaluations de l'utilisateur, limitant par là l'effet de découverte.

La section qui suit introduit aux méthodes de diversification et de sérendipité qui ont pour but d'aborder les trois points limitant précédemment évoqués.

2.1.4 Diversification et sérendipité

Les systèmes de recherche et de recommandation font face à deux problèmes majeurs. Le premier est que les résultats recommandés aux utilisateurs sont souvent très redondants et proviennent d'objets déjà populaires. La Figure 2.1, présentée en page 14, montre ce problème ; la première colonne ne prend pas en compte la diversification et les résultats sont très similaires entre eux. Ce problème de redondance est généralement abordé par des méthodes dites de diversification [13, 15, 112, 119] : il s'agit ici de retourner aux utilisateurs des listes de recommandations dont les résultats sont les plus pertinents possibles mais également les plus divers entre eux. Dans le second cas, les utilisateurs ne reçoivent des recommandations ne provenant que d'objets très similaires à ceux qu'ils ont déjà consultés, limitant donc l'effet bénéfique de la recommandation : la découverte

de nouveaux contenus. Ce point est traité par des méthodes de « sérendipidé », c'est-à-dire entraînant un *hasard heureux* qui retournerait des résultats pertinents mais inattendus à l'utilisateur : ces méthodes consistent à recommander des objets distants de ceux déjà consultés [2, 11, 59, 73, 98, 112].

La première sous-section suivante présente les méthodes de diversification et la suivante, celles de sérendipité.

Méthodes de diversification

La diversité a été abordée dans le cadre de nombreux contextes, chacun offrant une problématique différente. Cette section présente ces approches au travers de deux principaux domaines : la recommandation, où le profil de l'utilisateur est exploité dans le calcul des résultats, et la recherche d'information, où les requêtes soumises par les utilisateurs sont exploitées.

Plusieurs caractéristiques et méthodes de diversifications ont été proposées pour les moteurs de recommandation :

- **Diversification des thèmes [169, 178, 179] :** généralement, sur les plateformes de *e-commerce*, tous les objets sont structurés par thème (*e.g.* livre, livre/action, informatique). L'objectif est donc de combiner la pertinence des recommandations avec leur couverture des thèmes existants (*i.e.* représentativité des résultats pertinents). Ziegler *et al.* [179] proposent une approche dans laquelle deux listes de N résultats sont calculées : R_p regroupe les plus pertinents et R_d contient les plus divers étant donné une mesure de similarité. Les deux listes étant triées par score décroissant. Le score d'un objet it est la moyenne pondérée – la pondération est définie par l'utilisateur – de sa position dans les deux listes, et les k meilleurs sont retournés – par l'application d'un *top-k*.
- **Diversification des commentaires [1] :** Abbar *et al.* proposent une approche de la diversité d'articles d'actualité. En se basant sur les plateformes d'actualité, ils ont montré que les commentaires des utilisateurs étaient de meilleurs indicateurs des thématiques abordées dans un article que leur contenu même et donc de meilleurs objets pour mesurer la diversité. Ici, l'objectif est de recommander à un utilisateur u, lisant un article it, un ensemble de k articles, notés R, à la fois pertinents (*i.e.* similaires) par rapport à it et divers. Ce dernier point est calculé en utilisant *MaxMin* [36, 62], c'est-à-dire en maximisant la distance minimale qu'il peut exister entre deux articles de R :

$$R = \underset{R \subset I, |R|=k}{\arg\max} \min_{it_1, it_2 \in R} dist(it_1, it_2) \tag{2.9}$$

  Où I est l'ensemble des articles et *dist* une mesure de distance qui peut être calculée comme l'inverse de la similarité entre deux articles. Rappelons

que l'article *it* est la requête qui sera utilisée dans le calcul de l'ensemble divers. Le hachage est ensuite utilisé pour traiter les requêtes. L fonctions de hachages g_i sont définies, chacune étant une combinaison de m fonctions de hachage. L paniers de m groupes chacun sont ainsi créés. L'application d'une fonction g_i permet d'attribuer à un objet *it* un groupe du panier i. Chacun de ces groupes ne contient que des articles similaires. Ils sont ensuite réduits à une taille s, de telle manière que les s objets maximisent la diversité du groupe. Le traitement d'une requête *it* est le même que celui de son indexation : les paniers sont récupérés, puis, un sous-ensemble R maximisant la diversité en est déduit.

- **Diversification des utilisateurs [176] :** sur certaines plateformes, les attributs décrivant chaque objet peuvent être insuffisants, voire erronés. Les *signets* (*i.e. bookmarks*) partagés sur la plateforme *Delicious* ne se composent que d'un nombre très réduit d'étiquettes (*i.e.* tags) soumises par les utilisateurs. Toutes les recommandations sont ici associées à une liste d'explications. Dans le cadre d'un filtrage collaboratif, ces explications (*i.e. explanation-based*) sont constituées de l'ensemble des utilisateurs *neighbors(u)* utilisés dans le processus de recommandation, partageant un objet donné :

$$Expl(u, it) = \{u' \in neighbors(u) | it \in Items(u')\} \tag{2.10}$$

Une recommandation est donc un couple composé de l'objet recommandé *it* et de l'explication de cette recommandation :

$$\langle it, Expl(u, it) \rangle \tag{2.11}$$

L'objectif de cette approche est de diversifier ces listes d'explications. Afin de calculer ces résultats de manière efficace, le processus de diversification est opéré en deux étapes : calcul d'un *top-N* des recommandations les plus pertinentes, puis d'un *top-k* dont les résultats, extraits du premier ensemble, sont divers et sont notés R. Deux méthodes principales permettent d'obtenir ce but. Les algorithmes gloutons (*i.e. greedy*) vont d'abord initialiser R avec les k objets maximisant la pertinence, issus du *top-N*, puis, itérativement, vont éliminer les objets it_i dont leur similarité avec un autre objet du *top-N* est supérieure à un seuil prédéfini δ. Ce dernier est ensuite remplacé par l'objet suivant de la liste de taille N. L'algorithme s'arrête lorsque tous les objets satisfont cette condition de seuil ou que les N objets de la première liste ont été accédés.

D'un autre côté, les algorithmes d'échange (*i.e. swap*) vont aussi initialiser une liste R avec les k objets maximisant la pertinence. Puis, itérativement, ils vont sélectionner l'objet minimisant la diversité et vont le remplacer avec l'objet suivant de la liste de taille N. Un seuil limitant la perte maximale de pertinence indique quand l'algorithme doit s'arrêter. En effet, chaque fois qu'un objet est remplacé, il l'est par un autre moins pertinent.

- **Théorie des graphes [60, 64] :** le problème de diversification est très proche des problèmes de dominance dans les graphes. Il s'agit ici, étant donné un graphe $\mathcal{G} = (S, A)$, où S représente l'ensemble des sommets et A l'ensemble des arrêtes, de trouver un sous-ensemble $D \subseteq S$ tel qu'il n'existe pas deux sommets dans D connectés par une arrête dans A et qu'il n'existe pas d'autres sommets dans $s \in S \backslash D$ qui ne seraient pas connectés via une arrête à un sommet de D. En d'autres termes, supposons que les arrêtes représentent la similarité entre les sommets, il s'agit de trouver le plus grand ensemble complètement divers. De nombreux algorithmes et heuristiques ont été proposés pour ce problème [60].
- **Diversification par radius [50] :** l'objectif de cette méthode n'est pas de calculer un sous-ensemble de k objets maximisant la diversité – et la pertinence –, mais de définir comme paramètre cette diversité r (*i.e.* radius) et de calculer le plus petit ensemble $\mathcal{S}^\star$ tel que :
  1. pour tout objet $it_i \in \mathcal{S}^\star$ il existe un objet $it_j \in I$ tel que $dist(it_i, it_j) \leq r$, où $\mathcal{I}$ est l'ensemble des objets (*i.e. items*) ;
  2. Pour chaque objet $it_i, it_i \in \mathcal{I}$ on ait $dist(it_i, it_j) > r$.

  De manière plus formelle, le problème est le suivant : étant donné l'ensemble des objets $\mathcal{I}$, il s'agit de trouver un sous-ensemble $\mathcal{S}^\star$ satisfaisant les deux précédents points, tel que pour tout autre ensemble $\mathcal{S}^{\star'}$ on ait la propriété suivante : $f(\mathcal{S}^\star) \leq f(\mathcal{S}^{\star'})$ où $f(\mathcal{S}^{\star'}) = |\mathcal{S}^{\star'}|$.

D'un autre côté, dans le cadre de la recherche d'information, des techniques différentes ont également été proposées afin d'introduire de la diversité dans les résultats :

- **Diversification par reformulation de la requête [133] :** l'objectif initial de cette méthode est de rajouter de la personnalisation dans les résultats d'une recherche. Lorsqu'un utilisateur u soumet une requête q, le moteur de recherche va aussi exécuter k requêtes sémantiquement proches de q, notées $C(q)$ (*i.e. close to* q). Les requêtes $C(q)$ sont déterminées grâce au comportement des utilisateurs. Ainsi, si un grand nombre d'entre eux soumet la requête q et la reformule ensuite en q', alors q' est candidate pour faire partie du groupe $C(q)$. Chaque reformulation q_i est ainsi associée à une probabilité p_i^q d'appartenir à $C(q)$. Cette probabilité est proportionnelle au nombre de fois que q a été reformulée en q_i. Finalement, lors du calcul des *top-100* meilleurs résultats, seuls les $\frac{100}{k+1}$ résultats de chaque requête sont conservés – exceptée la requête q initiale, pour laquelle, les 100 premiers résultats sont conservés – et classés en fonction de la proximité du résultat avec le profil de u. La valeur k est pré-définie. Radlinski et Dumais [133] ont choisi des valeurs dans l'intervalle suivant : $k \in \{0, 2, 4, 9, 19\}$.

- **Diversification par échantillonnage [15] :** la motivation de l'échantillonnage est dû à la taille du corpus à rechercher. Les auteurs proposent de réduire l'homogénéité des résultats d'une recherche en utilisant un algorithme d'échantillonnage tel que chaque objet pertinent a une certaine probabilité d'apparaître dans la liste R^q finale. L'algorithme exploite les listes inversées pertinentes pour la requête q mais y sélectionne aléatoirement les résultats.

- **Diversification par *clustering* [51, 91] :** l'objectif de la classification (*i.e. clustering*) est de présenter à l'utilisateur les différentes thématiques présentes dans la liste des résultats R^q. Pour cela, les auteurs proposent de réaliser une taxonomie à la fois compacte, mais couvrant l'ensemble des thématiques présentes dans le corpus global. Chaque résultat doit être choisi de manière à maximiser la couverture des concepts pertinents pour la requête q, mais également dans le but de couvrir le maximum de concept relié aux autres objets pertinents. En d'autres termes, il faut maximiser le nombre d'objets dont les concepts pertinents pour q sont représentés par un résultat dans R^q.

- ***Top-k* diversifié [17, 36, 39, 62] :** les *top-k* diversifiés utilisent l'idée que chaque utilisateur accède aux résultats d'une liste dans un ordre particulier. Ainsi, supposons une liste de résultats R^q, l'objet $it_i \in R^q$ doit être choisi de manière à maximiser la pertinence par rapport à q ainsi que sa diversité par rapport aux objets $it_1, ..., it_{i-1}$, déjà vus par l'utilisateur [36] :

$$it_i = \underset{it_i \in I, it_i \notin R^q}{\arg\max} [\lambda \times sim(it_i, q) - (1 - \lambda) \times \max_{it_j \in R^q} (sim(it_i, it_j))] \tag{2.12}$$

  Où la partie sur la diversité (*i.e.* minimisation de la distance maximum) peut être remplacée par n'importe quelle fonction de diversification. Une approche probabiliste est possible à partir des mêmes idées :

$$\mathcal{P}\left(S_{R^q}\right) = \mathcal{P}\left(\bigcup_{it_i \in R^q} S_{it_i}\right) \tag{2.13}$$

  Où $\mathcal{P}\left(S_{R^q}\right)$ est la probabilité que les k résultats R^q, provenant de la requête q, satisfassent l'utilisateur. $\mathcal{P}\left(S_{r_i}\right)$ est la probabilité que l'objet $it_i \in R^q$ convienne à l'utilisateur. La probabilité que la liste de résultats R^q satisfasse l'utilisateur est donc la probabilité que chacun des résultats en faisant partie le satisfasse (*i.e.* union). Cette probabilité peut se développer de la manière suivante – on suppose que la probabilité que deux objets soient divers est

indépendante des autres objets [17] :

$$\begin{aligned}\mathcal{P}\left(\bigcup_{it_i \in R^q} S_{it_i}\right) &= \sum_{i \in 1,\dots,k} \mathcal{P}\left(S_{it_i}\right) - \mathcal{P}\left(S_{it_i} \cap \left(S_{it_1} \cup \dots \cup S_{it_{i-1}}\right)\right) \\ &= \sum_{i \in 1,\dots,k} \mathcal{P}\left(S_{it_i}\right) \times \left(1 - \mathcal{P}\left(S_{it_1} \cup \dots \cup S_{it_{i-1}} | S_{it_i}\right)\right) \\ &= \sum_{i \in 1,\dots,k} \mathcal{P}\left(S_{it_i}\right) \times \mathcal{P}\left(\overline{S_{it_1}} \cup \dots \cup \overline{S_{it_{i-1}}} | S_{it_i}\right) \\ &= \sum_{i \in 1,\dots,k} \mathcal{P}\left(S_{it_i}\right) \times \prod_{j \in 1,\dots,i-1} \left(1 - \mathcal{P}\left(S_{it_j} | S_{it_i}\right)\right)\end{aligned} \tag{2.14}$$

Où la probabilité $\mathcal{P}\left(S_{it_i}\right)$ peut s'exprimer comme la similarité entre it_i et la requête q, notée $sim(it_i, q)$ et la probabilité $\mathcal{P}\left(S_{it_j} | S_{it_i}\right)$ peut s'exprimer comme la similarité entre l'objet it_j et it_i, notée $sim(it_j, it_i)$. En d'autres termes, le score d'un objet, en fonction de la liste de résultats R^q,

$$score(it_i, q, R) = sim(it_i, q) \times \prod_{j \in 1,\dots,i-1} 1 - sim(it_i, it_j) \tag{2.15}$$

Où la similarité entre deux objets prend sa valeur entre 0 et 1, permettant donc d'utiliser le seuil d'un algorithme de *top-k* normal. L'algorithme 3 présente ainsi un *top-k* diversifié. Il va d'abord effectuer un accès dans les listes inversées (ligne 3) et calculer le score diversifié de l'objet récupéré, noté it, en prenant compte des objets déjà émis, c'est-à-dire déjà dans R^q. Le score d'un objet v est tout d'abord borné en fonction des objets déjà dans R^q (ligne 5). Finalement, si un objet déjà accédé dans les listes inversées mais qui n'a pas encore été ajouté dans R^q a son score supérieur au seuil δ, alors il est émis (ajouté dans R^q) tandis que le score borné des autres objets est mis à jour en tenant compte de leur diversité par rapport à it (ligne 8). L'algorithme s'arrête lorsque k objets ont été émis.

Méthodes de sérendipité

La sérendipité est un problème très proche de la diversité. Il s'agit, étant donné un utilisateur u, de trouver un ensemble de recommandations R tel qu'aucune d'entre elles ne soient proches des objets déjà évalués ou partagés par u (à l'inverse de la diversité où celles-ci doivent être différentes des autres recommandations plutôt que des objets évalués ou partagés par u) [2, 59, 73, 98, 112]. Plusieurs propositions ont été faites dans ce sens :

- ***Clustering* (*i.e. OTB*) [2] :** l'idée est de regrouper chaque objet dans un groupe (*i.e. cluster*) en utilisant son contenu (ou le profil des utilisateurs l'ayant évalué) et un algorithme de *clustering*. La notion d'ordre, ou de source, consiste à associer un groupe c_1 à un groupe c_2, si les utilisateurs

Algorithme 3: *top-k* diversifié

Input:
L_q : les listes inversées pour la requête q
k : le nombre de résultats souhaités
Output:
R^q : La liste des k meilleurs résultats divers

```
1 begin
2     R^q ← ∅
3     repeat
4         1. sorted access in parallel in each list in L_q
5         2. given each item accessed in 1 in a list L ∈ L_q, perform a random
          access in the other lists in L_q to compute its full score taking into
          account diversity with respect to R^q
6         3. compute the threshold δ
7         4. if the best item it ∉ R^q accessed in 1 has a score superior to δ
          then, add it to R^q
8     until |R^q| = k
9     return R^q
```

qui accèdent les contenus dans c_1 ont également tendance – étant donné un seuil – à accéder ceux présents dans c_2. La familiarité entre un groupe d'objets et un utilisateur dépend du nombre d'objets que ce dernier ou son voisinage ont accédé en son sein. Chaque utilisateur u est associé à un ensemble de groupes familiers C_u dans lesquels il a évalué ou accédé des objets. Enfin, il s'agit de recommander des objets provenant de groupes qui ne sont pas familiers à u, mais dont une partie de leurs sources se situent dans C_u.

- **Aléatoire [98] :** le manque de surprise des systèmes de recommandation vient du fait qu'en prédisant tellement bien le comportement de l'utilisateur, les objets recommandés lui sont déjà connus et les recommandations prévisibles. En introduisant un facteur aléatoire, l'imprécision engendrée n'est pas suffisante pour retourner de mauvaises recommandations, mais permet ainsi d'augmenter la surprise de l'utilisateur.
- **Basé sur les anomalies [73, 98] :** cette contribution est basée sur les réseaux Bayésien. Deux classes sont proposées : recommandation *intéressante* (classe c^+) pour l'utilisateur, ou *non-intéressante* (classe c^-). L'objectif est d'effectuer les recommandations les plus inattendues à l'utilisateur. Étant donné un objet *it* et un utilisateur u, si le réseau Bayésien retourne une forte probabilité que l'objet appartienne à la classe c^+, alors cela signifie que celui-ci est très proche du profil de u et donc, a une forte probabilité de ne pas le surprendre. À l'inverse, si le classifieur Bayésien retourne une

forte probabilité que *it* appartienne à la classe c^-, alors il y a une forte probabilité que l'objet ne corresponde pas aux goûts de l'utilisateur et soit une mauvaise recommandation. Finalement, si le classifieur ne parvient pas à déterminer avec certitude la classe d'appartenance de l'objet *it*, alors on se situe dans un cas où *it* est relativement pertinent mais aussi relativement distant du profil de u : *it* est un objet qui a une forte probabilité de surprendre positivement u.

Bilan des méthodes de diversification et de sérendipité

Deux approches ont été présentées ici. Tout d'abord, la diversification a pour but de présenter à l'utilisateur des résultats distants les uns des autres. Cette distance est généralement déduite de méthodes de similarité. Deux aspects sont généralement abordés par les contributions : le modèle de diversification et l'algorithme permettant le calcul de résultats divers. Le calcul d'un ensemble maximisant ces modèles est généralement *NP-Complet* [17] et les contributions se concentrent sur la réalisation d'heuristiques. Le second point concerne la sérendipité. L'objectif de ces méthodes est d'introduire une distance entre les résultats présentés aux utilisateurs et leur profil. Ces contributions sont plus difficiles à mettre en place puisqu'elles s'opposent directement à la notion de recommandation : il s'agit de recommander des objets à la fois proches et distants du profil utilisateur. Proches car il est important de satisfaire la notion de prédiction/recommandation et distants car ils doivent apporter de la nouveauté.

2.2 Systèmes de gestion de données distribués

Cette section présente les différentes topologies *P2P* ainsi que les algorithmes qui leur sont rattachés. Celle-ci commence par une présentation générale accompagnée de définitions en Section 2.2.1. Les différentes catégories de réseaux distribués *P2P* sont ensuite présentées en Section 2.2.2. Enfin, une attention supplémentaire est accordée aux techniques de réplication de données au sein de ces réseaux, en Section 2.2.3.

2.2.1 Présentation et définitions

Nous présentons, au sein de cette section, les systèmes, ou réseaux, *P2P* (*i.e.* pair-à-pair). Ils sont une approche alternative aux applications centralisées, dans le sens où chaque nœud, ou pair, peut faire à la fois office de client et de serveur. Les échanges se font donc directement entre nœuds du réseau, évitant par là une entité (*i.e.* serveur) centrale. Il s'agit donc d'un paradigme de structure du réseau – on parle souvent de réseau *P2P*. De manière plus précise, le *P2P* opère au niveau logique en établissant des liens entre chaque pair. L'ensemble des pairs

dont a connaissance un pair u (*i.e.* il existe un lien provenant de u vers ces pairs) définit son voisinage.

De manière plus précise, un réseau pair-à-pair peut s'exprimer à partir d'un graphe $G = (U, E)$, où $U = \{u_1, ..., u_n\}$ est l'ensemble des utilisateurs disponibles sur le réseau et $E = \{e_1, ..., e_k\}$ les liens entre utilisateurs.

Définition 3 (Voisinage *P2P* ou *P2P neighborhood*)**.**

Le voisinage *P2P* d'un pair u fait référence à tous les autres pairs dont u a connaissance. De manière plus précise, u a connaissance d'un pair v s'il existe une arête $e(u, v) \in E$ qui relit u à v.

Le réseau formé par l'ensemble des liens E s'appelle le *recouvrement* ou *overlay*. Il s'agit, de manière plus précise de l'union des voisinages de chaque pair présent dans le réseau. Notons que le voisinage d'un pair est exploité afin de faire transiter les messages, tels que des requêtes et leurs résultats dans un cadre de recherche d'information, sur le réseau. Nous utilisons par la suite la notion de couverture définie ci-après. Il s'agit d'une évaluation de la qualité du voisinage d'un pair u, établie comme la probabilité que celui-ci (*i.e.* le voisinage) puisse répondre (*i.e.* retourner des résultats, ou objets, pertinents étant donné une requête) aux requêtes soumises par u.

Définition 4 (Couverture ou *coverage*)**.**

Étant donné Q, l'ensemble des requêtes (*e.g.* à mots clés, *exact-match*) qui peuvent être soumises au système et $neighbors(u) = \{v_1, ..., v_n\}$, les pairs présents dans le voisinage de u, la couverture est la probabilité qu'au moins un de ces derniers $\{v_1, ..., v_n\}$ puisse retourner un résultat pertinent (*e.g.* des objets pertinents en recherche et en recommandation) étant donné une requête $q \in Q$. Cette probabilité peut se formuler de la manière suivante : $\mathcal{P}(Q^u_{v_1} \cup Q^u_{v_2} \cup ... \cup Q^u_{v_n})$ où $\mathcal{P}(Q^u_{v_1})$ est la probabilité que v_1 puisse répondre aux requêtes de u.

Les systèmes *P2P* offrent de nombreux avantages :

- **Passage à l'échelle (*i.e. scalability*)** : les performances du système restent stables quelle que soit sa taille.
- **Tolérance aux pannes** : le système doit être capable de maintenir ses fonctionnalités malgré les pannes (*i.e.* déconnection) de certains nœuds.
- **Dynamicité** : le système doit pouvoir s'adapter aux modifications d'états comme l'arrivée d'un nouveau pair.

- **Auto-organisation ou organisation autonome** : le système ne doit pas dépendre d'entités extérieures ou centralisées pour fonctionner.
- **Contrôle décentralisé** : la gestion du système ne doit pas dépendre d'un seul pair mais doit être décentralisée à l'ensemble d'entre eux.

Ainsi, puisque les ressources sont proportionnelles au nombre de pairs impliqués dans le réseau (*i.e.* passage à l'échelle), les systèmes *P2P* permettent de réaliser des plateformes de gestion de données à bas coût [94, 143], sans avoir recours à d'entités centrales.

Ces systèmes peuvent être classés au sein de plusieurs catégories [123, 124, 138] du type de topologie utilisée : réseaux structurés ou non, avec super-pairs et dynamiques. Nous présenterons par la suite uniquement les catégories « réseau non-structuré », « réseau structuré » et « réseau dynamique ».

2.2.2 Catégories de réseaux *P2P*

Les quatre catégories de réseaux sont maintenant introduites.

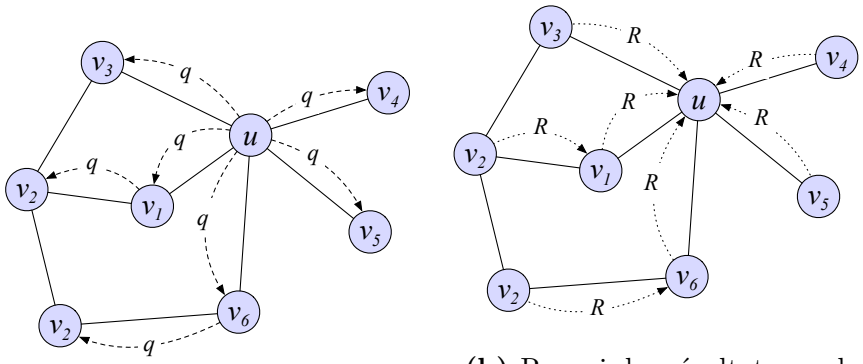
Réseaux *P2P* avec topologie non-structurée

Les réseaux avec topologie non-structurée représentent l'ensemble des solutions *P2P* où le voisinage de chaque nœud, ou pair, d'une part, et le placement des données, d'autre part, ne souffrent que de peu de restrictions [43]. De cette manière, chaque pair connaît un sous-ensemble des pairs disponibles sur le réseau, son *voisinage* (*i.e. neighbors*), généralement choisis aléatoirement sur un serveur de *bootstrap* – lorsqu'un pair rejoint le réseau, il contacte ce serveur. Chaque pair partage ses propres données qu'il stocke localement et le placement des autres données lui est inconnu.

Plusieurs méthodes ont été mises en place pour le traitement des requêtes [165] comme l'inondation (*i.e.* BFS) présentée plus bas. L'objectif de chacune de ces méthodes est de limiter le coût de la requête, tout en maximisant le rappel (*i.e. recall*). Ce coût représente le nombre de sauts qu'elle doit faire sur le réseau – et donc, le nombre de pairs impliqués. Le rappel représente la fraction des objets pertinents qu'il a été possible de récupérer lorsque la requête a été soumise :

$$recall(q) = \frac{|R^q \cap Rel^q|}{|Rel^q|} \tag{2.16}$$

Où R^q représente l'ensemble des objets récupérés lors du traitement de la requête q et Rel^q représente tous les objets pertinents présents dans le système. Un certain nombre de contributions se focalisant sur le traitement des requêtes sont présentées ici :



(a) Propagation de la requête q.

(b) Renvoi des résultats par le même chemin que la requête q.

Figure 2.6: Traitement des requêtes dans les réseaux *P2P* avec topologie non structurée.

BFS (*i.e. Breath-First Search*) [78] : lorsqu'un utilisateur u soumet une requête q, celle-ci est propagée au voisinage de u récursivement (cf. Figure 2.6a). Le nombre maximum de sauts, ou récursions, que peut faire la requête est défini par le système : *TTL* (*i.e. Time To Live*). Chaque pair ayant reçu la requête calcule localement les résultats de celle-ci, *QueryHit*, et les transmet par le même chemin qu'a suivi la requête (cf. Figure 2.6b).

***Iterative Deepening* [175] :** cette méthode est une adaptation de *BFS* où la requête était directement propagée sur le réseau avec une grande valeur de *TTL*. *Iterative Deepening* est un *BFS* progressif. Pour cela, plusieurs exécutions de *BFS* sont nécessaires, chacune ayant une valeur plus grande de *TTL* (*e.g.* 1, 2, 3 puis 4). L'exécution de la requête s'arrête lorsque des résultats pertinents sont trouvés ou qu'un *TTL* maximum est atteint.

***Random walks* [100] :** les marches aléatoires sont ici exploitées. Les nœuds sont structurés de manière à ce que le degré de connexion (*i.e.* nombre de voisins) de chacun dépende du nombre de requêtes qu'ils sont capable de traiter, étant donné un espace de temps prédéfini. Les marches aléatoires sont également fixées par une valeur *TTL*. Finalement, étant donné un pair u, chaque autre pair p dans le voisinage de u est associé à une valeur définissant le nombre maximum de fois que u peut le contacter dans un espace de temps donné (*i.e.* valeur prédéfinie). Les pairs maximisant cette probabilité reçoivent la requête.

Autre : à cela s'ajoutent de nombreux autres algorithmes, tels que APS (*Adaptive Probabilist Search*) [165], *Local indices* [45, 174], basés sur les filtres de Bloom [137] ou encore DRLP (*i.e. Distributed resource location protocol*) [113].

Réseaux *P2P* avec topologie structurée

Les réseaux avec topologie structurée font très souvent référence aux *DHTs* (*i.e.* Tables de Hachage Distribuées). À l'inverse des réseaux avec topologie non-structurée, ici, les données et les utilisateurs sont positionnés de manière prédéfinie. Ce genre de structures propose principalement une fonction de *lookup* (*i.e.* recherche), qui, à partir d'une clé *key* retourne le nœud responsable de celle-ci. Plus généralement, dans le cadre de la gestion de données, l'opération se décompose en deux étapes. Dans l'optique de stockage d'une valeur, celle-ci est d'abord transformée en clé :

$$key = hash(value) \tag{2.17}$$

Puis, la *DHT*, via son mécanisme de *lookup*, permet de retrouver le nœud qui est responsable de ce couple *(key,value)* :

$$node = lookup(key) \tag{2.18}$$

Pour cela, chaque nœud est représenté par un identifiant unique et chaque approche structure les nœuds entre eux en fonction de cet identifiant. Contrairement aux réseaux avec topologie non-structurée, l'objectif n'est pas de maximiser le *recall* qui est systématiquement de 1 (les algorithmes étant déterministes dans leur localisation), mais uniquement d'optimiser le coût de traitement de la requête. Un nombre varié d'approches a été proposé [63] :

Anneau (*i.e. ring*) [47, 156] : un représentant de cette approche est *Chord*. Les nœuds sont positionnés les uns après les autres, par identifiant croissant ; le nœud ayant l'identifiant le plus grand est quant à lui connecté avec celui qui a l'identifiant le plus petit. Un nœud est responsable de l'ensemble des clés dont la valeur est inférieure ou égale à son identifiant mais supérieure à celui du nœud précédent. Afin d'accélérer l'opération de *lookup*, chacun d'entre eux est associé à une table de routage (*e.g. finger table*) composée de k lignes associant toutes une clé au nœud qui en est responsable. La valeur de ces clés suit l'équation suivante :

$$key = nodeId + 2^i \tag{2.19}$$

$0 \leq i < k$ étant le numéro de la ligne. Finalement, l'opération de *lookup* consiste à envoyer une requête $q = key$ de manière récursive au nœud ayant le plus grand identifiant inférieur à la clé. L'opération s'arrête lorsque le nœud situé après le nœud courant a un identifiant supérieur à la clé.

Comme pour la plupart des implémentations de *DHT*, la structure en anneau permet un *lookup* avec une complexité de $\mathcal{O}(log(n))$.

Autres : plusieurs autres topologies de *DHTs* ont été proposées comme l'arbre (*i.e. tree*) [63], l'hypercube [134], en forme de fleur [53], de papillon [106] ou encore, sous forme de structures hybrides [139]. Avec l'arrivée du *NoSQL*, les *DHTs*

sont revenues à l'ordre du jour avec des implémentations modifiées comme *Dynamo* de Amazon [47]. Kademlia [111] (*i.e. XOR*) est aujourd'hui utilisée pour la diffusion de données sur le réseau *Bittorrent*. Chaque utilisateur effectue périodiquement l'opération $put(key, ip)$, signifiant qu'à son adresse *IP*, l'objet avec la clé *key* est disponible. À l'inverse, un utilisateur voulant télécharger un objet effectuera un $get(key)$ pour récupérer la liste des utilisateurs le partageant.

Deux remarques peuvent être formulées à propos des réseaux structurés. Tout d'abord, bien que les *DHTs* offrent une opération de *lookup* efficace de complexité $\mathcal{O}(log(n))$, satisfaisante pour la recherche exacte (*i.e. exact-match queries*), celles-ci ont plutôt du mal à s'adapter aux requêtes plus complexes (*e.g.* à base de mots clés) où le positionnement des données n'est plus évident. En second lieu, l'autonomie de chaque nœud se réduit, puisqu'il ne choisit pas son recouvrement et se conforme à une règle (*i.e.* topologie).

Réseaux *P2P* avec topologie dynamique

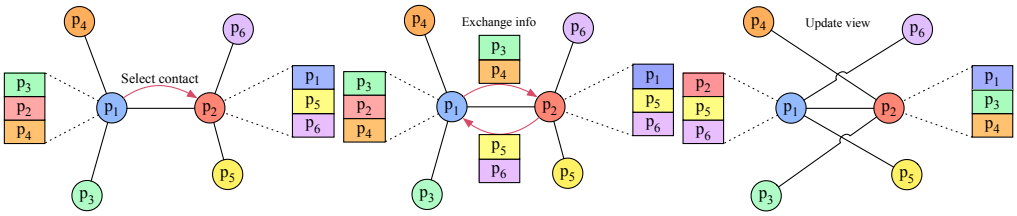
Les réseaux *P2P* avec topologie dynamique exploitent des protocoles de bavardage (*i.e. gossip*). En se calquant sur la propagation de rumeurs et les bavardages dans la vie réelle, ces protocoles sont caractérisés par des échanges réguliers – la régularité est définie par le système – entre pairs.

En plus de leur capacité de passage à l'échelle, ils sont faciles à implémenter, robustes, résistants aux pannes et permettent de s'adapter à la dynamicité du réseau. Les protocoles de *gossip* sont principalement utilisés pour (1) la dissémination d'information sur le réseau, (2) le calcul d'agrégations et (3) la détection d'erreurs. Cette caractéristique de dissémination de l'information fait que les protocoles de bavardage sont parfois appelés protocoles épidémiques. Par la suite, nous nous focalisons sur les protocoles de *gossip* pour les réseaux dynamiques, et seule la dissémination d'information nous intéresse donc : il s'agit de maintenir une vue dynamique du réseau et donc, de permettre à chaque pair u de découvrir d'autres pairs v présents sur le réseau.

Les protocoles de *gossip* utilisent une vue aléatoire du réseau – dont la taille est définie par le système – associée à chaque pair [75, 83]. Deux *threads* sont exécutés en parallèle : le premier, ou bavardage actif, exécuté par un pair u, initiera périodiquement des *bavardages* avec des pairs v choisis dans la vue aléatoirement de u ; le second, ou bavardage passif, exécuté par v, consistera à recevoir ces demandes de bavardage, émanant de u. Chaque pair se retrouve donc successivement actif et passif dans le protocole de *gossip*.

Un bavardage s'effectue en trois étapes, telles que décrites dans la Figure 2.7 :

1. **sélection du contact :** un pair p sélectionne aléatoirement un autre pair p_2 dans sa vue ;



(a) Sélection du contact. **(b)** Échange. **(c)** Mise à jour.

Figure 2.7: Échange entre deux pairs exécutant un protocole de *gossip*.

2. **échange :** p envoie un sous-ensemble aléatoire de cette vue à p_2, qui, en retour, envoie un sous-ensemble aléatoire de sa propre vue ;
3. **mise à jour :** enfin, p et p_2 mettent à jour leur vue respective en remplaçant aléatoirement des pairs qu'ils connaissaient par ceux qu'ils viennent de voir.

À terme, l'ensemble des utilisateurs (*e.g.* leur profil ou identifiant) sera diffusé sur le réseau – généralement en $\mathcal{O}(log(n))$.

2.2.3 Techniques de réplication

Étant donné un objet *it* appartenant à un pair p, la réplication est l'action de le dupliquer en le plaçant sur $1, ..., n$ pairs choisis. Elle est utilisée pour augmenter la disponibilité d'un contenu et donc pour réduire le coût nécessaire aux requêtes pour le retrouver.

Il est possible de déterminer deux niveaux de réplications [8, 44, 84, 124] :

1. **réplication passive :** il s'agit d'un processus naturel dans les réseaux *P2P*. Lorsqu'un pair soumet une requête, il va répliquer les contenus pertinents localement sur son nœud ;
2. **réplication active :** chaque pair du réseau va analyser le trafic et répliquer des contenus en conséquence (*e.g.* un contenu fréquemment recherché sera répliqué pour être rendu plus disponible).

Avec les réseaux structurés [8, 47, 124], les techniques de réplications peuvent tenir compte de l'identifiant des nœuds. Ainsi, une technique consiste à utiliser plusieurs fonctions de hachages afin que chaque objet soit associé à plusieurs *IDs*, et donc stocké sur plusieurs nœuds. Une autre solution est de stocker un objet sur les k nœuds dont l'identifiant est le plus proche, où suivant le nœud responsable de la clé donnée.

Dans les réseaux avec topologie non-structurée, ou dynamique, plusieurs autres possibilités de placement sont possibles [124] :

1. **aléatoire :** le choix du nœud responsable d'une copie est fait aléatoirement ;

2. **chemin de la requête :** les données R répondant à une requête q sont copiées sur l'ensemble des pairs se trouvant sur le chemin de q vers u, l'initiateur de la requête.

Cependant, une fois la règle de placement des données choisie, il est également important de déterminer le nombre de réplicas qui doivent être réalisés. Trois possibilités principales existent pour déterminer le nombre de copies d'un objet [44] :

1. **stratégie uniforme :** chaque objet est associé au même nombre de copies ;
2. **stratégie proportionnelle :** chaque fois qu'une requête q est exécutée, l'ensemble des objets y répondant sont répliqués ;
3. **stratégie de la racine carrée :** le nombre de copies d'un objet it est égal à la racine carrée de sa probabilité d'être accédée.

2.2.4 Bilan des systèmes de gestion de données distribués

Plusieurs types de réseaux *P2P* ont été introduits. Les réseaux avec topologie non-structurée, bien qu'offrant de nombreux avantages, sont limités en ce qui concerne le traitement des requêtes. Ces dernières, afin d'obtenir un *rappel* (*i.e.* fraction des objets pertinents que le système a pu retrouver dans le réseau) convenable, doivent recourir à des techniques comme le *flooding* : le réseau doit être inondé par la requête, ce qui entraîne un coût et donc un temps de réponse plus élevé.

À l'inverse, les réseaux structurés possèdent des algorithmes de traitement des requêtes beaucoup plus efficaces pour la recherche exacte. Ils ne s'adaptent cependant pas très bien aux requêtes complexes, comme celles à mots clés – les résultats pertinents pour chaque mot clé peuvent être stockés sur des nœuds différents –, nécessaires dans nos cas d'application.

Les recherches plus récentes se dirigent vers les réseaux dynamiques. En exploitant les protocoles épidémiques, ces réseaux possèdent les mêmes avantages que les réseaux avec topologie non-structurée, et peuvent traiter les requêtes plus efficacement. En effet, puisque des liens sont établis entre les pairs pertinents (*i.e.* étant donné un score) via le processus de *gossip* (*i.e.* chaque pair est maintenant relié à k autres pairs qu'il a choisi, étant donné un score), il n'est plus utile d'envoyer la requête à un très grand nombre de pairs pour obtenir des niveaux plus élevés de rappel. Cependant, ces systèmes entraînent un coût réseau supplémentaire dû aux échanges de *gossip* en tâche de fond.

La section suivante présente le déploiement des techniques de recherche et de recommandation, introduites en Section 2.1.3, sur une architecture distribuée.

2.3 Systèmes distribués de recherche et de recommandation

Cette section se propose d'introduire à la recherche d'information et à la recommandation dans les systèmes distribués. Sont tout d'abord présentés les systèmes *P2P* de recherche d'information en Section 2.3.1, suivis de la recommandation, toujours sur une architecture *P2P*, présentée en Section 2.3.2. Enfin, nous discutons des objectifs et des enjeux de la recherche d'information sur une architecture *multisite*, en Section 2.3.3.

2.3.1 Systèmes *P2P* pour la recherche d'information

Dans cette section, nous présentons les techniques mises en place dans le cas particulier de la recherche d'information sur une architecture *P2P*. En effet, dans nos cas d'application, les données ne sont pas disponibles sur une plateforme centralisée et nécessitent la mise en place de protocoles distribués. Ces techniques peuvent être classées en deux catégories principales : les solutions basées sur le *clustering*, qui sont présentées en Section 2.3.1, et celles basées sur les raccourcis, qui sont présentées en Section 2.3.1.

Recouvrement basé sur la classification ou *Clustering Overlay*

Deux solutions principales existent quant aux approches basées sur la classification :

- **Regroupement par profil de pairs [24, 72, 76, 86, 159] :** chaque pair stocke et partage ses propres données localement. Les pairs ayant un profil proche sont regroupés (*i.e. cluster*) [24, 72, 86, 76]. Le réseau est donc composé d'un ensemble de $1, ..., m$ *clusters*. Lorsqu'un utilisateur u soumet une requête q, celle-ci est transmise au *cluster* ayant la plus forte probabilité de posséder des résultats pertinents [46]. Elle est ensuite *inondée* au sein du *cluster* (*e.g. BFS*, cf. Section 2.2.2).
- **Regroupement par données [26, 97, 140, 159, 161] :** ces solutions se focalisent sur le regroupement des données. Les données peuvent par exemple être regroupées sur certains pairs en fonction de leur localité (*e.g. locality-based*) [26, 140]. Généralement, ces solutions utilisent des réseaux structurés (cf. Section 2.2.2). De fait, si un objet *it* répond à une requête q, il y a une forte probabilité pour que d'autres objets à proximité dans le réseau soient donc sémantiquement proches de q.

Recouvrements avec raccourcis, ou *Shortcut Link Overlays*

Alors que les approches précédentes adoptent une vision globale – chaque pair ou donnée étant associé à un unique *cluster* prédéfini –, ici, chaque pair u définit son propre recouvrement de manière à augmenter la probabilité que son entourage puisse répondre aux requêtes soumises par u. Ces solutions utilisent généralement les réseaux dynamiques, et donc, les protocoles de *gossip*. Comme cela a été présenté précédemment, chaque pair est alors associé à une vue aléatoire [75, 83] contenant des pairs disponibles sur le réseau, choisis aléatoirement. Afin de construire un voisinage pertinent, chaque pair u sélectionne les pairs les plus pertinents (*e.g.* en fonction d'un score, de l'amitié) rencontrés dans la vue aléatoire : des raccourcis sont ainsi établis entre le pair courant u, et ceux qui lui sont le plus pertinents, et traiteront les requêtes que u soumettra. Plusieurs approches existent :

Basés sur les intérêts [33, 42, 71, 155, 168] : ces solutions utilisent des techniques de filtrage collaboratif (cf. Section 2.1.2) : étant donné deux utilisateurs u et v, si u et v ont des contenus en commun, il y a une chance que les autres contenus, partagés uniquement par l'un des deux puissent intéresser l'autre. En plus d'un recouvrement de base (cf. Section 2.2), chaque pair, ou utilisateur, maintient un raccourci vers ceux ayant le profil le plus proche. Ainsi, lorsqu'une requête q est soumise, les raccourcis empruntés par q permettent de trouver des résultats pertinents tout en réduisant le nombre de sauts (*i.e. TTL*). Cette solution est utilisée par *Gnutella* [42]. Afin d'augmenter les performances de la recherche, des raccourcis peuvent être établis entre chaque utilisateur u et les autres utilisateurs qui lui ont précédemment retourné des résultats pertinents [155] : si ces pairs ont répondu à des requêtes de u par le passé, il y a de fortes chances qu'ils y répondent encore dans le futur.

P2PRec [49] extrait des thèmes (*i.e. topics*) associés à chaque document qu'il partage, afin de calculer le profil de chaque utilisateur, noté *profile(u)*. Finalement, en plus d'un recouvrement aléatoire, chaque pair maintient un recouvrement *sémantique* en sélectionnant les k pairs qui lui sont le plus similaires. Lorsqu'une requête est soumise par u, elle est transmise en parallèle aux utilisateurs présents dans le recouvrement aléatoire et sémantique.

P3Q et *P4Q* [21] exploitent les étiquettes (*i.e.* tags) que chaque utilisateur peut attribuer aux objets qu'il partage. Lorsqu'un utilisateur u soumet une requête à mots clés q, l'objectif est d'appliquer un *top-k* personnalisé. Chaque utilisateur u est représenté par un profil noté *profile(u)*, associant chaque objet partagé par u ainsi que leurs étiquettes. À cela s'ajoute le réseau de n utilisateurs connus par u, noté *network(u)*. Celui-ci est composé de deux parties : la première, notée *profileList(u)* et de taille c ($c << n$), regroupe les utilisateurs les plus similaires à u, chacun étant associé à un *filtre de bloom* indiquant les étiquettes attribuées par l'utilisateur ainsi que son profil. La seconde, notée *bloomfilterList(u)* et de

taille $n - c$, regroupe les utilisateurs moins similaires. Seul le *filtre de Bloom* de ces utilisateurs est conservé. La similarité entre deux utilisateurs est calculée à partir des étiquettes qu'ils ont soumises.

Finalement, lorsque u soumet une requête à mots clés q, celle-ci est exécutée localement en utilisant les profils des utilisateurs proches. Si aucun résultat n'est trouvé ou que les résultats ne conviennent pas à u, un protocole de *gossip* est exécuté pour la traiter :

1. u choisit les utilisateurs *bloomfilterList(u)* dont le filtre contient les étiquettes de la requête. Ces utilisateurs sont ajoutés à la liste *remainingList(u)* ;
2. la requête et la liste sont ensuite envoyées à un utilisateur v choisi aléatoirement dans la liste ;
3. lorsqu'un utilisateur reçoit une requête et une liste, il regarde en premier s'il possède localement les profils d'utilisateurs dans la liste et traite la requête le cas échéant. Enfin, il renvoie les résultats et une partie de la liste à u. À ce niveau, la partie renvoyée à u sera traitée par u tandis que l'autre le sera par v.

Tveit [166] propose de caractériser chaque pair par son vecteur de notes (*i.e.* les notes qu'il a attribuées à des objets). Ce vecteur est ensuite échangé sur le réseau afin de regrouper ensemble les pairs les plus similaires. Cela permet de réduire le nombre de sauts de chaque requête sur le réseau tout autant que d'augmenter la qualité des recommandations.

Basés sur les relations sociales [32, 41, 55, 109, 131, 167] : ces approches utilisent les données sociales de chaque utilisateur (*e.g.* amis, étiquettes ou *tags*) pour construire le recouvrement et rechercher les contenus.

SPROUT [109] s'appuie sur une *DHT* en anneau (*i.e. Chord*) (cf. Section 2.2.2) et rajoute des raccourcis à chaque pair en s'appuyant sur les relations explicites d'amis ; ces amis sont choisis soit pour des raisons de confiance, soit pour des raisons de performances. L'algorithme de routage de la requête fonctionne de la même manière que précédemment sauf qu'il va choisir en priorité dans la table de raccourcis plutôt que dans celle de routage. Ainsi, lorsqu'une requête q avec la clé k est soumise par un utilisateur u, celle-ci va être transmise à *l'ami* de u ayant un identifiant le plus grand possible, mais plus petit que k, jusqu'à ce que le pair responsable de k soit trouvé. Si un tel utilisateur n'existe pas dans la table de raccourcis de u le routage (*i.e. lookup*) normal est utilisé.

Tribler [131] combine deux aspects pour construire le recouvrement. Premièrement, les utilisateurs ayant explicitement établi une amitié sont connectés entre eux. Dans un second temps, grâce à des protocoles de *gossip*, les utilisateurs les plus similaires en termes de fichiers partagés sont également reliés.

Cependant, Kermarrec et Taïani [82] montrent que le rappel dans ces solutions reste faible. Leur expérience exploite des données issues de Twitter et consiste à

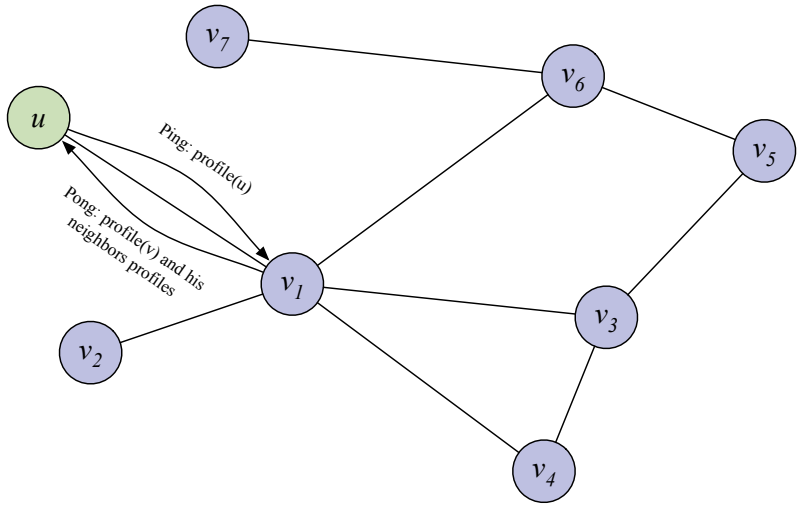


Figure 2.8: Construction de la matrice de filtrage collaboratif au sein d'un réseau non structuré.

relier les pairs similaires en termes d'abonnement Twitter afin de leur en recommander d'autres. En utilisant une mesure commune pour créer le recouvrement, un certain nombre de recommandations ne sont pas faites car trop éloignées sur le réseau. Les auteurs soulignent que, si chaque pair utilise sa propre métrique de similarité, le rappel peut être augmenté.

2.3.2 Systèmes *P2P* de prédiction

Nous introduisons l'application de techniques de recommandation (cf. Section 2.1.3) sur une architecture *P2P*. Celles-ci se décomposent principalement en techniques de filtrage collaboratif, présentées dans un premier temps, suivies de celles de filtrage social, décrites dans un second temps.

Filtrage collaboratif

À l'opposé des méthodes de recommandation centralisées, ici chaque utilisateur n'est initialement associé qu'à ses propres notes et objets. L'objectif des méthodes suivantes consiste à construire des prédictions en *P2P*.

PocketLens [115] exploite des techniques de filtrage collaboratif afin de découvrir des pairs sur le réseau :

1. **Aléatoire (cf. Figure 2.8) :** lorsqu'un pair u rejoint le réseau, il envoie un message *ping* à un utilisateur v avec son profil. En retour, v renvoie un message *pong* à u, incluant son profil et ceux des voisins de v.
2. **Traversée transitive :** l'utilisateur u inonde une requête q, et prend en

compte tous les pairs rencontrés sur le chemin de q. Seuls les plus similaires sont conservés.

3. **DHT :** chaque objet est stocké sur un nœud donné. Les données similaires (*i.e.* en termes de filtrage collaboratif) sont stockées sur les mêmes nœuds ou des nœuds proches. Finalement, chaque utilisateur u procède à une agrégation pour calculer les prédictions en fonction des objets similaires à la manière d'un filtrage collaboratif.

En agrégeant les notes appartenant aux pairs rencontrés sur le réseau, le pair courant u est donc capable de calculer des prédictions.

PipeCF [67] stocke également les données et les notes au sein d'une *DHT*. Tous les utilisateurs ayant évalué un objet *it* avec la même note sont regroupés au sein de paniers (*i.e. bucket*). Finalement, chaque utilisateur u calcule ses prédictions en récupérant les paniers de tous ses objets.

Kermarrec *et al.* [81] proposent de construire la matrice en deux étapes. La première consiste en l'exécution d'un protocole de bavardage afin de regrouper les utilisateurs similaires. La seconde se propose d'exécuter des marches aléatoires afin de récupérer les notes données par des utilisateurs afin de remplir la matrice.

Filtrage social

Il s'agit d'une sous-catégorie du filtrage collaboratif. Les prédictions combinent les notes des utilisateurs à leurs données sociales pour être calculées. Massa et Avesani [110] proposent ainsi une plateforme *P2P* de filtrage collaboratif basée sur le confiance entre utilisateurs (*i.e. trust-aware collaborative filtering*). La confiance entre deux utilisateurs, notée *trust(u,v)*, est une valeur entre 0 et 1 (cf. Figure 2.4). Cette information peut être modelée comme un graphe $\mathcal{G} = (U, E)$, U représentant les utilisateurs et E les liens dirigées du graphe entre deux utilisateurs u et v, chaque arête existant si un niveau de confiance à été calculé par le système de u envers v. Le niveau de confiance entre deux utilisateurs ne l'ayant pas exprimée se calcule de la manière suivante :

$$trust(u, v) = \frac{d - n + 1}{d} \tag{2.20}$$

Où d est la distance maximale existant dans le réseau ; c'est une valeur définie par le système. n est la distance minimale entre u et v. Ainsi, dans la Figure 2.4, la confiance entre u_3 et u_2, avec $d = 4$ a une valeur de $(4 - 3 + 1)/4 = 0,5$.

2.3.3 Systèmes *multisites* de recherche d'information

Toujours dans un contexte distribué, les systèmes multisites peuvent être définis par leur opposition au *P2P* [20, 28, 35, 80] : alors que pour le *P2P*, chaque nœud représente un unique pair, dans une approche multisite, ils peuvent représenter un ensemble d'utilisateurs ; le nombre de sites est donc très réduit comparé

au nombre de nœuds d'un réseau *P2P*. De plus, la probabilité que de nouveaux sites rejoignent le réseau ou que l'un d'entre eux le quitte – de manière volontaire ou suite à une panne – est faible. Ces approches tentent de combiner certains avantages du *P2P* (*e.g.* réduction des coûts, pas de point d'accès unique) avec ceux des infrastructures centralisées (*e.g.* pérennité des données, stabilité). Chaque site est composé d'un ensemble d'utilisateurs qui soumettent leurs requêtes. Étant donné l'ensemble des objets I, chaque site s_i indexe un sous-ensemble de I, noté I_{s_i} La principale problématique consiste à réduire le coût réseau du traitement des requêtes, ou, en d'autres termes, de maximiser le nombre de requêtes qui peuvent être calculées localement. Cette problématique conduit à trois sous-problèmes :

1. **Transmission de la requête [19, 35, 58] :** dans quels cas doit-on transmettre ou non une requête q d'un site s_1 à un site s_2 ? Si les *top-k* documents permettant de répondre à une requête sont stockés et indexés sur le site s_1, alors il n'est pas utile d'envoyer la requête au site s_2. Étant donné une requête q, il est possible de définir un seuil (*i.e. threshold*) de la manière suivante :

$$threshold(q, s_i, s_j) = \max score(q, I_{s_j} \backslash I_{s_i}) \tag{2.21}$$

   Ainsi, si un utilisateur $u \in s_1$ soumet une requête q au site s_1, celle-ci est traitée de la manière suivante. Le site s_1 va tout d'abord calculer un $top - k$ local, noté $R^q_{s_1}$. Puis, le site s_1 va comparer le score des objets de $R^q_{s_1}$ avec les seuils des autres sites s_i :

$$\exists it \in R^q_{s_1}, score(it) < threshold(q, s_1, s_i) \tag{2.22}$$

   Si la condition est satisfaite, alors le site s_i possède potentiellement des résultats qui pourraient être insérés dans $R^q_{s_1}$: la requête est alors propagée au site s_i.

   Des techniques de *machine learning* peuvent être mises en œuvre pour réduire encore plus la quantité de requêtes transmises entre les sites [58].

2. **Assignation d'un document à un site [28] :** puisqu'un objet donné ne doit pas être indexé par tous les sites, il doit se voir attribuer un *maître* (*i.e. master*) afin de toujours rester disponible si un site décide de le supprimer. L'objectif devient donc de définir un score d'affinité pour chaque site en fonction d'un objet : $affinity(it, s_i)$. Une manière simple de définir ce score est d'évaluer la langue utilisée dans le contenu de l'objet et celles que les utilisateurs du site utilisent. Le maître sera le site où les utilisateurs parlant la langue de l'objet seront les plus nombreux. Cette méthode reste limitée puisque, par exemple, les contenus en anglais intéressent beaucoup plus d'individus que la seule communauté anglophone. Une manière plus pertinente consiste à exploiter les documents déjà présents sur chaque site et les requêtes qui leur ont récemment été soumises afin d'évaluer la probabilité que le nouveau document intéresse les utilisateurs du site s_i. En

d'autres termes, on s'intéresse à la probabilité que ce nouvel objet corresponde aux contenus déjà sur le site et répondant aux requêtes soumises par les utilisateurs de ce dernier.

3. **Réplication entre sites [80] :** l'objectif de la réplication ici est de maximiser le nombre de requêtes qui peuvent être traitées localement dans le site d'appartenance des utilisateurs les soumettant. La réplication doit se soumettre à deux contraintes : le nombre de réplicas d'un objet, au niveau global, ne doit pas être supérieur à un seuil prédéfini ; le nombre de réplicas présents sur un site ne doit pas excéder un seuil local prédéfini.

Bilan des systèmes distribués de recherche et de recommandation

Trois points ont été évoqués. Le premier aborde les systèmes de recherche d'information en *P2P*. Ils permettent aux utilisateurs de soumettre des requêtes à mots clés et d'en récupérer les résultats les plus pertinents. L'objectif principal de ces solutions est de réduire le coût de traitement des requêtes (*i.e.* en temps et en coût réseau, les deux points étant intimement liés), sans détériorer la qualité des résultats. Pour cela, deux catégories d'approches existent :

1. celles à base de classification qui grouperont soit les données, soit les pairs, en fonction d'une mesure de similarité ;
2. celles à base de raccourcis, qui, étant donné un pair u établiront des raccourcis entre u et les pairs qui lui sont le plus pertinents (*e.g.* mesure de similarité, amitié).

Le second point concerne les systèmes de prédiction *P2P*. Ces derniers retournent de manière pro-active un ensemble de recommandations aux utilisateurs, étant donné leur profil – on peut, à ce stade, faire un parallèle avec les techniques *top-N* précédemment évoquées. L'objectif est de déterminer la manière dont les informations de la matrice « *Users* × *Items* » S doivent être agrégées afin de calculer les prédictions, et cela, sans détériorer la capacité de passage à l'échelle et les performances du système. Cependant, à l'inverse du premier point, l'utilisateur ne peut pas exprimer ses besoins au travers de requêtes. Enfin, le multisite est une architecture intermédiaire entre les systèmes centralisés et *P2P*. Chaque nœud regroupe un grand nombre d'utilisateurs ; la quantité des sites est également réduite et leur fiabilité forte (*i.e.* ils ont une faible probabilité de se déconnecter). De nouvelles techniques sont ainsi proposées afin d'obtenir des temps de traitement des requêtes quasiment équivalents à ceux obtenus dans un cadre entièrement centralisé.

2.4 Conclusion et Discussion

Dans cette section, nous présentons une synthèse des techniques présentées au sein de ce chapitre, ainsi qu'une discussion à propos de leur utilisation ou de

leurs limites dans la recherche et la recommandation diversifiées et distribuées, au sein de cette thèse.

Tout d'abord, rappelons que des schémas récapitulatifs sont présents pour résumer les méthodes présentées :

1. **recherche et recommandation** : Figure 2.9 en page 56 ;
2. **diversification et sérendipité** : Figure 2.10 en page 57 ;
3. **recherche et recommandation dans les systèmes *P2P*** : Figure 2.11 en page 58.

Le contexte de ce travail se situe à la croisée de plusieurs domaines qui sont la recherche d'information et la recommandation imbriquées au sein d'architectures *P2P* et multisites. Nous évoquons les problématiques abordées dans ce travail de la manière suivante : le modèle de recherche et recommandation puis la distribution des données.

2.4.1 Modèle de recherche et recommandation

La quantité phénoménale de données issues du phénotypage tout autant que de la botanique oblige à réaliser des plateformes de recherche et de recommandation afin de retrouver les contenus les plus intéressants pour les utilisateurs. L'objectif est donc, à partir d'une requête à mots clés q soumise par un utilisateur u, de recommander les meilleurs résultats ; ce qui se définit en deux points.

En premier lieu, un résultat doit être pertinent par rapport à q mais également par rapport au profil de u : en phénotypage, supposons que u soumette la requête $q = \textit{modèle}$, le profil de u permet, par exemple, de savoir qu'un résultat pertinent parlera d'un modèle de plante et non d'un modèle mathématique ou informatique. Dans le cadre de la botanique, cette personnalisation permet de demander aux utilisateurs d'identifier des plantes qu'ils ont une forte probabilité de connaître.

On utilise, tout d'abord, le *filtrage collaboratif*. Ce dernier permet de trouver, par l'application de modèles ou le calcul d'un voisinage, des corrélations et des motifs dans le comportement des utilisateurs qui sont ensuite exploités afin de prédire des scores pour les données de la plateforme. Ces systèmes ont tendance à souffrir du problème de démarrage à froid, c'est-à-dire que les utilisateurs, n'ayant jamais interagi avec une donnée ou les données n'étant reliées à aucun utilisateur, elles ne peuvent pas se faire recommander ; à cela s'ajoute la possibilité pour les utilisateurs d'interagir avec des données très similaires mais différentes (*e.g.* deux observations de la même plante). Le calcul d'un voisinage ou la détection de motifs devient ici, également, très difficile.

Le *filtrage basé sur les contenus* a pour but de résoudre, ou du moins, de limiter cet effet. Ces techniques, en exploitant le contenu de chaque objet, détecte les similarités, motifs et corrélations qui existent entre eux, et effectue des prédictions. La principale limitation est, par définition, la très grande similarité

qu'il y aura entre les recommandations et les objets partagés par l'utilisateur en question, réduisant donc l'effet de découverte des contenus.

Le *filtrage hybride* essaye de combiner ces deux précédentes méthodes afin d'exploiter la qualité des recommandations de la première avec la flexibilité en termes de calcul de corrélation, voisinage, etc. de la seconde. Dans notre contexte, cette combinaison peut se faire en calculant des profils basés sur les contenus pour chaque utilisateur, et en exploitant des techniques collaboratives dans le processus de recommandation.

Le *filtrage social* est une sous-catégorie du filtrage collaboratif qui utilise les réseaux sociaux et toutes les plateformes de ce type afin d'enrichir les profils et donc de combler l'effet du démarrage à froid.

Une fois le modèle de recommandation défini, des techniques comme les listes inversées et les algorithmes de *top-k* permettent à l'utilisateur de soumettre des requêtes dans lesquelles ils expriment leur souhait (*i.e.* les caractéristiques des données qu'ils recherchent). Malheureusement, ces modèles ont tendance à être très redondants dans leurs prédictions : seuls les objets très similaires entre eux se voient attribuer de bonnes prédictions. La *diversification* et la *sérendipité* sont là pour limiter cet effet. En diversification, des distances entre objets sont calculées, et si une recommandation a est faite, alors la seconde recommandation b doit être distante de a. En sérendipité, cette distance est introduite avec les objets déjà associés avec le profil utilisateur.

Comme on a pu l'observer en Section 2.1.4, les méthodes de diversification peuvent parfois être limitées : le contenu n'est pas suffisant pour établir une distance correcte entre deux objets. En particulier, nous mettons en avant deux aspects :

1. **Mauvaise description du contenu :** la mauvaise qualité de description (*i.e.* description erronée ou trop succincte) des contenus réduit l'effet de la diversification [176]. Deux jeux de données peuvent par exemple être respectivement étiquetés avec « Croissance de la feuille en fonction de l'apport en eau » et « Évolution de la taille des feuilles sous un stress hydrique ». Ces jeux de données seront considérés par le système comme très divers alors qu'ils font, en réalité, référence à une même chose.
2. **Ambiguïté sémantique :** un mot particulier, utilisé dans le processus de recherche, peut faire référence à plusieurs sémantiques. Par exemple, la requête *diversité* peut correspondre à la *bio-diversité*, *la diversité en recherche d'information* et bien plus encore. Ainsi, avec la diversité des contenus, l'ensemble des recommandations peut toujours faire référence à la même sémantique (*e.g.* un ensemble divers de documents à propos de la bio-diversité). De manière plus générale, en science, un même mot peut être utilisé par plusieurs communautés scientifiques avec de légères différences sur leurs manières d'aborder les problèmes. L'utilisation d'ontologies permet de résoudre ce problème dans certains scénarios, mais leur déploiement n'est

pas toujours évident. Par exemple, dans le contexte du phénotypage, les requêtes soumises par les utilisateurs correspondaient rarement aux concepts d'ontologies de biologie, et deux objets pouvaient être associés aux mêmes concepts, alors même qu'ils étaient relativement différents.

Pour cela, diverses propositions ont été faites, comme recommander des résultats d'une manière telle que les utilisateurs associés à deux d'entre eux ne soient pas les mêmes. Ces solutions restent limitées (*e.g.* deux jeux de données provenant de la même équipe de recherche mais partagés par deux utilisateurs différents deviendraient très divers).

2.4.2 Distribution des données

Approches *P2P*

Les données de chaque botaniste et biologiste sont produites de manière distribuée (*i.e.* localement à leur machine, sur un serveur, sur le *cloud*). La seconde catégorie de problèmes abordés dans cette thèse fait donc référence à la distribution des données. En effet, les botanistes et les biologistes produisent leurs données de manière distribuée (*e.g.* sur leur ordinateur personnel) ; les algorithmes de recherche et recommandation centralisés ne peuvent plus s'appliquer et doivent être déployés sur une architecture distribuée.

Les systèmes de prédiction *P2P* retournent un ensemble de recommandations en fonction du profil de chaque utilisateur – de la même manière que leurs homologues centralisés. Ces solutions proposent différentes méthodes afin de distribuer la matrice *Users* × *Items* sur le réseau. Le coût d'agrégation de celle-ci limite le passage à l'échelle et la qualité des prédictions du système. Qui plus est, contrairement à ce qui est souhaité dans notre scénario, les utilisateurs ne peuvent pas soumettre de requêtes.

Les systèmes *P2P* de gestion de contenu – ou de recherche d'information – utilisent différentes techniques comme le *clustering* pour faciliter la recherche. Deux approches ont été présentées :

1. **Basées sur la classification :** les pairs similaires ou les données similaires sont regroupés au sein de *clusters*. La recherche s'effectue ainsi en retrouvant d'abord le *cluster* pertinent et en inondant la requête au sein de celui-ci, ce qui accroît le trafic réseau. Dans le cadre des réseaux structurés, ces derniers sont utilisés afin de retrouver les *clusters* pertinents en charge des données répondant à la requête. Le coût réseau de maintien d'une telle infrastructure reste élevé.
2. **Basées sur les raccourcis :** ces solutions établissent des liens logiques entre utilisateurs aux intérêts similaires. Ces derniers sont calculés soit via les données partagées par chaque pair, soit en exploitant des relations sociales. L'inondation est encore une fois la méthode utilisée pour propager la

requête. Cependant, plus les liens d'intérêts seront bien définis, plus le *TTL* (*i.e.* distance que la requête va parcourir sur le réseau) pourra être minimisé, réduisant le temps de réponse. Avec les réseaux dynamiques, chaque pair peut, en toute autonomie, réaliser ses propres liens d'intérêts. Ces échanges périodiques entraînent cependant une légère augmentation du trafic réseau de fond (*i.e.* nombre de messages échangés sur le réseau).

En regroupant les pairs via diverses méthodes de similarités, les recommandations et les résultats aux requêtes soumises par les utilisateurs proviennent de pairs très similaires. Ainsi, la probabilité que chacun retourne le même ensemble de résultats s'en retrouve accrue et limite, par définition, les niveaux de rappel. Rappelons que le rappel représente la fraction des objets pertinents, étant donné une requête q, que le système a pu récupérer sur le réseau, en respectant les contraintes comme le *TTL*. Les solutions de l'état de l'art se trouvent donc dans l'obligation de parcourir un plus grand nombre de nœuds sur le réseau augmentant ainsi les niveaux de rappels mais également les temps de réponse.

Approches multisites

Comme cela a été présenté, chaque botaniste ou biologiste produit ses données de manière distribuée. Cependant, les sites ne représentent pas nécessairement un unique utilisateur (*e.g.* serveur d'équipe, site dans le *cloud*) et les techniques *P2P* se retrouvent limitées. Les approches multisites sont une manière d'aborder ce problème. Cependant, la grande hétérogénéité des infrastructures qui seront présentes sur le réseau ne permet pas l'application directe des solutions de l'état de l'art. La staticité de ces propositions n'est pas acceptable dans notre scénario, puisque le réseau est à la fois composé de nœuds de type ordinateur personnel, de serveurs d'équipe ou de sites dans le *cloud* par exemple.

2.4.3 Conclusion

Nous avons introduit l'ensemble des techniques importantes à la compréhension de cette thèse.

Nous avons mis en avant un certain nombre de limitations et donc de directions de recherche qui ont été l'axe directeur de ce travail. Plus précisément, nous avons indiqué que les techniques de diversifications restaient limitées dans nos deux cas d'utilisation. Dans les Chapitres 3 et 4, nous présentons la diversité des profils et les algorithmes correspondants permettant d'aborder ces limitations. Les limites de l'approche *P2P* en termes de rappel et de performance sont également exposées. Le Chapitre 5 présente nos contributions à ce propos. Enfin, nous avons montré que les récentes contributions dans le domaine du *multisite* ne permettaient pas, en tant que telles, de répondre à nos objectifs. Le Chapitre 6 décrit notre approche multisite, accompagnée d'un prototype composé de deux versions répondants à nos cas d'application.

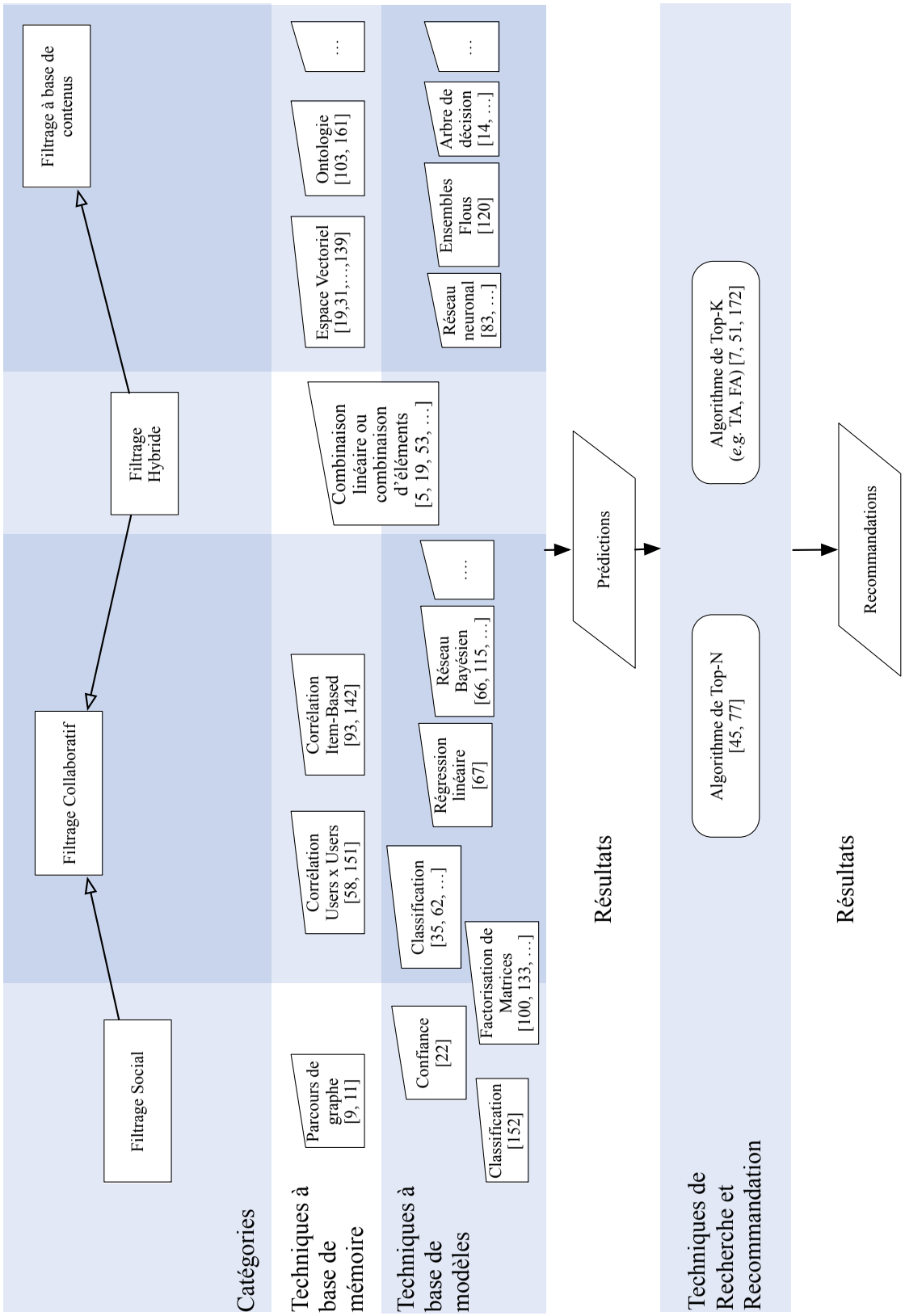


Figure 2.9: Synthèse non-exhaustive des méthodes de recherche et recommandation (pour plus de citations se référer aux sections correspondantes).

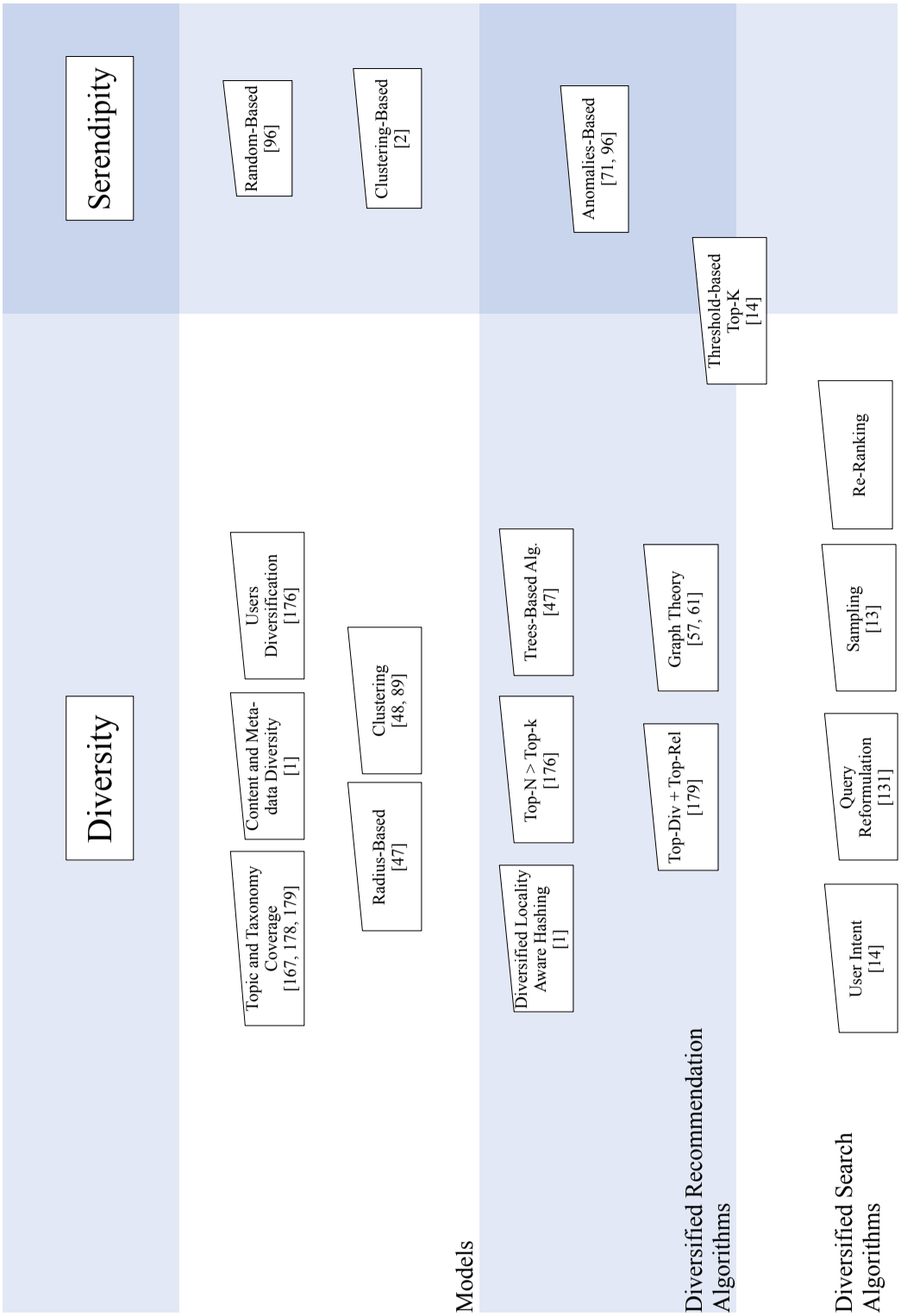


Figure 2.10: Synthèse non-exhaustive des méthodes de diversification et de sérendipité (pour plus de citations se référer aux sections correspondantes).

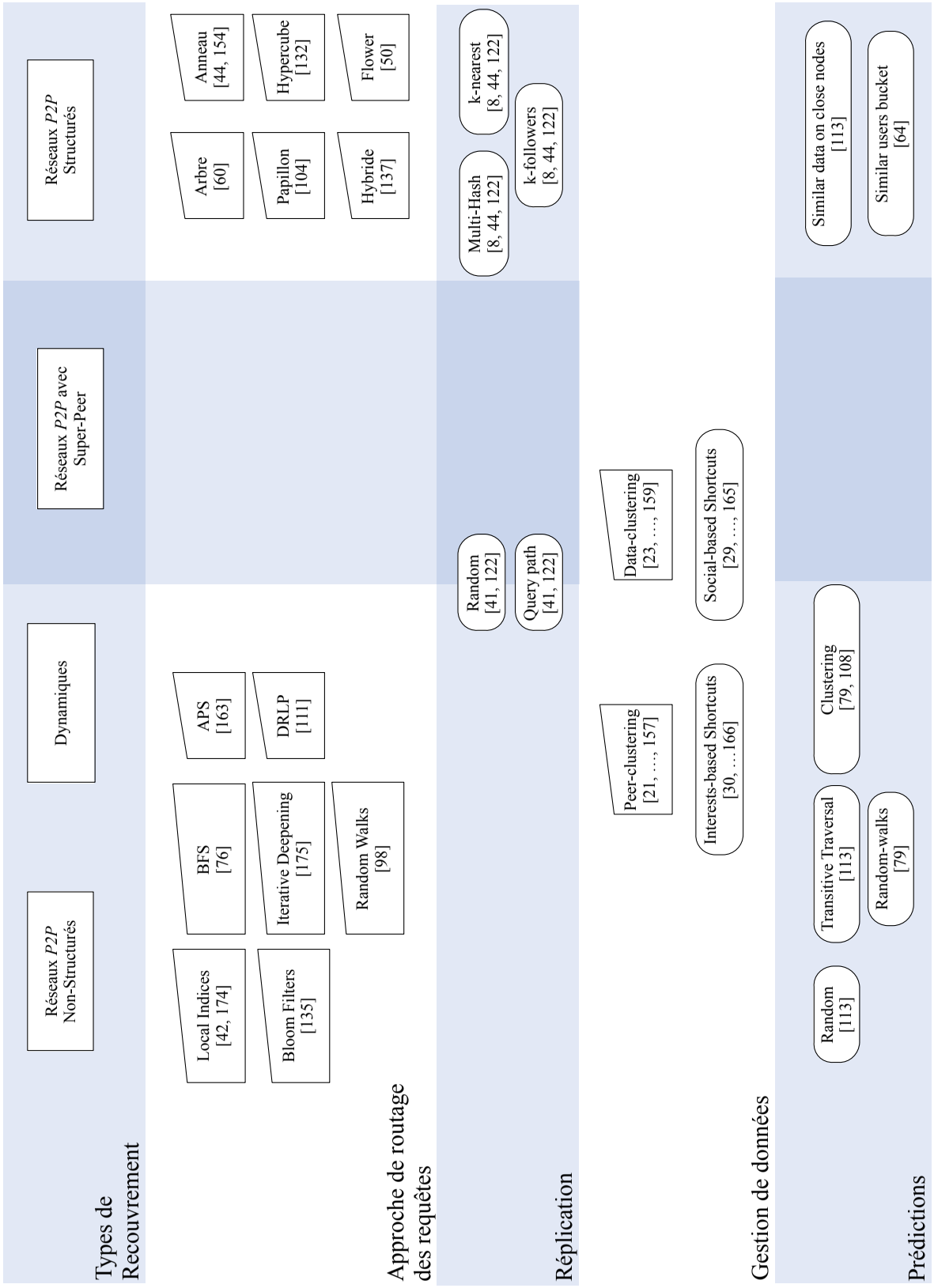


Figure 2.11: Synthèse non-exhaustive des méthodes et techniques *P2P* (pour plus de citations se référer aux sections correspondantes).

CHAPITRE 3

Diversification des profils pour la recherche et la recommandation

Résumé. Ce chapitre présente une nouvelle approche de la diversité où les profils des utilisateurs sont exploités ; l'objectif est d'améliorer la qualité de la diversification en limitant les effets dus à une mauvaise description des contenus ainsi que de l'ambiguïté sémantique entre certains termes et mots clés. Nous présentons notamment notre modèle de score ainsi que les algorithmes nécessaires au calcul de résultats diversifiés. Afin d'évaluer nos propositions, nous avons réalisé un ensemble d'expériences, incluant notamment un sondage, sur trois jeux de données : dans plus de 75% des cas, la diversité des profils a été préférée par les utilisateurs en comparaison avec les autres approches.

3.1 Introduction

Comme cela a été présenté au Chapitre 1, de nombreux utilisateurs sont de grands producteurs de données, qu'il s'agisse d'observations de botanique, de données de phénotypage, d'images, de vidéos ou encore de signets (*i.e.* lien internet) ; ils peuvent exprimer leurs opinions en associant à celles-ci des commentaires ou des étiquettes (*i.e.* petit groupe de mots clés ou *tags*). Malheureusement, cette incroyable quantité de contenus complique la récupération de données pertinentes. En botanique, comme cela a été montré dans le Tableau 2.1 en page 14, si la diversité n'est pas prise en compte, les utilisateurs reçoivent uniquement des observations de plantes identiques. De la même manière, en 2009, après avoir analysé 10 millions de requêtes sur *Yahoo ! Travel*, Yu *et al.* [176] ont montré que plus de 90% d'entre elles étaient génériques, c'est-à-dire qu'elles ne contenaient que très peu de mots clés comme, par exemple, « voyage en famille ». Elles sont tellement vagues qu'il n'est pas possible au système de retourner des résultats pertinents

aux utilisateurs. À cette fin, de nombreuses plateformes telles que *Amazon*[1] ou *Netflix*[2] utilisent des stratégies de recommandation afin de filtrer les résultats en fonction du profil de chacun, permettant ainsi d'accroître la satisfaction de ces derniers.

Cependant, bien que ces stratégies de recommandations permettent de filtrer considérablement le corpus global, elles tendent également à effectuer des recommandations redondantes ou sur-spécialisées, comme cela a été présenté au Chapitre 2. Cette redondance peut devenir très ennuyante et ainsi entraîner une baisse de l'utilisation de la plateforme [176]. Par exemple, sur *Delicious*, en mars 2014, l'étiquette `#tunisia` ne permettait de récupérer que des concernant les difficultés post-révolution du pays. De même, en 2008, lors de la primaire démocrate, l'étiquette `#election` ne renvoyait que des liens à propos de *Barack Obama* [176]. La diversité des contenus a été étudiée afin d'aborder ce problème et de nombreux résultats positifs ont été obtenus [2, 176, 179].

Malheureusement, cette diversité peut être limitée dans certains cas. Le Tableau 3.1 présente quatre cas extraits de nos expériences utilisant un jeu de données provenant de *Flickr* – le Tableau 3.2 présente un exemple proche pour les données de phénotypage, et le Tableau 6.2 en page 135 s'appuie sur des données de botanique. Deux utilisateurs ont soumis la requête « lion ». Les cas 1 et 2 montrent les recommandations faites pour chacun des deux utilisateurs sans et avec la diversité des contenus. Les cas 3 et 4 montrent celles faites pour les deux mêmes utilisateurs en tenant compte de la diversité des profils. Dans le premier cas, seules des images de lion dans des contextes très similaires sont renvoyées. Dans le second, l'ensemble est plus divers mais reste malgré tout très similaire à une solution non diversifiée. Deux raisons peuvent être mises en avant pour expliquer ce phénomène :

1. **Mauvaise description du contenu :** sur de nombreuses plateformes comme *Delicious*, la mauvaise qualité de description des contenus réduit l'effet de la diversification [176]. Deux images de la tour Eiffel peuvent respectivement être étiquetées avec « Paris vacances voyage » et « Paris tour Eiffel ». Elles seront considérées par le système comme très diverses alors qu'elles sont en réalité très redondantes. De même, en phénotypage, deux jeux de données peuvent respectivement être décrits comme « Croissance de la feuille en fonction de l'apport en eau » et « Évolution de la taille des feuilles sous un stress hydrique ». Ils seront ainsi considérés par le système comme très divers alors qu'ils font, en réalité, référence à une même chose.
2. **Ambiguïté sémantique :** un mot particulier, utilisé dans le processus de recherche, peut faire référence à plusieurs sémantiques. C'est par exemple le cas avec la requête `diversité` qui peut renvoyer à la *bio-diversité*, *la diversité en recherche d'information* et bien plus encore. Ainsi, avec la diversité

1. `www.amazon.com`
2. `www.netflix.com`

des contenus, l'ensemble des recommandations peut toujours faire référence à la même sémantique (*e.g.* un ensemble divers de documents à propos de la bio-diversité). De manière plus générale, en science, un même mot peut être utilisé par plusieurs communautés scientifiques dans des contextes très différents : le phénotypage étudie la plante dans sa totalité alors que la génétique étudie les plantes en termes de gênes.

Avec la diversité des profils, le système de recommandation prend en compte des objets (*i.e. items*) provenant d'utilisateurs aux profils divers. Cela permet de limiter l'effet des contenus ayant une description trop succincte et d'augmenter la probabilité que l'ensemble des objets recommandés corresponde bien à plusieurs thématiques (*i.e.* sémantique).

Afin d'illustrer les bénéfices de cette méthode, nous allons brièvement discuter des cas d'utilisations présentés dans le Tableau 6.2. Remarquons que pour les cas 3 et 4, en plus d'être pertinents avec la requête « lion » et avec le profil de chaque utilisateur (*i.e.* recommandation), les résultats prennent également en compte la diversité des contenus et des profils des utilisateurs ayant partagé les images ; cela permet d'obtenir une bonne qualité de diversification et donc des recommandations. Alors même que les résultats sont beaucoup plus ciblés sur le profil de l'utilisateur comparés à ceux des cas 1 et 2, ils restent aussi plus divers.

Un exemple similaire est exposé dans le Tableau 3.2 avec la requête « plant model » soumise par un éco-physiologiste : bien que diverses, lorsque la diversité des profils n'est pas prise en compte, les recommandations ne permettent aucune ouverture scientifique sur des communautés proches. À l'inverse, en incluant la diversité des profils, les résultats deviennent beaucoup plus intéressants et intègrent divers contenus, provenant de différentes communautés, toutes pertinentes.

En botanique, comme cela a été montré au Chapitre 1, un petit groupe de plantes diverses représentent la majeure partie des observations. Diversifier ces résultats ne permet donc pas de limiter ce problème puisque les observations le sont déjà. Ainsi, la description des contenus, bien que correcte, est trop succincte pour offrir réellement une bonne qualité de diversification : un expert préfèrera identifier des plantes provenant de certaines espèces plus rares plutôt que celles représentant la majeure partie des observations.

Abbar *et al.* [1] ont proposé un moteur de recommandation prenant en compte la diversité des commentaires ainsi que des méta-données relatives aux utilisateurs. Quant à Yu *et al.* [176], ils suggèrent que le calcul d'un ensemble divers doit être tel que chaque objet recommandé doit s'appuyer sur des utilisateurs différents. Ainsi, étant donné la matrice $users \times items$ et l'utilisateur u, chaque recommandation faite à u doit provenir d'un groupe d'utilisateurs différent de ceux utilisés pour les autres recommandations.

Ce Chapitre expose une approche originale pour la diversité des profils, exploitant un modèle probabiliste [149].

Table 3.1: Cas d'étude de la diversité des contenus et des profils avec la requête « lion ».

Methods	Case 1 No Diversity	Case 2 Content Diversity	Case 3 Profile Diversity 1	Case 4 Profile Diversity 2
Users	User 1	User 2	User 1	User 2
Profiles	Sea, boat, fishing	Savanna, jungle, Africa	Sea, boat, fishing	Savanna, jungle, Africa
Results				
				
				
				
				

Afin de pouvoir exécuter des requêtes utilisant la diversité des profils sur des jeux de données à grande échelle, de nouvelles techniques efficaces doivent être proposées. Notre diversification probabiliste [17, 39] s'appuie sur un calcul *top-k* utilisant des listes inversées. Comme cela a été présenté au Chapitre 2, une liste inversée associe un mot particulier du corpus (*i.e.* appartenant à au moins un contenu) à une liste triée des objets le contenant. Le tri est effectué en ordre décroissant en fonction de la pertinence de chacun par rapport au mot clé en question. Cependant, puisque notre modèle prend en compte la diversité, le calcul du score d'un objet ne dépend pas seulement de sa pertinence avec la requête mais aussi des objets précédemment ajoutés dans la liste de résultats : les *top-k* classiques ne sont donc plus utilisables. Pour cela, nous avons proposé un algorithme glouton permettant de calculer un *top-k* diversifié.

En résumé, les contributions de ce travail sont les suivantes :

1. proposition d'une fonction de score intégrant diversité des contenus et des profils au sein d'un modèle probabiliste ;
2. introduction d'un algorithme glouton de *top-k* basé sur les seuils (*i.e. threshold algorithm*) pour exécuter des requêtes avec le score proposé, utilisant le concept de liste des candidats ;
3. afin d'évaluer les bénéfices de notre fonction de score nous avons exécuté nos algorithmes sur trois jeux de données : deux provenant de *Delicious* et un de *Flickr*. Les résultats montrent une augmentation de la qualité globale des recommandations. Ces résultats ont été confirmés sur un jeu de données de phénotypage.

La suite de ce chapitre est organisée de la manière suivante : les bases de la recherche et de la recommandation pour les communautés en ligne sont tout d'abord exposées ; la problématique de diversification des profils sera également définie. Par la suite, nous décrivons notre fonction de score, notée *ProfDiv*, qui utilise un modèle de diversification probabiliste. Puis, les algorithmes nécessaires pour utiliser *ProfDiv* ainsi que deux techniques utilisant les retours utilisateurs pour adapter la diversité sont exposés en Section 3.4. En Section 3.5, nous comparons *ProfDiv* avec des modèles de diversification issus de l'état de l'art. La Section 3.6 se concentre sur les travaux connexes. Enfin, en Section 3.7, la conclusion et une présentation des perspectives de recherche.

3.2 Concepts de base et définition du problème

Dans cette section, les bases nécessaires pour délimiter le problème sont introduites.

Nous adoptons une approche hybride combinant *filtrage basé sur les contenus* [126] et *filtrage collaboratif* [61] où les profils utilisateurs - ou intérêts des

Table 3.2: Bénéfice de la diversité des profils pour les données de phénotypage avec la requête « plant model » soumise par un éco-physiologiste.

Undiversified Profiles		
Documents	Communities	Disciplines
Short-term responses of leaf growth rate to water defic...	Ecophysiologist community	Biologist discipline
Drought and Abscisic Acid Effects on Aquaporin Content...	Ecophysiologist community	Biologist discipline
Control of leaf growth by abscisic acid : hydraulic or non-hydraulic processes...	Ecophysiologist community	Biologist discipline
The importance of the anthesis-silking interval in breeding for drought tolerance in tropical maize...	Ecophysiologist community	Biologist discipline
Diversified Profiles		
Short-term responses of leaf growth rate to water defic...	Ecophysiologist community	Biologist discipline
A Multiscale Model of Plant Topological Structures...	Modeling community	Computer scientists discipline
Drought and Abscisic Acid Effects on Aquaporin Content...	Ecophysiologist community	Biologist discipline
Computational analysis of flowering in pea (Pisum sativum)...	Modeling community	Computer scientists discipline

utilisateurs - sont définis en fonction des objets $I_u = \{it_1, ..., it_w\}$ qu'ils partagent. Ainsi, nous disposons d'un ensemble d'utilisateurs $U = \{u_1, ..., u_n\}$. Un objet *it* peut être partagé par 1 à n d'entre eux. La popularité de ce dernier est un score proportionnel au nombre d'utilisateurs le partageant. Nous exploitons la recherche de recommandation dans le sens où chaque utilisateur peut soumettre des requêtes pour rechercher les objets parmi ceux recommandés et partagés par des utilisateurs similaires.

Un objet, ou contenu, est représenté de manière vectorielle [107, 141]. En utilisant $tf \times idf$, un objet est défini comme une liste de mots clés $k_1, ..., k_z$, et le vecteur représente le poids de chacun de ces derniers pour l'objet en question, étant donné le corpus global. Le profil d'un utilisateur u exprime ses intérêts ; il est calculé à partir des objets qu'il partage I_u. De manière plus précise, le profil d'un utilisateur est la moyenne des vecteurs $tf \times idf$ des objets qu'il partage. Les requêtes sont exprimées, quant à elles, par une liste de mots clés $k_1, ..., k_t$.

Définition du problème : étant donné un ensemble d'utilisateurs U, un ensemble d'objets I et une requête à mots clés q soumise par un utilisateur u, le problème est le suivant : il s'agit de recommander à u les *top-k* objets les plus pertinents et divers $R^q \in I$. On suppose que ces k objets sont triés par ordre décroissant de score dans la liste R^q.

L'intuition de notre approche est que des garanties de diversification peuvent être accrues en tenant compte du contenu des objets et du profil des utilisateurs les partageant. Ainsi, dans le calcul de R^q, nous identifions quatre critères pour la recherche et la recommandation, en fonction d'un objet it_i :

1. la similarité de it_i et de q ;
2. la diversification des contenus en fonction des objets déjà insérés dans R^q ;
3. la popularité de it_i ;
4. la diversification en fonction des profils des utilisateurs partageant les objets déjà choisis dans R^q. Ces profils doivent également être similaires à u et à q.

3.3 Modèle de score

De nombreuses méthodes ont été proposées pour la diversification [4, 17, 39, 176, 177]. Cependant, celles-ci n'adressent que le deuxième critère présenté précédemment. Certains travaux [176] pointent la limitation de la diversification des contenus et proposent de diversifier les utilisateurs (*i.e.* les utilisateurs partageant les objets doivent être différents). Notre but est d'introduire la diversification des profils (*i.e.* critère 4), en prenant en compte un modèle probabiliste de diversification offrant plus de garanties quant à la qualité de diversification, comme nous le montrons dans nos expériences en Section 3.5. La Section 3.3.1 présente le modèle probabiliste que nous utilisons, et la Section 3.3.2 introduit notre approche pour la diversification des profils.

3.3.1 Diversification probabiliste

Dans le domaine de la recherche d'information, étant donné I et une requête q, le calcul d'un *top-k* diversifié est un problème *NP-Complet*.

La fonction $div(it|\{it_1, ..., it_{i-1}\})$ [17, 39] est définie comme la probabilité que it soit divers (*i.e.* apporte de la nouveauté à l'utilisateur u) en fonction des objets précédemment choisis dans R^q (*i.e.* $\{it_1, ..., it_{i-1}\}$). Dans ce modèle, la diversité peut être exprimée en termes de redondance. Celle-ci, notée $red_c(it, it_j)$, peut se calculer par la similarité qu'il y a entre les objets it et it_j. Selon Angel et Koudas [17] la probabilité d'être divers est l'inverse de celle d'être redondant $1 - red_c(it|it_1, ..., it_{i-1})$. En utilisant l'hypothèse que la redondance entre deux objets est indépendanteres [62, 17, 39], le score probabiliste de diversification se définit comme suit :

$$1 - red_c(it|it_1, ..., it_{i-1}) = \prod_{it_j \in \{it_1, ..., it_{i-1}\}} (1 - red_c(it, it_j))^{\omega} \tag{3.1}$$

Où ω est un paramètre permettant d'introduire plus ou moins de diversité.

3.3.2 Fonction de score *ProfDiv*

Afin d'adresser les quatre critères présentés en Section 3.2, nous proposons cette nouvelle fonction de score :

$$score_{ProfDiv}(it, u, q) = \\ rel(it, q) \times div_c(it|\{it_1, ..., it_{i-1}\}) \times div_p(it|\{it_1, ..., it_{i-1}\}) \tag{3.2}$$

Ce score est donc la combinaison des éléments suivants :

- **pertinence :** $rel(it, q)$ est la probabilité que it réponde à la requête q. Celle-ci peut se définir comme la similarité entre it and q (*e.g. cosine, jaccard*) [141] et adresse le critère 1 ;
- **diversité des contenus :** $div_c(it|\{it_1, ..., it_{i-1}\})$ est une application directe de l'équation 3.1 et adresse le critère 2 ;
- **diversité des profils :** $div_p(it|\{it_1, ..., it_{i-1}\})$ est le score de diversification des profils d'un objet it – u_{it} représente les utilisateurs partageant l'objet it –, et prend en compte sa popularité (critère 3) et la diversification des profils des utilisateurs pertinents (critère 4).

De manière plus précise, le calcul de la diversité des profils se fait de la manière suivante. Pour chaque utilisateur dans U partageant it, un taux de confiance et un score de diversification sont ainsi évalués (critère 4) en fonction des objets déjà insérés dans R^q. Cette diversité est illustrée en Figure 3.1. Ce taux de confiance (*i.e. profile relevance*), noté $rel_p(u, v_i, q)$, indique la confiance qu'un utilisateur u peut avoir en un utilisateur v_i. Cette information peut se calculer de plusieurs manières (*e.g.* amis, position géographique, recommandations précédentes). Dans ce travail, nous considérons qu'elle prend en compte la pertinence d'un utilisateur v en fonction de u et de sa requête q. Celle-ci indique que v est proche de u (*i.e.* personnalisation) ainsi que de q (*i.e.* pertinent pour répondre à la requête q). L'équation suivante présente ce score :

$$rel_{trust}(u, v_i, q) = \alpha \times sim(u, v) + (1 - \alpha) \times sim(v, q) \tag{3.3}$$

De manière plus formelle, nous proposons le score de diversité de profil défini dans l'Équation 3.4. Rappelons que ce score tient compte de la popularité de it_i (requis 3) ; c'est pour cela que nous introduisons $\frac{1}{N}$, ce score étant également utilisé pour la normalisation.

$$div_p(it|\{it_1, ..., it_{i-1}\}) = \\ \frac{1}{N} \times \sum_{v \in u_{it_i}} \left[rel_p(u, v, q) \times \prod_{v_j \in \{u_{it_1}, ..., u_{it_{i-1}}\}} (1 - red_p(v_j|v))^{\beta} \right] \tag{3.4}$$

Où β est un score permettant d'introduire plus ou moins de diversité.

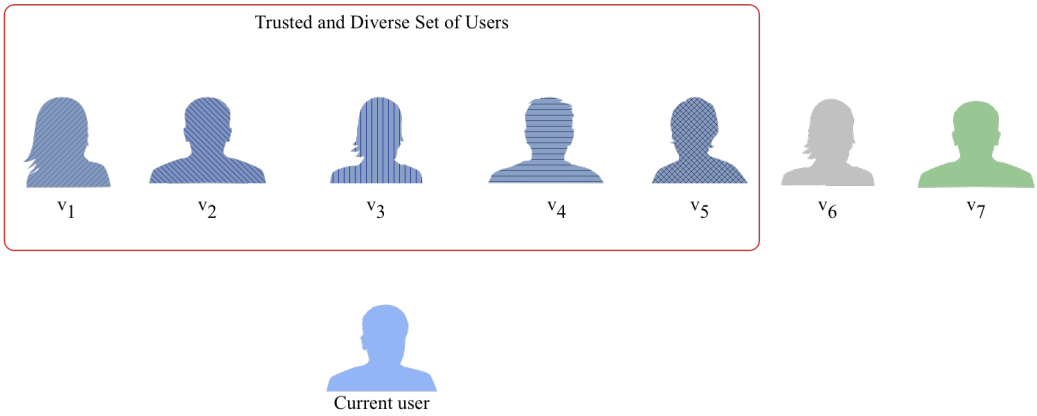


Figure 3.1: Avec la diversité des profils, les recommandations doivent provenir d'un ensemble d'utilisateurs de confiance (*i.e. trusted*), mais aussi divers.

3.4 Calcul des résultats divers

Nous présentons au sein de cette section le détail des algorithmes impliqués dans la diversification des profils. Dans un souci de clarté, en Section 3.4.1, la version adaptée de l'algorithme utilisé pour le calcul diversifié probabiliste [17] est décrite. Ensuite, en Section 3.4.2, nous introduisons celui utilisé pour le calcul de la diversité des profils.

3.4.1 Algorithme *top-k*

Angel et Koudas [17] proposent un algorithme (appelé *DAS*) pour la fonction de score diversifiée suivante :

$$rel(it, q) \times (1 - red(it|\{it_1, ..., it_{i-1}\})) \tag{3.5}$$

DAS est un algorithme à base de seuil (*i.e. threshold based*) qui, étant donné une requête q et un ensemble d'objets I, itère sur un ensemble de listes inversées :

$$\begin{aligned} &k_i \Rightarrow \langle it_a, w_{a,i}\rangle, \langle it_b, w_{b,i}\rangle, ..., \langle it_n, w_{n,i}\rangle \\ &... \\ &k_j \Rightarrow \langle it_e, w_{e,j}\rangle, ..., \langle it_n, w_{n,j}\rangle \end{aligned} \tag{3.6}$$

Où k_i est un mot, it_a un objet et $w_{a,i}$ le *score d'indexation* de l'objet par rapport au mot k_i (*e.g.* $w_{a,i} = sim(k_i, it_a)$). Les objets sont triés par score décroissant au sein de ces listes. Notons que le choix des listes inversées utilisées par l'algorithme dépend de la requête q – chacune étant associée à un mot clé, seules celles correspondant à ceux de la requêtes sont utilisées. Par exemple, si $q = \{k_i, k_j\}$, l'ensemble des listes sera alors composé de celles correspondant aux mots k_i et

k_m. Enfin, l'algorithme s'arrête lorsque la condition de seuil δ est satisfaite. Ce dernier est calculé à partir des listes inversées de la manière suivante :

$$\delta = f(s_i, ..., s_j) \tag{3.7}$$

où f est une fonction prédéfinie (*e.g.* cosinus) et s_i le dernier accès trié (*i.e. SA*) de la liste inversée correspondant au mot w_i. Par exemple, étant donné deux listes correspondant aux mots $\{w_i, w_j\}$, supposons que l'on veuille calculer le *top-1* objet, la condition d'arrêt sera satisfaite lorsqu'un élément aura un score supérieur ou égal à $\delta = f(s_i, s_j)$.

L'algorithme de *ProfDiv* est dérivé de *DAS*. La principale différence est le concept de liste des candidats utilisée dans le calcul de la diversité. Il s'agit de calculer une liste de résultats R^q telle qu'étant donné un objet $it_i \in R^q$, où $i \in \{1, ..., k\}$, il ne doit pas être possible de trouver un autre objet $it_j \notin \{it_1, ..., it_{i-1}, it_i\}$ tel que son score soit supérieur à celui de it_i en position i^{th} dans R^q. En d'autres termes l'algorithme doit garantir l'équation suivante :

$$\forall i \in \{1, ..., k\}, \nexists it \in I \backslash \{it_1, ...it_{i-1}\} | score(it, u, q) > score(it_i, u, q) \tag{3.8}$$

L'algorithme ne garantit cependant pas que la somme des scores des objets dans R^q soit maximale :

$$\max \left(\sum_{it \in R^q} score(it, u, q) \right) \tag{3.9}$$

Algorithme 4: *top-k ProfDiv*

Input: $q = k_i, ..., k_j$, u, k
Output: the top-k most relevant items wrt. to our scoring function.

```
1 R^q ⇐ ∅;
2 while |R^q| < min(k, |I|) do
3     it ⇐ index(k_i, ..., k_j).nextSortedAccess();
4     it.score = rel(it, q) × div_c(it|{it_1, ..., it_{i-1}}) × div_p(it|{it_1, ..., it_{i-1}});
5     add it to candidates;
6     compute δ ;
7     if ∃it ∈ candidates|score(it) > δ then
8         add it to R^q;
9         Update the score of the other candidates;
```

Le pseudo-code de *ProfDiv* est présenté dans l'Algorithme 4. L'entrée de ce dernier est la requête q, le profil de l'utilisateur courant u et le nombre de résultats souhaité k. L'algorithme s'arrête lorsque la liste R^q contient k objets (ligne 2) dont le score est supérieur au seuil. De la ligne 3 à 5, l'algorithme va effectuer des accès triés afin de récupérer les objets suivants. Puis, il va calculer leur score

(*i.e.* $score_{ProfDiv}$, formula 3.2) et l'insérer dans la liste des candidats (*i.e.* l'objet devient un candidat). Cette dernière contient tous les objets déjà accédés mais qui n'ont pas été insérés dans R^q car ils ne satisfont pas la condition de seuil : l'algorithme peut encore trouver des objets ayant des niveaux de diversification ou de pertinence supérieurs. Notons que le score d'un objet ne devient fixe que lorsqu'il est ajouté dans R^q.

En ligne 6, *ProfDiv* vérifie si le meilleur candidat (*i.e.* celui avec le score le plus élevé) satisfait la condition de seuil ; en d'autres termes, il vérifie si les listes inversées peuvent contenir un objet au score supérieur à ce candidat. Si le meilleur candidat est bien le meilleur objet non encore ajouté dans la liste R^q, *ProfDiv* va l'y insérer et mettre à jour le score de diversification des autres candidats (ligne 7 et 8).

3.4.2 Diversification des profils

Dans cette section, nous montrons comment calculer div_p (Algorithme 4, ligne 4).

L'algorithme 5 présente une méthode pour calculer div_p. Il prend en entrée la liste des objets déjà présents dans la liste des résultats R^q, l'utilisateur courant u, la requête q et l'objet courant (*i.e.* celui dont on calcule le score) it_i. De la ligne 1 à 7, l'algorithme va établir pour chaque utilisateur partageant it deux scores, le premier de confiance et le second de diversification. De manière plus précise, l'indice de confiance est calculé en ligne 3 sur v_n en fonction de u et de q. Ensuite, de la ligne 4 à 6, est évalué celui de diversification en fonction des utilisateurs partageant les objets déjà dans R^q.

Finalement, en ligne 7, l'algorithme combine l'indice de confiance et de diversification pour l'ajouter au score de diversification global. La ligne 8 permet de normaliser la valeur de div_p et ainsi de prendre en compte la popularité de it_i.

Le nombre d'itérations nécessaires à l'exécution de l'algorithme 5 est le suivant :

$$|u_{it}| \times \left| \bigcup_{it_i \in \{it_1, \ldots, it_{i-1}\}} u_{it_i} \right|$$

Où $|u_{it}|$ est le nombre d'utilisateurs partageant it. La complexité de la fonction est donc, dans le pire des cas, $\mathcal{O}(n^2)$, où n est le nombre total d'utilisateurs.

Rappelons que la redondance entre deux profils dépend de l'utilisateur soumettant la requête. Ce score ne peut donc pas être pré-calculé puisque cela nécessiterait un ensemble de listes inversées pour chaque utilisateur [9, 12].

3.4.3 Retours utilisateurs pour adapter la diversité

En Section 3.3.1, nous avons introduit deux paramètres permettant d'adapter la diversité : ω et β. Bien que la valeur par défaut de ces derniers soit fixée à

Algorithme 5: Profile Diversification Score Computation

Input: $\{it_1, ..., it_{i-1}\}$,u,q,it
Output: The profile diversity score of it wrt. q, u and $\{it_1, ..., it_{i-1}\}$
`/* the items are indexed based on sim(it,q). */`

$profDiv \Leftarrow 0$;
for v **in** u_{it} **do**
  $t \Leftarrow trust(u, v, q)$;
  $div \Leftarrow 1$;
  for v_j **in** $\bigcup_{it_i \in \{it_1,...,it_{i-1}\}} u_{it_i}$ **do**
    $div \Leftarrow div \times red(v, v_j)$;
  $profDiv \Leftarrow profDiv + t \times div$;
$profDiv \Leftarrow \frac{profDiv}{N}$;

1, il n'est pas évident de trouver celle qui maximise la satisfaction des utilisateurs. Nous proposons dans cette section d'exploiter les retours utilisateurs – ou *feedbacks* – afin d'obtenir de meilleurs résultats.

À chaque fois qu'une requête est soumise, les utilisateurs peuvent fournir leur avis sur le niveau de diversification. Ils peuvent ainsi indiquer s'ils souhaitent **plus** ou **moins** de diversité, jusqu'à ce qu'ils obtiennent le niveau qu'ils désirent. Notre objectif est d'agréger ces retours afin de calculer les meilleures valeurs de ω et de β. Pour cela, nous proposons deux approches :

1. **Adaptation de la diversité par utilisateur :** chaque fois qu'un utilisateur donne son avis sur le niveau de diversité, les valeurs de ω et de β correspondant au niveau qu'il préfère sont sauvegardées avec son profil. Ensuite, lorsque de nouvelles requêtes sont soumises par le même utilisateur, les valeurs de ω et de β choisies par le système, sont déduites du profil de ce dernier, *e.g.* la moyenne ou la médiane des *feedbacks* précédents. Avec cette approche, chaque utilisateur possède un niveau différent de diversité.
2. **Adaptation de la diversité par requête :** chaque fois qu'un utilisateur donne son avis sur le niveau de diversité d'une requête $q = k_1, ..., k_t$, les valeurs de ω et de β sont sauvegardées et agrégées à leurs précédentes valeurs associées à chacune des listes inversées correspondant au mots clés $k_1, ..., k_t$. La méthode d'agrégation peut être la moyenne ou la médiane. Ensuite, lorsqu'une nouvelle requête $q' = k'_1, ..., k'_t$ est soumise au système, les valeurs de ω et de β choisies sont calculées comme l'agrégation de la valeur qui leur est associée dans chaque liste inversée correspondant aux mots clés $k'_1, ..., k'_t$. Avec cette approche, chaque utilisateur a le même niveau de diversité, mais celui-ci diverge lorsque les requêtes sont différentes.

La Figure 3.2 présente l'architecture que nous avons utilisée pour gérer dynamiquement les retours des utilisateurs afin d'adapter les niveaux de diversité. Les

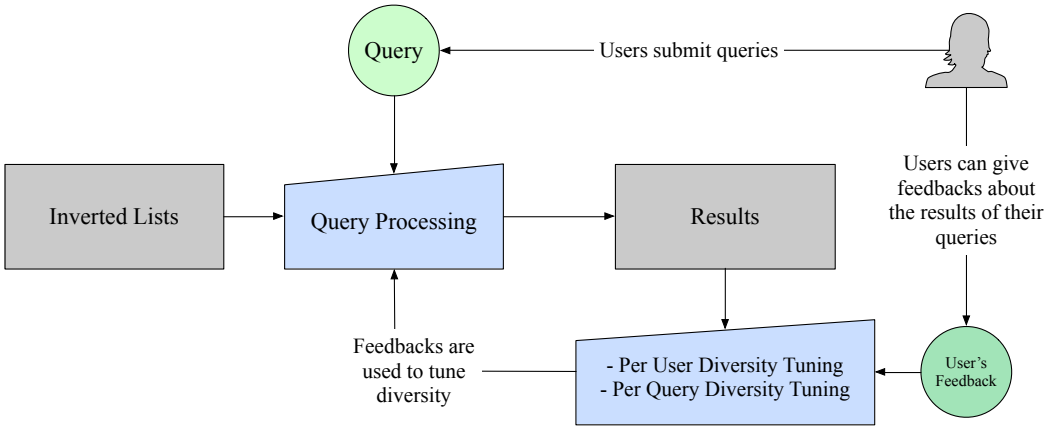


Figure 3.2: Architecture implémentée afin d'adapter le niveau de diversité aux avis retournés par les utilisateurs.

précédents retours utilisateurs sont utilisés au moment du traitement des requêtes afin d'adapter le niveau de diversité. Lorsqu'un utilisateur reçoit les résultats de sa requête, il peut donner ses retours.

3.5 Évaluation expérimentale

Cette section présente l'évaluation expérimentale de nos approches afin de valider la qualité des recommandations et de la diversification des contenus et des profils. Nous avons mené nos expériences via l'utilisation d'un jeu de données provenant de *Delicious*, d'une version enrichie de ce dernier, ainsi que d'un autre jeu provenant de *Flickr*. En parallèle nous utilisons un jeu de données de phénotypage pour confirmer les résultats précédents. La Section 3.5.1 décrit la mise en place des expériences, puis, la Section 3.5.2 présente les résultats.

3.5.1 Mise en place des expériences

Nous avons validé les gains en termes de qualité de diversité des profils par utilisation de jeux de données du Web ces derniers étant plus volumineux, nous permettant ainsi de réaliser des sondages plus importants et des tests plus réalistes. Cependant, les résultats obtenus à petite échelle sur les données de phénotypage (*i.e.* 300 documents scientifiques partagés par 30 vrais chercheurs ainsi que 180 requêtes) et de botanique (10 000 observations soumises par 1 500 vraies personnes) confirment les résultats.

Nos expériences ont été menées sur trois jeux de données différents. Il s'agit tout d'abord de 30 000 signets téléchargés de *Delicious*, associés à environ 55 000 étiquettes uniques soumises par 2 000 utilisateurs. Les signets précédents ont été

enrichis de leur contenu `html` afin de réaliser un second jeu de données : cela permet d'observer le comportement de nos algorithmes lorsque chaque objet est associé à peu de mots clés ou à l'inverse à un grand nombre. Enfin, le troisième jeu de données est composé de 3.25 millions d'images récupérées du site *Flickr* associées à 3.5 millions groupes d'étiquettes soumises par environ 272 000 utilisateurs ; chaque groupe est en réalité une liste de une à quinze étiquettes. L'ensemble des mots clés disponibles (*i.e.* titre, étiquettes, description ou contenu `html`) est utilisé pour l'indexation.

Une expérience consiste à analyser les résultats des algorithmes lorsque des requêtes sont soumises. Ces requêtes sont soit créées automatiquement, soit directement soumises par les utilisateurs lors de *sondages*. Dans le premier cas, elles sont construites comme l'association des étiquettes soumises par un unique utilisateur sur une unique image en un seul jour ; étant donné que la personnalisation est prise en compte, le profil de ce même utilisateur sera utilisé lorsque la requête sera soumise.

Évaluation de la qualité des expériences

Afin d'évaluer la qualité des recommandations et de la diversification, nous avons procédé en deux étapes : tout d'abord, des mesures *automatiques et numériques* des résultats, puis, un *sondage utilisateur*.

L'évaluation automatique consiste à soumettre toutes les requêtes créées sur différents algorithmes de *top-k* et à utiliser certaines mesures afin d'analyser la qualité des résultats. Notons tout de même que les résultats sont filtrés afin de ne pas tenir compte des objets partagés par l'initiateur de chaque requête. Finalement, les algorithmes de *top-k* comparés sont les suivants :

1. `Diversification probabiliste de contenu` : le modèle probabiliste de diversification décrit en Section 3.3.1.
2. `Max-Min` : la similarité maximale entre chaque couple d'objets de la liste de résultats doit être minimisée.
3. `Max-Sum` : la somme des similarités entre chaque couple d'objets doit être minimisée.
4. `Diversification probabiliste des profils` : notre fonction de score où l'indice de confiance est fixé à 1.
5. `Diversification probabiliste et personnalisée des profils` : notre fonction de score.

Nous avons ensuite utilisé les métriques suivantes :

1. `Pertinence` : la similarité moyenne entre chaque objet de la liste de résultats et la requête :

$$rel = \sum_{it \in R^q} \frac{rel(it)}{|R^q|}$$

2. `Diversification de contenu` : la distance moyenne entre chaque couple de la liste de résultats :

$$div_c = \sum_{it_i \in R^q} \sum_{it_j \in R^q} \frac{1 - sim(it_i, it_j)}{|R^q|^2}$$

3. `Diversification de profils` : la distance moyenne des profils utilisateurs partageant un objet par rapport à ceux qui en partagent un autre, pour chaque couple d'objets :

$$div_p = \sum_{it_i \in R^q} \sum_{it_j \in R^q} \frac{1 - sim(f(u_{it_i}), f(u_{it_j}))}{|R^q|^2}$$

4. `Confiance` : La similarité moyenne entre l'utilisateur ayant soumis la requête et ceux partageant les objets :

$$trust = \sum_{it \in R^q} \frac{rel(u, f(u_{it_i}), q)}{|R^q|}$$

Lors de la seconde étape, la qualité de la diversification des profils a tout d'abord été évaluée. Puis, elle a été combinée à l'indice de confiance. Le sondage a été réalisé pour les jeux de données *Delicious* et *Flickr*. Puisque la première partie se focalise principalement sur l'effet de la diversification des profils sur la qualité, nous avons comparé les scores suivants :

1. `Simple-topk` : seule la similarité entre la requête q et un objet est prise en compte.
2. `Diversification des contenus` : les objets sélectionnés sont les plus pertinents par rapport à la requête mais aussi les plus divers entre eux en fonction de leurs étiquettes.
3. `Diversification des profils` : la fonction de score où le niveau de confiance est fixé à 1.

Pendant la seconde partie du sondage, nous avons comparé les algorithmes suivants :

1. `Diversification des contenus personnalisée` : le modèle probabiliste de diversification associé à un score de confiance.
2. `Diversification des profils` : la fonction de score où le niveau de confiance est fixé à 1.
3. `Diversification des profils personnalisée` : notre fonction de score.

Enfin, lors de la troisième partie du sondage, le système a essayé, en fonction des retours utilisateurs, d'adapter le niveau de diversité (*i.e.* ω et β) en utilisant la méthode *par utilisateur* ou celle *par requête*, comme cela est présenté en Section 3.4.3. Afin d'analyser l'effet induit par ces deux méthodes, nous comparons les scores suivants :

1. `Simple-topk` : seule la similarité entre la requête q et un objet est prise en compte.
2. `Diversification des contenus` : les objets sélectionnés sont les plus pertinents par rapport à la requête mais aussi les plus divers entre eux en fonction de leurs étiquettes.
3. `Diversification des profils` : la fonction de score où le niveau de confiance est fixé à 1 et où les paramètres de diversité sont calculés en fonction des retours utilisateurs.

Les questions suivantes ont été posées aux sondés (les réponses aux questions 1-2 étaient une valeur entre 1 et 5) :

1. Est-ce que la liste de résultats est pertinente par rapport à la requête ?
2. Quel est le niveau de diversification des résultats ?
3. Pouvez-vous classer les listes de résultats (*i.e.* algorithmes) de la meilleure à la moins bonne en termes de pertinence et de diversité (deux listes peuvent avoir une qualité identique) ?

Afin d'évaluer correctement un score de confiance, une étape de *training* a été réalisée durant laquelle chaque utilisateur devait soumettre un ensemble de requêtes pour choisir des résultats qu'il aimait ; leur profil a ensuite été construit à partir de ces images.

Nous avons interrogé une vingtaine de personnes pour plus de 190 requêtes.

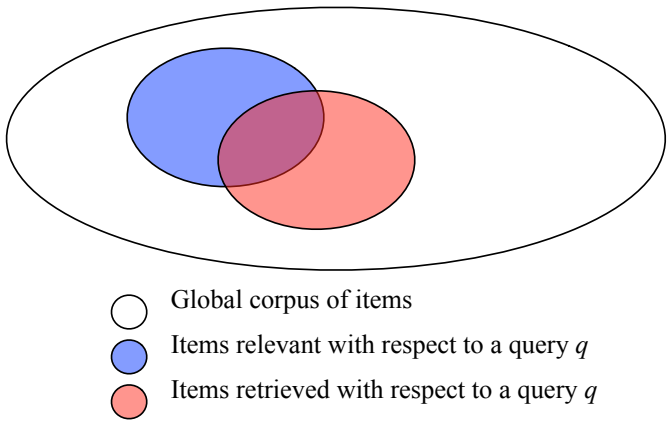


Figure 3.3: Illustration de la définition et du rappel en recherche d'information et recommandation.

Discussion sur le rappel et la précision : le rappel est une mesure permettant d'évaluer la quantité d'objets *oubliés* par le système. Cela est illustré par la

partie orangée qui n'est pas en intersection dans la Figure 3.3. Il se calcule de la manière suivante :

$$recall = \frac{|Items_{relevant} \cap Items_{retrieved}|}{|Items_{relevant}|} \tag{3.10}$$

Où $Items_{relevant}$ représente l'ensemble des objets pertinents et $Items_{retrieved}$ uniquement ceux que le système a retrouvé. À l'inverse, la précision mesure le nombre d'objets qui ne sont pas pertinents et qui ont été malgré tout retournés par le système. Dans la Figure 3.3, cela est représenté par le bleu qui n'est pas en intersection. La précision se calcule de la manière suivante :

$$precision = \frac{|Items_{relevant} \cap Items_{retrieved}|}{|Items_{retrieved}|} \tag{3.11}$$

Ces deux mesures sont généralement opposées : quelle valeur du rappel pour telle valeur de la précision. Elles ne sont cependant plus adéquates pour évaluer les modèles introduisant de la diversification pour accroître la qualité des résultats de recherche ou de recommandations [112, 176].

Une intuition simple est la suivante. Généralement une très forte précision est accompagnée d'un rappel faible puisque peu d'objets sont retournés. Afin d'accroître le rappel, il suffit d'augmenter le nombre d'objets à retourner ce qui aura pour effet de réduire la précision. Confronter ces deux méthodes permet de voir le compromis de chaque système entre précision et rappel. Introduire de la diversité pénalise volontairement les résultats afin de fournir de la nouveauté aux utilisateurs. Le rappel et la précision diminuerait alors simultanément, et ces modèles seraient jugés inférieurs aux modèles non-diversifiés. Les études utilisateurs ou l'analyse de leur comportement en ligne représentent la majeure partie des expériences sur la diversité [176].

3.5.2 Résultats expérimentaux

Les résultats de la première expérience sont présentés dans la Figure 3.4.

Tout d'abord, nous pouvons observer qu'en Figures 3.4a et 3.4b, le modèle probabiliste de *diversification des contenus* et les deux méthodes de *diversification des profils* obtiennent des résultats similaires en termes de pertinence et de diversité des contenus, alors que les deux autres solutions de diversification sont environ 10% moins efficaces en termes de diversité des contenus et possèdent une pertinence légèrement supérieure.

Il est également possible de remarquer, dans la Figure 3.4a (*i.e.* jeu de données *Flickr*), que notre fonction de score personnalisée améliore légèrement la diversification des profils et permet un gain considérable de l'indice de confiance avec un résultat plus de deux fois supérieur aux autres méthodes. La diversification des profils permet des résultats similaires en termes de confiance comparée aux

autres méthodes mais permet un gain de 15% en termes de diversification des profils.

La Figure 3.4b (*i.e.* jeu de données *Delicious*) montre que notre fonction de score personnalisée permet d'augmenter de 20% la diversification des profils et de multiplier par deux l'indice de confiance face aux autres méthodes. Le score de diversification des profils obtient un niveau de confiance légèrement moindre mais permet d'obtenir une diversification des profils plus de 30% supérieure aux autres méthodes.

En d'autres termes, il est possible d'améliorer la diversification des profils sans diminuer la qualité de diversification des contenus et de pertinence.

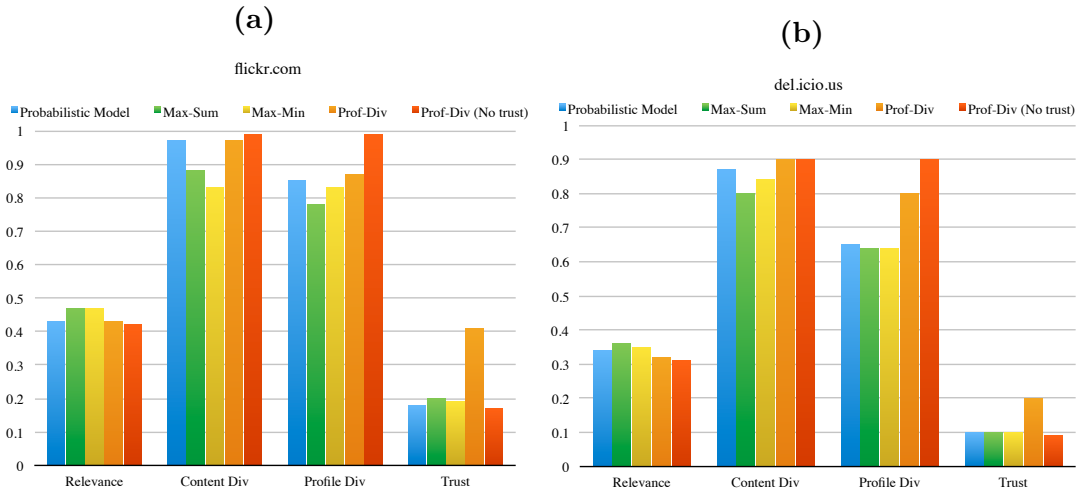


Figure 3.4: Évaluation automatique de la qualité au travers des métriques suivantes : pertinence des résultats, diversités des contenus et des profils et indice de confiance.

Cependant, bien que, numériquement, la diversification des contenus reste similaire suivant les méthodes utilisées, l'expérience utilisateur peut être différente. En effet, les étiquettes ne décrivant pas nécessairement correctement les objets, il est possible que la diversification réellement ressentie par l'utilisateur ne soit pas du niveau calculée par le système. Afin d'évaluer cet aspect, nous avons réalisé trois sondages utilisateurs présentés dans les Figures 3.5 et 3.6. La première figure présente les méthodes suivantes : *top-k non diversifié*, *top-k avec diversification des contenus* et *top-k avec diversification des profils non personalisée*.

Les Figures 3.5a et 3.5b montrent que les utilisateurs trouvent les listes de résultats générées en utilisant la diversification des profils à la fois plus diverses mais également plus pertinentes que celles diversifiées à partir de leurs contenus. Les listes non diversifiées sont quant à elles à la fois plus pertinentes mais également beaucoup moins diverses que les deux autres méthodes.

Dans les Figures 3.5c et 3.5d, les utilisateurs se voyaient présenter deux listes

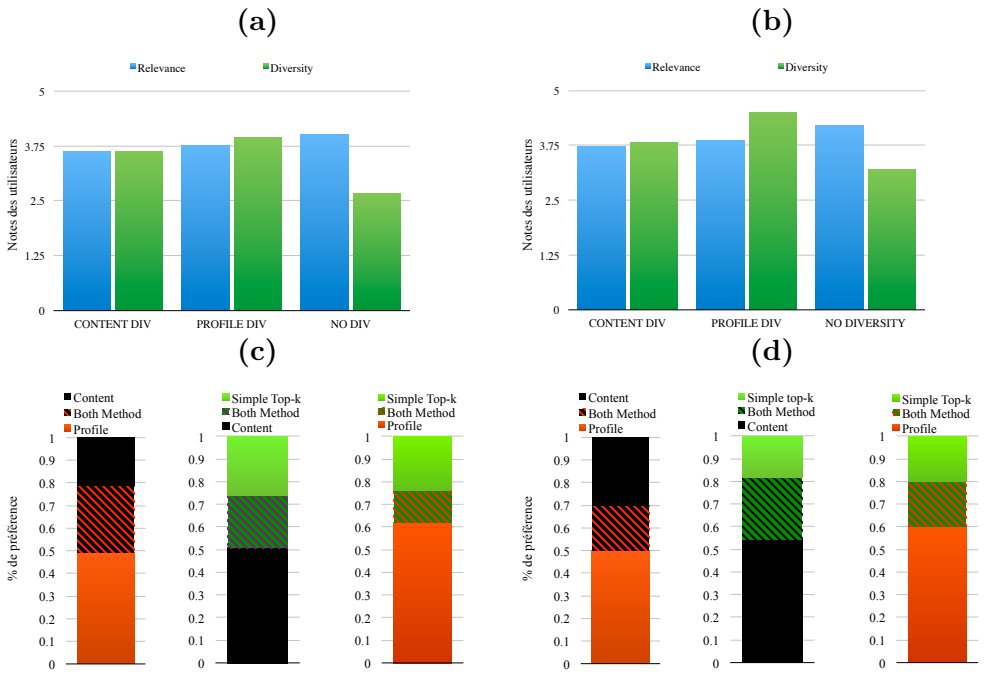


Figure 3.5: Sondage pour évaluer la qualité des fonctions de score suivantes : pertinence, diversification des contenus et diversification des profils non personnalisée.

de résultats (toutes les méthodes étaient comparées) et devaient choisir celle qu'ils préféraient. Chaque couleur représente le pourcentage de fois qu'une liste a été préférée à une autre et les couleurs rayées indiquent que les deux listes étaient considérées similaires par les utilisateurs.

Les Figures 3.5c et 3.5d montrent que la moitié du temps, la diversification des profils est préférée à la celle des contenus. Qui plus est, dans plus de 60% des cas la diversification des profils était préférée à la liste non diversifiée. Dans 20% des cas sur *Delicious* et 30% sur *Flickr*, la diversification des profils était similaire à celle des contenus. Dans 15 et 20% des cas sur *Delicious* et *Flickr* respectivement, la liste non-diversifiée et celle exploitant la diversification des profils étaient similaires. Enfin, dans plus de la moitié des cas, la diversification des contenus était meilleure que la liste non diversifiée, et dans 20 à 30% des cas sur *Delicious* et *Flickr* respectivement, les deux listes étaient similaires.

Discussion : dans la majorité des cas, la diversité (des profils ou non) permet d'améliorer la satisfaction de l'utilisateur. Mais tenir compte des profils dans le processus de diversification accroît encore plus cette dernière.

Dans la Figure 3.6, les méthodes analysées étaient : *top-k avec diversité des*

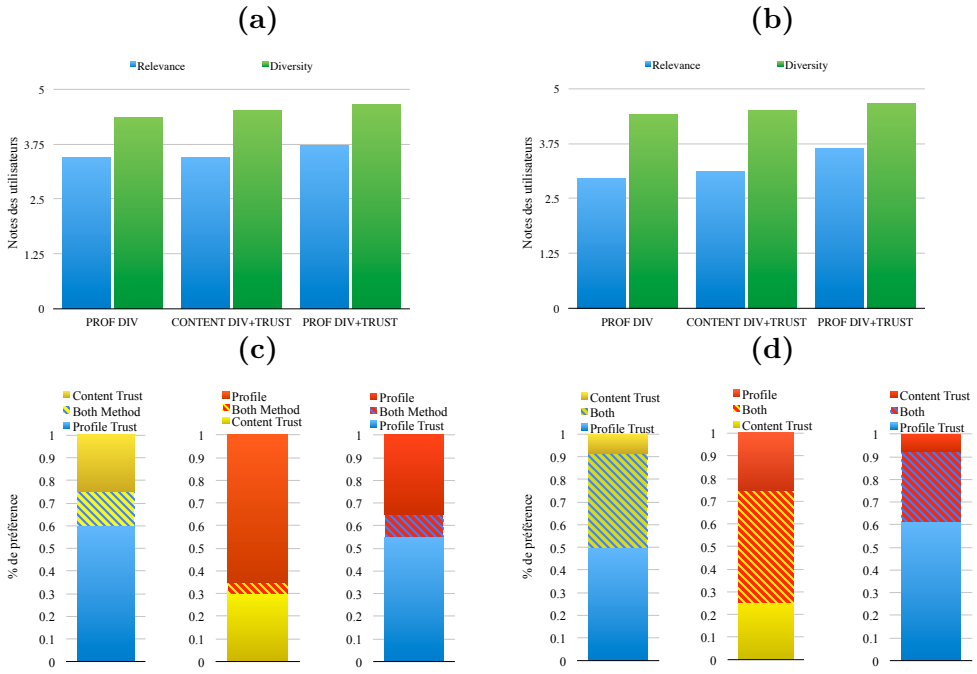


Figure 3.6: Sondage pour évaluer la qualité des fonctions de score suivantes : diversité des contenus personnalisée, diversification des profils personnalisée ou pas.

profils, *top-k avec diversité des contenus personnalisée* et *top-k avec diversité des profils personnalisée*.

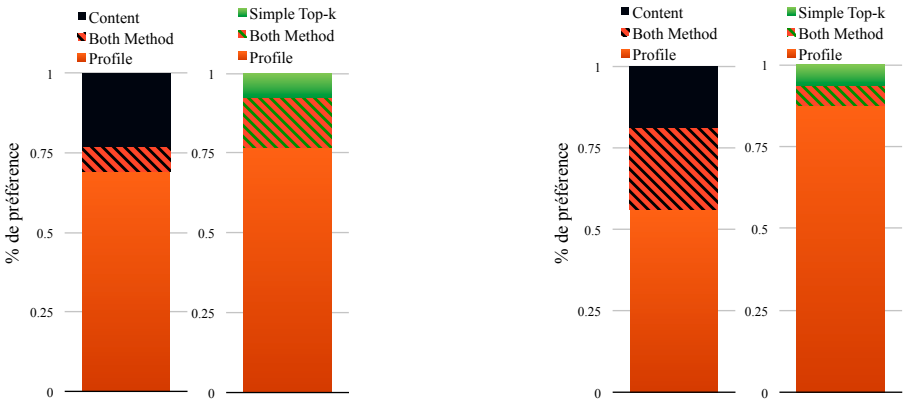
Les Figures 3.6a et 3.6b montrent que les utilisateurs trouvent toutes les méthodes plutôt similaires en termes de pertinence et redondance malgré un léger gain pour la diversité des profils.

Dans les Figures 3.6c et 3.6d, les utilisateurs se voyaient présenter deux listes de résultats (toutes les méthodes étaient comparées) et devaient choisir celle qu'ils préféraient. Chaque couleur représente le pourcentage de fois qu'une liste a été préférée à une autre et les couleurs rayées indiquent que les deux listes étaient considérées comme similaires par les utilisateurs.

Les Figures 3.6c et 3.6d montrent que dans 50 à 60% des cas sur *Del.icio.us* et *Flickr* respectivement, la diversification des profils personnalisée était meilleure que la diversité des contenus personnalisée. Ensuite, dans 15 et 40% des cas sur ces deux mêmes jeux de données, les deux méthodes étaient similaires. En d'autres termes, dans 75 à 90%, la diversification des profils personnalisée était meilleure ou similaire à la diversification des contenus personnalisée. La comparaison entre la diversification des profils non personnalisée et celle des contenus personnalisée retourne des résultats très différents suivant le jeu de données utilisés. En effet, dans 65% des cas sur *Flickr*, la diversification des profils est préférée à celle des contenus personnalisée. Sur *Delicious*, ce chiffre descend à 25%, et les méthodes

sont similaires dans 50% des cas. Finalement, dans 55 à 60% des cas, la diversification des profils personnalisée est considérée comme meilleure que la diversification des profils ; et dans 10 à 35% des cas, les deux solutions sont considérées comme similaires.

Discussion : deux conclusions peuvent être formulées. Tout d'abord, tenir compte de la confiance n'améliore pas nécessairement la satisfaction de l'utilisateur lorsqu'elle s'accompagne d'une diversification des contenus. En second lieu, combiner la confiance avec la diversification des profils permet un gain de satisfaction considérable.

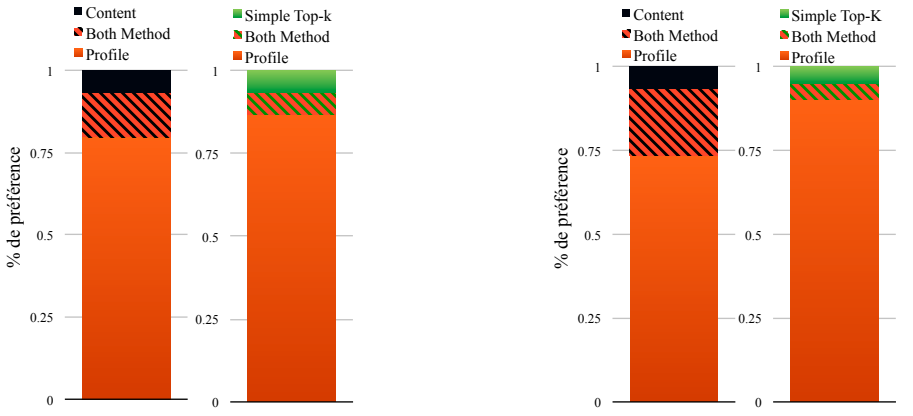


(a) Jeu de données *flickr* **(b)** Jeu de données *delicious*

Figure 3.7: Évaluation de la satisfaction des utilisateurs lorsque le système adapte la diversité pour chaque utilisateur en fonction de leurs retours.

Nous analysons maintenant l'effet qu'adapter la diversité en fonction des retours utilisateurs (*i.e.* ω et β, cf. Section 3.4.3) peut avoir sur leur satisfaction. Les expériences sont composées d'un ensemble de requêtes d'*apprentissage* et de *tests*. Afin de construire l'ensemble d'apprentissage, les utilisateurs devaient soumettre des requêtes et donner leurs avis (*i.e. feedbacks*) pour l'ensemble d'entre elles. Les Figures 3.7 et 3.8 présentent les résultats de nos expériences. La Figure 3.7 introduit ces résultats lorsque ω et β sont calculés avec la méthode *par utilisateur*. Sans surprise, apprendre à quel point un individu donné aime la diversité permet d'accroître sa satisfaction. Comme cela est montré en Figure 3.7, la diversité des profils est maintenant préférée dans 77 à 82% des cas. Les gains sont encore plus grands lorsque cette méthode est comparée à un *top-k* simple.

La Figure 3.8 montre les résultats obtenus lorsque ω et β sont calculés en utilisant la méthode *par requête*. Les gains sont supérieurs à ceux de la méthode



(a) Jeu de données *flickr* **(b)** Jeu de données *delicious*

Figure 3.8: Évaluation de la satisfaction des utilisateurs lorsque le système adapte la diversité pour chaque requête en fonction des retours utilisateurs.

par utilisateur. Comme cela est montré, les utilisateurs préfèrent la diversité des profils jusqu'à 94% des cas. Cela peut s'expliquer de la manière suivante. Certains thèmes ou sujets, comme la littérature, peuvent utiliser (*i.e.* contenir) beaucoup de synonymes et les résultats apparaitront au système comme très divers, alors qu'ils ne le sont pas vraiment. D'autres thèmes, comme l'informatique, utilisent beaucoup moins de synonymes, et les résultats sembleront redondants alors qu'ils ne le sont pas tant que ça non plus. Ainsi, apprendre des utilisateurs quelles requêtes nécessitent plus ou moins de diversité permet de réduire cet effet et d'accroître significativement la satisfaction de ces derniers.

Remarques sur les données de phénotypage : après avoir réalisé un sondage à petite échelle, il en résulte que la diversité des profils montre notamment un réel avantage dans les cas où les scientifiques cherchent des contenus multi-disciplinaires (*e.g.* un éco-physiologiste cherchant une méthode pour modéliser une plante).

3.6 Travaux connexes

La diversité des contenus a été étudiée dans les domaines de la recherche sur le Web, des requêtes en base de données et pour la recommandation. Diversifier les résultats de recherche du web ou de recommandations a pour objectif d'atteindre un compromis entre pertinence et hétérogénéité des résultats. Dans [62], les auteurs adoptent une approche axiomatique de la diversité où l'objectif est d'adresser l'intention de l'utilisateur. Ils montrent notamment qu'il n'est pas possible de

définir une fonction de score diversifiée satisfaisant l'ensemble des axiomes proposés simultanément. Dans [15], les taxonomies sont utilisées afin d'échantillonner les résultats d'une recherche permettant par là d'en réduire l'homogénéité. Dans le cadre des bases de données [40, 169], certaines solutions se proposent de structurer les résultats en post-traitement en les organisant par exemple en un arbre de décision [40] permettant une navigation plus simple parmi les résultats. Dans [13], l'idée d'une diversité hiérarchique dans les bases de données est introduite et des algorithmes de *top-k* efficaces sont proposés.

Dans certains travaux sur la recommandation, *e.g.* [52, 176, 179], les résultats sont post-traités en utilisant les similarités entre les objets afin de générer une liste offrant un compromis entre précision (*i.e. accuracy*) et diversité. Par exemple, dans [179] les auteurs ont défini une méthode de similarité intra-liste qui s'appuie sur la connexion de chaque objet avec une taxonomie afin de déterminer le thème, ou en utilisant des attributs de l'objet comme l'auteur ou le genre. Cette méthode s'appuie sur un calcul exhaustif où une première liste de N éléments est générée, maximisant la pertinence, un sous-ensemble divers de k éléments en est déduit (n>k). À l'inverse, dans [52], la diversité est formulée comme un problème de couverture d'ensemble. [57] introduit un *framework* pour la diversité dans la publicité sponsorisée lors dans les moteurs de recherche. Les auteurs proposent des algorithmes pour sélectionner les publicités permettant d'accroître l'hétérogénéité permettant à chaque publicitaire de garder leurs gains au plus haut niveau possible. L'hétérogénéité est abordée dans le sens où une seule requête peut être vue comme ayant plusieurs significations. Dans [1], les auteurs proposent une technique de diversification basée sur les commentaires laissés par les utilisateurs sur des objets plutôt que sur le contenu de ces derniers. Ils proposent d'intégrer de la diversification afin d'améliorer la navigation au sein d'articles de journaux en ligne. *DisC* [50] calcule la diversité d'un ensemble d'objets R de telle manière que chaque chaque objet pertinent étant donné la requête de l'utilisateur doit être représenté par un objet dans R. Dans [50], la diversité n'est pas calculée comme un ensemble de k éléments où k est défini par l'utilisateur mais comme un ensemble d'éléments tel que tous les objets pertinents étant donné une requête ont une distance avec un objet de R inférieur à r (*i.e.* radius).

Cependant, ces contributions n'abordent pas le problème de diversification de la même manière que nous. Dans notre approche, nous proposons d'associer les objets avec les profils des utilisateurs les partageant, et de diversifier les résultats en exploitant ces profils.

3.7 Conclusion et perspectives de recherche

Dans ce chapitre, la diversité des profils a été introduite afin d'améliorer la qualité de la diversification dans la recherche et la recommandation.

Nous avons proposé une fonction de score tenant compte de la pertinence de

la requête, de la diversité des contenus, de la popularité des objets ainsi que de la diversité des profils, dits de confiance. Cela permet de retourner des résultats limitant l'effet d'ambiguïté des mots clés ou de mauvaises descriptions des objets. La diversification a ainsi de fortes probabilités d'être de meilleure qualité. Nous avons également proposé un nouvel algorithme à seuil afin de calculer des *top-k* diversifiés. Enfin, nous avons introduit l'idée de considérer les retours utilisateurs pour adapter les niveaux de diversités et accroître leur satisfaction.

À travers une évaluation expérimentale sur trois jeux de données, nous avons comparé la diversification des profils avec d'autres fonctions de score et nous avons montré qu'elle présente le meilleur compromis entre tous les critères que nous avions identifiés. Un sondage utilisateur confirme que la diversité des profils est préférée dans la majeure partie des cas. Ces résultats sont validés sur deux jeux de données à petite échelle : l'un de phénotypage et l'autre de botanique.

Cependant, le calcul d'un *top-k* reste complexe et les temps de réponses sont élevés. Dans le prochain chapitre, nous étudions des solutions d'optimisation afin de réduire le coût de traitement des requêtes.

De nombreuses pistes restent exploitables. Ainsi, nous avons introduit dans ce chapitre le fait d'exploiter les *feedbacks* des utilisateurs afin d'adapter les niveaux de diversités. Nous avons proposé d'agréger ces *feedbacks* en utilisant des techniques comme la moyenne afin d'adapter les niveaux futurs de diversité de chaque utilisateur. Il peut être intéressant d'appliquer à ce stade d'autres techniques. En s'appuyant sur l'idée de recommandation, il s'agit par exemple de recommander une « note de diversité » étant donné un utilisateur u et l'objet « requête q ».

CHAPITRE 4

Techniques d'optimisation dans le calcul d'un *top-k* divers

Résumé. Ce chapitre décrit en détail diverses méthodes d'optimisation utiles au calcul d'un top-k *divers. Nous présentons dans un premier temps les points limitant des algorithmes de* top-k *divers et nous définissons le problème. Nous décrivons ensuite nos méthodes d'optimisation. Nous validons expérimentalement les bénéfices de ces dernières en utilisant trois jeux de données. Nos optimisations ont permis dans certains cas d'obtenir des temps de réponse plus de 12 fois plus faibles qu'un algorithme* top-k *classique.*

4.1 Introduction

Le chapitre précédent a introduit la notion de diversité des profils, accompagnée des algorithmes de *top-k* permettant le calcul d'un ensemble divers respectant ce modèle. Ces algorithmes s'appuient sur plusieurs structures de données comprenant notamment des listes inversées et une liste des candidats. Comme cela a été présenté au Chapitre 2, une liste inversée associe un mot particulier du corpus (*i.e.* appartenant à au moins un objet) à une liste triée des objets le contenant. Le tri est effectué en ordre décroissant en fonction de la pertinence de chacun des objets par rapport au mot clé en question. Cependant, puisque notre modèle prend en compte la diversité, le calcul du score d'un objet ne dépend pas seulement de sa pertinence avec la requête mais aussi des objets précédemment ajoutés dans la liste des résultats : les *top-k* classiques ne sont donc plus utilisables. Pour cela, nous avons introduit au chapitre précédent la notion de liste des candidats qui est, en réalité, une étape intermédiaire permettant le calcul des scores de diversification de chaque objet.

Nous identifions cependant trois points de l'algorithme rendant le calcul d'un *top-k* potentiellement lent. Tout d'abord, notre modèle de score est complexe et, comme cela a été précisé au chapitre précédent, ne peut pas être pré-calculé puisqu'il dépend du profil de l'utilisateur soumettant la requête – son pré-calcul nécessiterait énormément de stockage puisque les listes inversées devraient être dupliquées en autant d'exemplaires qu'il n'y a d'utilisateurs [9]. Cette complexité entraîne donc un accroissement du temps de calcul. En second lieu, le nombre d'objets candidats à considérer à chaque requête est considérable alors même que certains d'entre eux sont très redondants – et ne seront donc jamais retournés à l'utilisateur. Le calcul des scores de diversité de ces candidats représente un large pourcentage du temps de réponse. Enfin, le nombre d'accès dans les listes inversées est élevé, alors même que les k objets qui seront retournés à l'utilisateur peuvent avoir déjà été accédés. Nous présentons dans ce chapitre des optimisations adressant chacun de ces trois points limitants.

En résumé, nos contributions sont les suivantes :

1. simplification du modèle afin de permettre le pré-calcul ;
2. optimisation du seuil, et score d'indexation diversifié afin de limiter le nombre d'accès aux listes inversées ;
3. sélection de candidats pour limiter le nombre de scores diversifiés à calculer.
4. afin d'évaluer les bénéfices de nos optimisations, nous avons exécuté nos algorithmes sur trois jeux de données : deux provenant de *Delicious* et un de *Flickr*. Les résultats montrent une réduction significative du temps de réponse lors du calcul d'un *top-k* diversifié, grâce à nos techniques d'optimisation.

La suite de ce chapitre est organisée de la manière suivante : les bases de notre algorithme de *top-k* et de ses points limitants sont rappelés en Section 4.2, où le problème sera également défini. La Section 4.3 introduit l'ensemble des techniques d'optimisation que nous avons proposé pour notre score *ProfDiv*, mais qui s'applique également à d'autres fonctions de score diversifiées. En Section 4.4, nous comparons nos algorithmes avec l'algorithme de *top-k* présenté au chapitre précédent et nous montrons le bénéfice de nos optimisations. La Section 4.5 se concentre sur les travaux connexes. Enfin, en Section 4.6, la conclusion et une présentation des perspectives de recherche.

4.2 Concepts de base et définition du problème

Dans cette section, les bases nécessaires pour délimiter le problème sont introduites.

Dans le contexte de la recherche d'information, nous avons présenté, au chapitre précédent, une approche hybride combinant *filtrage basé sur les contenus* [126] et *filtrage collaboratif* [61] où les profils utilisateurs - ou intérêts des

utilisateurs - sont définis en fonction des objets $I_i = \{it_1, ..., it_m\}$ qu'ils partagent. Ainsi, nous disposons d'un ensemble d'utilisateurs $U = \{u_1, ..., u_n\}$. Un objet *it* peut être partagé par 1 à n d'entre eux. Nous exploitons la recherche de recommandations dans le sens où chaque utilisateur peut soumettre des requêtes pour rechercher ses résultats parmi les objets recommandés et partagés par des utilisateurs similaires.

Un objet, ou contenu, est représenté de manière vectorielle [107, 141]. En utilisant $tf \times idf$, un objet est exprimé comme une liste de mots clés $k_1, ..., k_z$, et le vecteur représente le poids de chacun de ces derniers pour l'objet en question, étant donné le corpus global. Le profil d'un utilisateur exprime ses intérêts et est calculé à partir des objets qu'il partage I_u. De manière plus précise, le profil d'un utilisateur est la moyenne des vecteurs $tf \times idf$ des objets qu'il partage. Les requêtes sont exprimées, quant à elles, par une liste de mots clés $k_1, ..., k_t$.

L'algorithme 4 du Chapitre 3 décrit la méthode de *top-k* que nous utilisons. Il s'agit d'un *top-k* classique (tel que décrit au Chapitre 2), auquel s'ajoute la notion de liste des candidats. Celle-ci regroupe l'ensemble des objets accédés dans les listes inversées qui n'ont pas encore été insérés dans la liste de résultats R^q. Elle est utile pour le calcul des scores diversifiés de ces objets [1].

Définition du problème : étant donné un ensemble d'utilisateurs U, un ensemble d'objets I et une requête à mots clés q soumise par un utilisateur $u \in U$, le problème est le suivant : il s'agit de recommander efficacement à u les *top-k* objets les plus pertinents et divers $R^q \in I$, étant donné une fonction de score diversifiée. On suppose que ces k objets sont triés par ordre décroissant de score dans la liste R^q.

4.3 Optimisations

Dans cette section, nous proposons un ensemble de techniques d'optimisation améliorant les performances de notre approche, proposée dans le chapitre précédent. Nous simplifions tout d'abord notre modèle de score afin de réduire sa complexité de calcul en Section 4.3.1. Ensuite, en Section 4.3.2, nous proposons un nouveau seuil pour les algorithmes de *top-k* diversifiés, afin de réduire le nombre d'accès aux listes inversées. La Section 4.3.3, présente deux techniques utilisées pour limiter le nombre d'éléments présents dans la liste des candidats et donc le nombre de scores diversifiés à calculer. La Section 4.3.4, introduit deux nouveaux scores d'indexation (*i.e.* le score utilisé pour trier les éléments dans les listes inversées) prenant en compte des éléments de diversification. Enfin, en Section 4.3.5, nous proposons une méthode d'indexation adaptative pour réduire le nombre d'accès aux index, en fonction de toutes les requêtes soumises au système.

1. Pour plus de détails quant au fonctionnement de l'algorithme, se référer au Chapitre 3

4.3.1 Simplification du score de diversification

Calculer le score de diversification des profils, en utilisant l'Équation 3.4, a une complexité de $\mathcal{O}(n^2)$, où n est le nombre total d'utilisateurs, comme cela a été montré en Section 3.4.2. Cette complexité est trop haute, notamment pour le cas d'applications en ligne. Ce score étant personnalisé, il ne peut pas être pré-calculé [9, 12]. Pour cela, nous proposons une adaptation simplifiée de celui-ci.

L'intuition derrière cette simplification est la suivante : l'ensemble des profils des utilisateurs partageant un objet est agrégé afin de simplifier le calcul de la diversité des profils. Ainsi, ce dernier score peut être calculé efficacement sans avoir à considérer chaque utilisateur partageant les objets séparément.

Dans cette approche, chaque objet est maintenant associé à un double vecteur, le premier représentant son contenu et le second l'agrégation des profils des utilisateurs le partageant. Étant donné $profile(u)$ le vecteur $tf \times idf$ des étiquettes soumises par l'utilisateur u (*i.e.* le profil vectoriel de u) et u_{it} l'ensemble des utilisateurs partageant it, la fonction d'agrégation f est la suivante :

$$f(it) = \frac{\sum_{u \in u_{it}} profile(u)}{|u_{it}|} \tag{4.1}$$

Puisque $f(it)$ retourne le même vecteur quel que soit l'utilisateur soumettant la requête q, il peut maintenant être pré-calculé. En utilisant cette représentation d'un objet it, le calcul du score de diversification des profils a maintenant une complexité de $\mathcal{O}(k)$:

$$div_p(it|\{it_1, ..., it_{i-1}\}) = rel_p(u, f(it), q) \times \prod_{it_j \in \{it_1, ..., it_{i-1}\}} 1 - red(f(it)|f(it_j)) \tag{4.2}$$

Où $rel(u, f(it), q)$ est la pertinence des utilisateurs partageant it étant donné u et sa requête q, et $red(f(it)|f(it_j))$ est la redondance de ces mêmes utilisateurs étant donné ceux partageant it_j. Maintenant, le score de diversification complet peut s'exprimer de manière générique :

$$score(it, u, q) = rel_{c\&p}(it, u, q) \times div_{c\&p}(it|\{it_1, ..., it_{i-1}\}) \tag{4.3}$$

Où $rel_{c\&p}(it, u, q)$ prend en compte la pertinence de it_i par rapport à q et aux utilisateurs partageant l'objet et $div_{c\&p}$ est la diversification des objets en prenant en compte à la fois le vecteur de contenu et celui des profils étant donné les éléments déjà insérés dans R^q.

La Figure 4.1 illustre le gain qu'une simplification de notre fonction de score peut entraîner.

4.3.2 Seuil raffiné

Une des principales limitations de l'Algorithme 4 est le nombre d'accès dans les listes inversées nécessaires pour calculer R^q. Comme cela est présenté dans

(a) Calcul de la diversité profil par profil.

(b) Calcul de la diversité sur les profils agrégés.

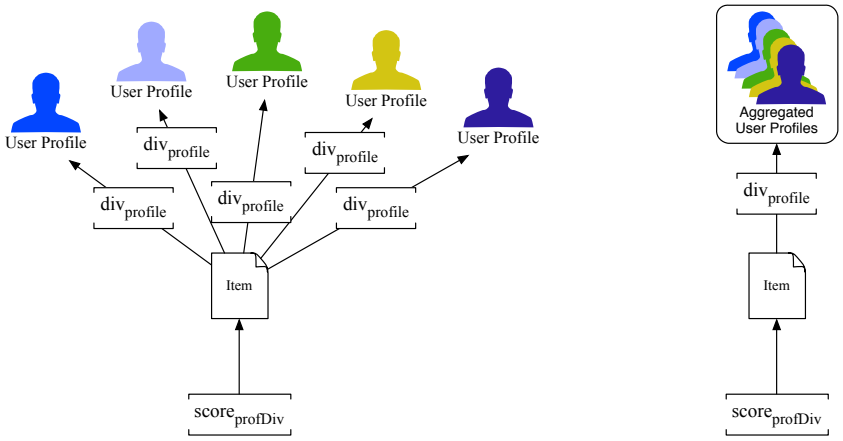


Figure 4.1: Illustration de l'effet de la simplification du score sur son calcul.

l'Équation 3.7, au sein du Chapitre précédent, le seuil δ est évalué en utilisant le score d'indexation de chaque objet dans les listes inversées correspondants aux mots clés $\{k_1, ..., k_t\}$. Dans l'Algorithme 4, div_c et div_p (et $div_{c\&p}$) sont toujours inférieurs à 1. Ainsi, lorsque le nombre d'éléments dans R^q augmente, les scores de diversification des profils et des contenus diminuent pour tous les objets $it \notin R^q$. Par exemple, dans nos expériences, même si tous les objets qui seront retournés à l'utilisateur ont déjà été accédés, l'algorithme continue d'effectuer un nombre élevé d'accès dans les listes inversées.

Nous proposons donc d'utiliser un nouveau seuil δ', déduit de notre fonction de score, afin de limiter le nombre d'accès dans les index :

$$\delta' = f(s_1, ..., s_t) \times f_{div_c}(R^q, \{s_1, ..., s_t\}) \times f_{div_p}(R^q, \{s_1..., s_t\}) \tag{4.4}$$

Notons que pour calculer f_{div_c} et f_{div_p}, nous avons besoin d'informations supplémentaires car les accès $\{s_1, ..., s_t\}$ ne sont pas suffisants. Pour cela, nous introduisons quatre fonctions dont la valeur peut être pré-calculée :

1. $max\_div_c(it, k_i)$: il s'agit du score de diversité des contenus maximum entre un objet it et tout objet après lui dans la liste correspondant au mot clé k_i. ;
2. $max\_div_p(it, k_i)$: il s'agit du score de diversité des profils maximum entre un objet it_i et tout objet après lui dans la liste correspondant au mot clé k_i. ;
3. $max\_trust(k_i)$: c'est le taux de confiance maximum qui peut exister entre la requête et tous les utilisateurs partageant les objets présents dans la liste correspondant au mot k_i ; la partie de ce score de confiance évaluant la similarité entre u et les autres utilisateurs v est égale à 1 ;

4. $max\_pop(k_i)$: il s'agit de la popularité maximale des objets présents dans la liste correspondant au mot k_i.

En utilisant ces fonctions, il est possible de calculer f_{div_c} et f_{div_p} de la manière suivante :

$$f_{div_c}(R^q, \{s_1, ..., s_t\}) = \prod_{it \in R^q} \max_{k_i \in q}(max\_div_c(it, k_i)) \tag{4.5}$$

$$\begin{aligned} f_{div_p}(R^q, \{s_1, s_2, ..., s_t\}) &= \frac{\max_{k_i \in q}(max\_pop(k_i))}{N} \\ &\times \max_{k_i \in q}(max\_trust(k_i)) \times \prod_{it \in \{it_1, ..., it_i - 1\}} \max_{k_i \in q}(max\_div_p(it, k_i)) \end{aligned} \tag{4.6}$$

Lemma 4.3.1. *La diversité des contenus d'un objet it est forcément inférieure ou égale à f_{div_c}.*

Lemma 4.3.2. *La diversité des profils d'un objet it est forcément inférieure ou égale à f_{div_p}.*

La preuve de ces lemmes peut être faite simplement, via l'utilisation d'inégalités entre les seuils et notre fonction de score ; il suffit de montrer que chaque élément du seuil est forcément supérieur à son équivalent dans le score.

Notons également que

$$\prod_{it \in \{it_1, ..., it_i - 1\}} \max_{k_i \in q}(max\_div_c(it, k_i))$$

et

$$\prod_{it \in \{it_i, ..., it_i - 1\}} \max_{k_i \in q}(max\_div_p(it, k_i))$$

peuvent être calculés incrémentalement à chaque fois qu'un objet est ajouté à R^q.

Afin d'expliquer les bénéfices apportés par l'utilisation d'un seuil raffiné, nous présentons un exemple. Dans un souci de clarté, nous simplifions la fonction de score de l'Algorithme 4 en retirant l'indice de confiance rel_{trust} et celui de popularité $\frac{1}{N}$ de div_p :

$$div_p(it | \{it_1, ..., it_{i-1}\}) = \sum_{v_n \in u_{it}} \left[\prod_{v_m \in \{u_{it_1}, ..., u_{it_{i-1}}\}} (1 - red_p(v_m | v_n) \right] \tag{4.7}$$

Retirer $\frac{1}{N}$ et rel_{trust} de la fonction de score permet d'obtenir un seuil δ'' évidemment plus simple comparé à δ', mais offrant le même comportement :

$$\delta'' = \delta \times \prod_{it \in R^q} \max_{k_i \in q}(max\_div_c(it, k_i)) \times \prod_{it \in R^q} \max_{k_i \in q}(max\_div_p(it, k_i)) \tag{4.8}$$

Le Tableau 4.1 illustre cet exemple d'utilisation de l'Algorithme 4 pour construire

Table 4.1: Exemple de l'effet du seuil sur le nombre d'accès triés.

Step	Sorted Access	$rel(it, q)$	Max div_c	Max div_p	Final Score	δ	δ'	R^q	C
1	it_a	0.90	0.85	0.45	0.9	0.9	0.345	it_a	-
2	it_b	0.88	0.84	0.46	0.238	0.88	0.34	it_a	it_b
3	it_c	0.87	0.95	0.65	0.34	0.87	0.33	it_a, it_c	it_b
...									
n	it_z	0.55			0.0023	0.55			

R^q en utilisant δ''. Le nombre d'accès dans les index aurait été largement supérieur en utilisant δ. L'entrée de l'algorithme est une liste inversée d'objets, triés par pertinence décroissante – on suppose que la requête ne contenait qu'un seul mot clé. La première colonne indique le numéro de l'étape qui correspond à une itération complète de l'Algorithme 4 (ligne 3 à 9). La seconde colonne, *sorted accessed*, présente l'objet récupéré lors du dernier accès trié (ligne 3 de l'Algorithme 4). Les colonnes *max* div_c et *max* div_p font référence à la diversité maximale des contenus et des profils de l'objet sur lequel l'accès trié a été effectué, en tenant compte de la liste inversée, fournie en entrée de l'algorithme. R^q est la liste de résultats et C celle des candidats. Les colonnes δ et δ'' montrent les valeurs des seuils à chaque étape.

À l'étape 1, l'Algorithme 4 effectue un accès trié sur it_a. Puisqu'il s'agit du premier objet, son score de diversification est égal à 1 et son score final est $rel(it, q) = 0.9$.
Au cours de la seconde étape, l'Algorithme 4 effectue un accès trié sur it_b. Son score final est 0.238 du fait de sa diversification par rapport à it_a. Notons que δ'' (dont la valeur est inférieure à celle de δ) a une valeur de 0.34, supérieure au score de it_b. Il est encore possible de trouver un meilleur objet dans les index.
Durant la troisième étape, l'Algorithme 4 récupère l'objet it_c dont le score final (en tenant compte de it_a) est 0.34. Ce dernier est supérieur ou égal à δ" et it_c peut donc être inséré dans R^q. Notons que δ est égal à 0.87 et n'aurait pas permis l'insertion de it_c dans R^q. Qui plus est, à l'étape n, la valeur de δ est de 0.55, c'est-à-dire toujours supérieure à it_c et l'algorithme continuerait d'itérer. En utilisant δ'', ce dernier a pu s'interrompre après trois accès aux listes inversées.

Notre nouveau seuil permet ainsi de limiter le nombre d'accès dans les index.

4.3.3 Limiter la taille de la liste des candidats

Le calcul du score de diversification des objets dans la liste des candidats (Algorithme 4, en page 68, ligne 8) représente une grosse proportion du temps de réponse. Dans cette section, nous proposons deux stratégies permettant d'en limiter la taille. Tout d'abord, la Section 4.3.3 montre comment fixer une taille

maximale, puis, la Section 4.3.3, introduit une approche permettant d'en limiter la taille en filtrant les objets en fonction d'une limite dynamique.

Liste des candidats avec taille maximale

Afin de réduire simplement le nombre de scores de diversification à calculer, il est possible de fixer une taille maximale N à la liste des candidats. À cette fin, les candidats C sont triés par score décroissant. Puis, chaque fois que la taille dépasse N, l'algorithme retire les éléments en fin de liste (*i.e.* ceux qui ont le score minimum).

Pour cela, nous modifions deux fonctions de l'Algorithme 4. Tout d'abord, la méthode `add`, après avoir inséré un élément dans C, doit préserver les candidats triés par score décroissant et retirer le dernier si la taille excède N. La méthode `update`, quant à elle, doit re-trier les candidats à chaque fois qu'elle en modifie le score.

Bien que ces méthodes ajoutent un calcul supplémentaire, le fait d'éliminer certains candidats permet de réduire considérablement le temps de réponse de l'Algorithme 4.

Liste des candidats filtrée

Dans cette sous-section, nous proposons une méthode permettant de filtrer dynamiquement les candidats en exploitant un seuil. Définir ce dernier de manière statique n'a pas de sens puisque le score de diversification des objets dans la liste des candidats décroît au fur et à mesure que la taille de R^q augmente. L'idée est donc de définir un seuil proportionnel au score du meilleur candidat présent dans la liste des candidats C. Avec un ratio de 0.1, tous les objets dont le score est inférieur à 10% de celui du meilleur candidat seront ainsi retirés de la liste.

Afin d'implémenter cette approche, nous avons modifié les méthodes `add` et `update` : Algorithmes 6 et 7 respectivement.

Algorithme 6: Add an *item* to Candidate List

Input: it, C, $ratio$, $best$
Output: it is added to the candidate list C if its score is higher to $ratio \times score(best)$, and $best$ is updated if needed

```
1 if score(it) > ratio × score(best) then
2     add it to C;
3     if score(it) > score(best) then
4         best = it;
```

Comme cela est montré en Section 3.5, notre approche de filtrage permet de réduire le nombre de scores diversifiés à calculer tout en préservant une bonne

Algorithme 7: Update Diversification Scores of Candidate List

Input: C, it, $ratio$, $best$
Output: C's items have their scores updated wrt. it, and C keeps only the items whose score is less than $ratio \times score(best)$

1 **Update** the score of $\forall i \in C$ **wrt.** it and **compute** best;
`/* recall that the best candidate has just been added in the results list...` *best* `is no longer up to date */`
2 **Remove** $\forall items$, such that $score(it) < ratio \times score(best)$;

qualité des résultats (*i.e.* les éléments initialement retournés à l'utilisateur le sont toujours après le filtrage).

4.3.4 Indexation diversifiée

Si le premier objet ajouté à la liste R^q est redondant par rapport à tous les autres, alors le nombre d'accès nécessaires pour la construire s'accroît. En effet, si le premier objet est redondant, alors le score du dernier objet de la liste de résultats sera inférieur et le nombre d'accès nécessaires pour satisfaire la condition de seuil supérieur. Dans cette section, nous proposons une méthode qui permet de décroître le score d'indexation de certains objets très redondants en leur attribuant une pondération faible. Ainsi, les chances qu'ils apparaissent dans la liste de résultats sont réduites.

Dans notre approche, cette pondération se calcule comme suit :

$$weight(it, items) = \frac{\sum_{it_j \in items} sim(it_j, it)}{|items|} \tag{4.9}$$

En d'autres termes, la pondération d'un objet it est calculée en fonction d'un ensemble d'objets, noté $items$. Ces derniers sont soit tous les objets existants, soit uniquement ceux qui sont présents dans la liste inversée courante. Ainsi, le score d'indexation d'un objet dans une liste inversée est calculé de la manière suivante :

$$indexScore(it, L_k/I) = rel(it, k) \times \frac{\sum_{it_j \in L_k/I} sim(it, it_j)^{\mu}}{|L_k/I|} \tag{4.10}$$

Où μ est un paramètre permettant d'évaluer l'importance de cette pondération. L_k/I indique que l'ensemble $items$ est soit celui des objets présents dans la liste inversée L_k, soit l'ensemble total des objets I. Si un objet est très pertinent mais également très redondant par rapport aux autres, son score sera alors réduit. Cette approche a néanmoins un défaut : certains éléments sont définitivement éliminés.

4.3.5 Indexation diversifiée adaptative

Dans cette section, nous proposons une technique qui adapte le score d'indexation des objets en fonction des requêtes soumises au système.

Étant donné une liste inversée et une requête q, nous identifions trois catégories d'objets :

1. les objets pertinents et divers qui sont accédés dans les index et retournés à l'utilisateur ;
2. les objets pertinents et redondants qui sont eux aussi accédés mais qui ne sont pas renvoyés à l'utilisateur ;
3. les objets non pertinents qui ne sont pas accédés dans les index.

Nous exploitons l'idée que certains objets seront toujours dans la seconde catégorie étant donné les requêtes soumises par les utilisateurs. Ces objets vont provoquer des calculs supplémentaires pour produire R^q, mais n'amélioreront jamais sa qualité puisqu'ils ne seront pas retournés à l'utilisateur.

L'intuition est d'indexer chaque objet en tenant compte de leur pertinence par rapport au mot clé de la liste inversée mais aussi d'un poids indiquant la fréquence à laquelle l'objet a été accédé dans cette liste mais n'a pas été retourné à l'utilisateur.

Cette approche est dite adaptative dans le sens où le poids dépend de toutes les requêtes soumises. Chaque liste inversée est ensuite mise à jour dynamiquement lorsqu'une requête est exécutée, comme cela est présenté dans la Figure 4.2. Étant

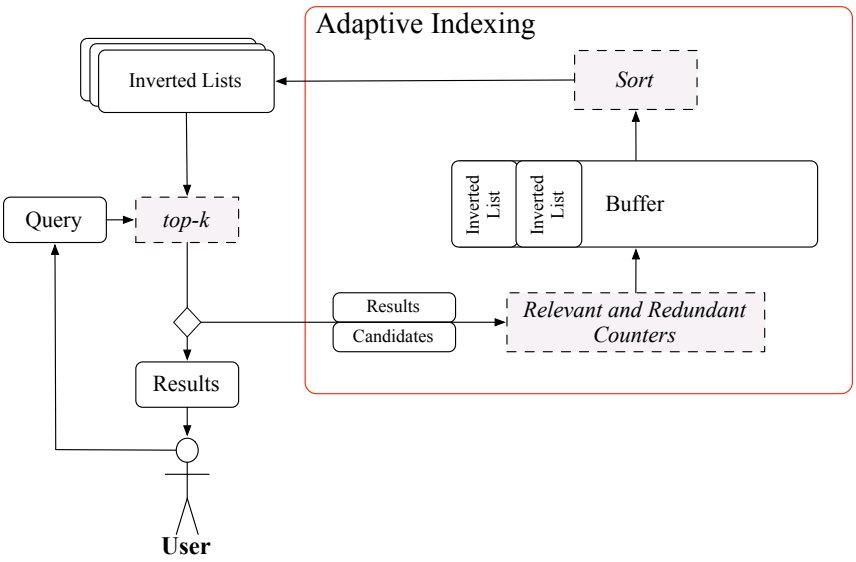


Figure 4.2: Architecture de l'indexation diversifiée adaptative.

donné une liste inversée L_k, un objet $it \in L_k$ est associé à deux compteurs, l'un

indiquant le nombre de fois où il a été pertinent et le second le nombre de fois où il a été redondant.

À la fin de l'exécution d'une requête (cf. Algorithme 4), deux listes d'objets existent : l'une, notée R^q, regroupant les résultats et l'autre, notée C^q, les candidats. Chaque objet de $it \in R^q$ voit son compteur de *pertinence* augmenter, alors que ceux de C^q voient celui de *redondance* accru. Notons qu'un objet ne peut pas appartenir aux deux catégories simultanément. En utilisant ces compteurs, la fréquence à laquelle un objet est considéré comme redondant se calcule comme suit :

$$f_{red,L_k}(it) = \frac{|it_{redundant}|}{|it_{redundant}| + |it_{relevant}|} \tag{4.11}$$

Où $|it_{redundant}|$ est initialisé à 0 et $|it_{relevant}|$ à 1. Dans chaque liste inversée, les objets sont triés en fonction du score suivant :

$$indexScore(it, L_k) = rel(it, L_k) \times W(f_{red,L_k}(it)) \tag{4.12}$$

Où $W(x)$ est une fonction décroissante continue avec les propriétés suivantes :

$$\begin{gathered} W(0) \simeq 1 \\ \lim_{x \to 1} W(x) \to w_{min} \end{gathered} \tag{4.13}$$

Où $0 < w_{min} < 1$. En d'autres termes, plus un objet sera redondant, plus son poids et donc son score d'indexation seront faibles. Ainsi, un élément systématiquement redondant sera poussé de plus en plus loin dans la liste inversée et sera de moins en moins accédé, permettant ainsi un meilleur temps de réponse. La fonction $W(x)$ possède un minimum défini par le système, noté w_{min} supérieur à 0, impliquant qu'aucun objet ne peut être strictement retiré de l'index.

Dans notre approche, $W(x)$ est une fonction sigmoïde [66] et est donc caractérisée par deux états stables (asymptotes) au début et à la fin (dans notre cas $y = 1$ et $y = w_{min}$) et une phase transitoire reliant les deux premiers états, comme cela est présenté dans la Figure 4.3. La fonction de pondération $W(x)$ est définie comme suit :

$$W(x) = \frac{1 - w_{min}}{1 + e^{a \times (x-b)}} + w_{min} \tag{4.14}$$

Où w_{min} est le pire poids qu'un objet puisse recevoir, b permet de décaler l'état transitoire et a de modifier sa durée. La variable x est la fréquence avec laquelle l'objet est considéré comme redondant.

L'utilisateur doit définir la fréquence de redondance maximale f_{max} ($0 < f_{max} < 1$) telle que tous les objets dont la redondance en est inférieure ont la garantie de ne pas avoir de perte supérieure à l_{max} en termes de score d'indexation, sachant que $l_{max} \to 0$ et $l_{max} > 0$. En d'autres termes, les objets qui sont périodiquement pertinents auront un poids supérieur à $1 - l_{max}$ et un score d'indexation comme suit : $indexScore(item, L_k) = rel(item, L_k) \times W(f_{red,L_k}(item)) \simeq$

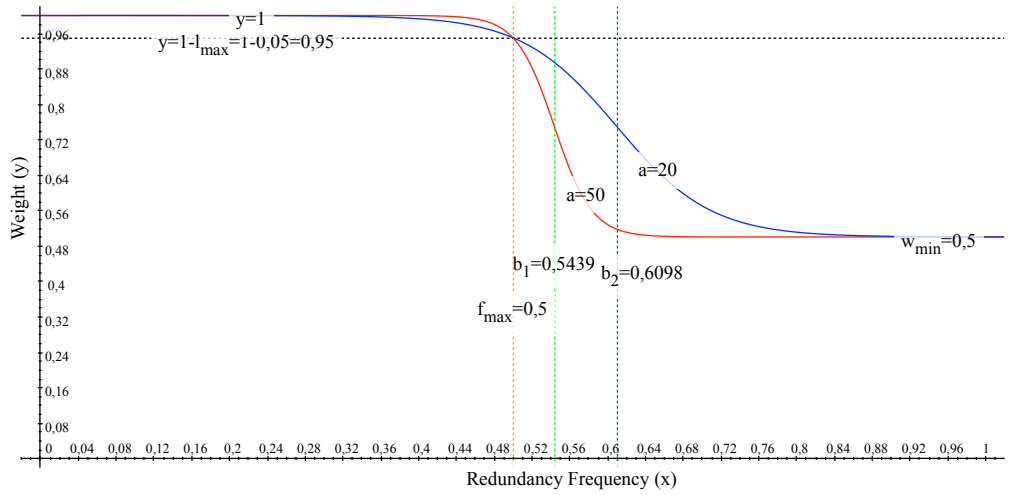


Figure 4.3: Fonction de poids $W(x)$ en fonction de ces divers paramètres.

$rel(item, L_k)$. Cependant, si un objet est trop redondant (sa fréquence est supérieure à f_{max}), son poids, et donc son score d'indexation, commenceront à décroître. Étant donné f_{max} et l_{max}, les variables a et b se définissent comme suit :

$$a \geq \frac{2 \times ln(\frac{l_{max}}{1-l_{max}-w_{min}})}{f_{max} - 1} \tag{4.15}$$

De petites valeurs de a impliquent un état transitoire large et de grosses valeurs de a, un état court. Le paramètre b s'exprime en fonction de a :

$$b = f_{max} - \frac{ln(\frac{l_{max}}{1-l_{max}-w_{min}})}{a} \tag{4.16}$$

Puisque tous les objets auront leur score réduit du fait de $W(x)$ – incluant ceux qui sont pertinents –, le seuil δ risque de sous-estimer la valeur de certains objets et la condition de seuil risque d'être satisfaite alors que des objets pertinents n'ont pas encore été accédés. Ce comportement est désiré pour les objets redondants mais ne l'est pas pour ceux qui sont divers. En fonction du paramètre utilisateur f_{max}, nous savons que tous les objets dont la fréquence de redondance en est inférieure ne devraient pas être manqués. Sachant que $W(f_{max}) = 1 - l_{max}$, il est possible de définir le seuil suivant :

$$f'(s_1, ..., s_t) = \frac{f(s_1, ..., s_t)}{1 - l_{max}} \tag{4.17}$$

Où s_i est le dernier accès trié dans la liste correspondant au mot k_i.

Étant donné les définitions du seuil et de la fonction de poids $W(x)$, il est possible de définir les deux garanties suivantes en termes de qualité des résultats et de performance de l'algorithme :

Théorème 1 (Garantie de qualité)**.**

Étant donné une liste inversée L_k, si une requête q représente $h \times 100\%$ des requêtes accédant L_k, et si $h \geq 1 - f_{max}$, alors les résultats de q ont la garantie d'être identiques à l'algorithme utilisant un index statique.

Théorème 2 (Garantie de performance)**.**

Étant donné l_{max}, une requête q, $nbSA_a$ et $nbSA_b$ le nombre d'accès nécessaires pour calculer q, en utilisant un index statique et un index adaptatif, si q représente $h \times 100\%$ des requêtes accédant L_k, et si $h \geq 1 - f_{max}$, alors la probabilité que l'algorithme utilisant l'index adaptatif soit meilleur que celui utilisant un index statique tend vers 1 lorsque l_{max} tend vers 0 :

$$\lim_{l_{max} \to 0} \mathcal{P}(nbSA_a < nbSA_b) = 1 \tag{4.18}$$

Ces deux théorèmes sont prouvés dans l'Annexe B.

4.4 Évaluation expérimentale

Cette section présente l'évaluation expérimentale de nos approches afin de valider la performance des algorithmes. Nous avons mené nos expériences via l'utilisation d'un jeu de données provenant de *Delicious*, d'une version enrichie de ce dernier, ainsi que d'un autre jeu issu de *Flickr*. La Section 4.4.1 décrit la mise en place des expériences, puis, la Section 4.4.2 discute des résultats.

4.4.1 Mise en place des expériences

Nos expériences ont été menées sur trois jeux de données différents. Il s'agit tout d'abord de 30 000 signets téléchargés de *Delicious*, associés à environ 55 000 étiquettes uniques soumises par 2 000 utilisateurs. Les signets précédents ont été enrichis de leur contenu `html` afin de réaliser un second jeu de données : cela permet d'observer le comportement de nos algorithmes lorsque chaque objet est associé à peu de mots clés ou, inversement, à un grand nombre. Enfin, le troisième jeu de données est composé de 3.25 millions d'images récupérées du site *Flickr* et associées à 3.5 millions groupes d'étiquettes soumis par environ 272 000 utilisateurs ; chaque groupe est en réalité une liste d'une à quinze étiquettes. L'ensemble des mots clés disponibles (*i.e.* titre, étiquettes, description ou contenu `html`) est utilisé pour l'indexation.

Une expérience consiste à analyser les performances des algorithmes lorsque des requêtes sont soumises. Ces dernières sont soit créées automatiquement, soit directement soumises par les utilisateurs lors de *sondages*. Dans le premier cas, elles sont construites comme l'association des étiquettes soumises par un utilisateur unique sur une seule image en un jour ; étant donné que la personnalisation est prise en compte, le profil de ce même utilisateur sera utilisé lorsque la requête sera soumise.

Le code des expériences est développé en *C++* et est exécuté sur une machine *core-2-duo 2.5Ghz* avec *8GB* de *RAM DDR3*.

Évaluation de la performance des algorithmes

Afin d'évaluer les performances de nos solutions, les techniques suivantes ont été comparées :

1. `Seuil raffiné` : le seuil raffiné prenant en compte diverses informations comme cela est présenté en Section 4.3.2.
2. `Liste des candidats avec une taille maximale` : la taille de la liste des candidats est limitée à une valeur pré-définie par le système comme présenté en Section 4.3.3.
3. `Liste des candidats filtrée` : les candidats sont filtrés à partir d'un seuil dynamique, comme présenté en Section 4.3.3.
4. `Indexation diversifiée globalement` : le score d'indexation de chaque objet dépend de statistiques de redondance globale comme présenté en Section 4.3.4.
5. `Indexation diversifiée localement` : le score d'indexation de chaque objet dépend de statistiques de redondance locale à une liste inversée comme présenté en Section 4.3.4.
6. `Indexation diversifiée adaptative` : le score d'indexation est mis à jour dynamiquement en fonction des requêtes soumises au système comme décrit en Section 4.3.5. Cette expérience est exécutée deux fois : la première avec $f_{max} = 99\%$ et la seconde avec $f_{max} = 99,9\%$.

Les métriques suivantes sont évaluées :

1. temps de réponse lorsque les listes inversées sont en mémoire ;
2. temps de réponse lorsque les listes inversées sont sur disque.

Afin de simuler les accès disque, nous avons procédé comme suit. Tout d'abord, les index ont été matérialisés de manière à ce qu'un accès à un objet d'une liste inversée permette de charger tout le vecteur de ce dernier. Ainsi, nous évitons les accès supplémentaires dus au calcul de la diversification entre deux objets. Nous supposons que chaque vecteur a une taille normalisée et que sa taille en mémoire est de $1KB$. Un accès sur le disque requiert une opération de recherche, ou *seek*,

Operation	Characteristic
Seek Operation	duration : $9ms$
Data transfer	speed : $1Gb/s$
Number of blocs transferred per access	$64(\times 1KB)$

Table 4.2: Caractéristiques d'un accès trié sur le disque.

puis une opération de transfert. Enfin, chaque fois que l'algorithme effectue un accès sur le disque, plusieurs blocs sont transférés afin d'éviter de futurs *seek*. Nous avons utilisé les paramètres présentés dans le Tableau 4.2.

Notons que certaines propositions d'optimisation doivent être configurées (*e.g.* la taille maximale de la liste des candidats). Dans nos expériences, ces paramètres de configuration sont définis de manière à ce qu'ils permettent d'éviter une perte de qualité dans la liste de résultats tout en offrant les meilleures performances possibles.

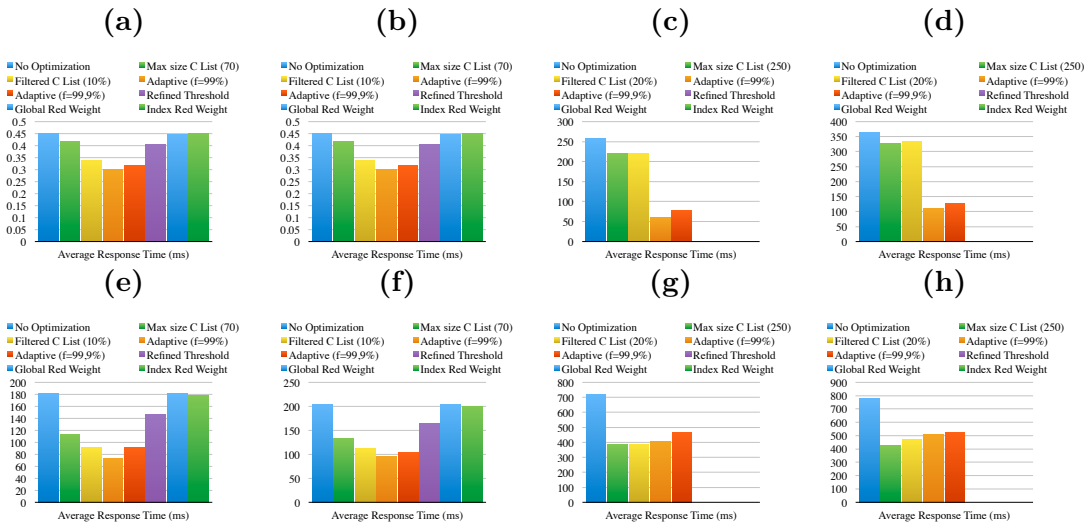


Figure 4.4: Évaluation des performances de plusieurs optimisations en utilisant la diversité des profils : (a) jeu de données *Flickr*, 5 000 photos en mémoire, (b) jeu de données *Flickr*, 5 000 photos sur le disque, (c) jeu de données *Flickr*, 3 250 000 photos en mémoire, (d) jeu de données *Flickr*, 3 250 000 photos sur le disque, (e) jeu de données *Delicious*, 5 000 signets en mémoire, (f) jeu de données *Delicious*, 5 000 signets sur le disque, (g) jeu de données *Delicious*, 3 250 000 signets en mémoire, (h) jeu de données *Delicious*, 3 250 000 signets sur le disque.

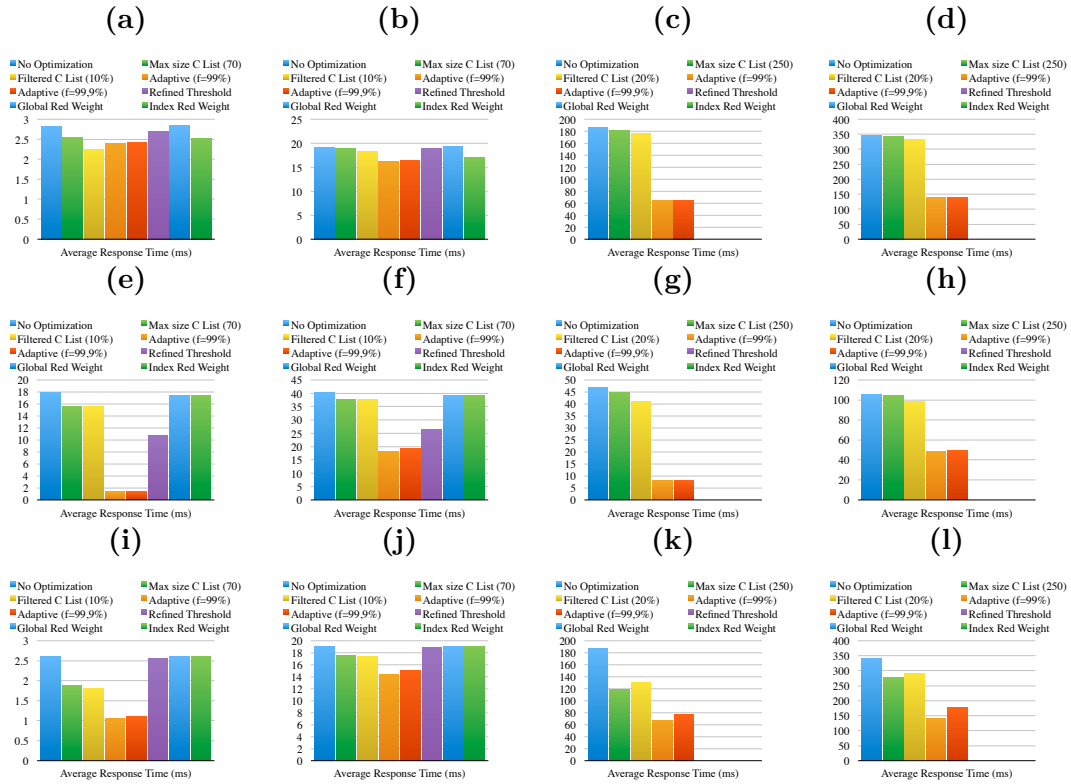


Figure 4.5: Évaluation des performances de plusieurs optimisations en utilisant la diversité des contenus : (a) jeu de données *Flickr*, 5 000 photos en mémoire, (b) jeu de données *Flickr*, 5 000 photos sur le disque, (c) jeu de données *Flickr*, 3 250 000 photos en mémoire, (d) jeu de données *Flickr*, 3 250 000 photos sur le disque, (e) jeu de données *Delicious*, 5 000 signets en mémoire, (f) jeu de données *Delicious*, 5 000 signets sur le disque, (g) jeu de données *Delicious*, 3 250 000 signets en mémoire, (h)jeu de données *Delicious*, 3 250 000 signets sur le disque, (i) jeu de données *Delicious* avec contenu `html` signets en mémoire, 5 000 signets en mémoire, (j) jeu de données *Delicious* avec contenu `html`, 5 000 signets sur le disque, (k) jeu de données *Delicious* avec contenu `html`, 3 250 000 signets en mémoire, (l) jeu de données *Delicious* avec contenu `html`, 3 250 000 signets sur le disque.

4.4.2 Résultats expérimentaux

Les résultats des mesures de performances relatives aux optimisations présentées en Section 4.3 sont évoqués ici. Dans ces expériences, le score de confiance est fixé à 1. La Figure 4.4 présente les résultats lorsque la fonction de score prend en compte la diversification des profils (simplifiée comme cela a été présenté en Section 4.3.1), et la Figure 4.5 expose les résultats lorsque la diversité des contenus est prise en compte.

Notons que chaque expérience est exécutée deux fois : l'une à petite échelle (*i.e.* 5 000 objets indexés) et l'autre à grande échelle (*i.e.* tous les objets sont indexés).

L'*indexation diversifiée adaptative* a, en moyenne, les meilleurs gains en termes de temps de réponse. En effet, lorsque la diversité des profils est prise en compte, la Figure 4.4 montre, qu'à petite échelle, *adaptive* 99% et 99, 9% sont environ 33% plus rapides sur le jeu de données *Flickr*, et plus de 2 fois plus rapides sur le jeu de données *Delicious* lorsque les index sont en mémoire. Lorsque ces derniers sont sur disque, le gain est inférieur à 3% sur *Flickr* mais trois fois plus rapides sur *Delicious*.

À grande échelle, les gains sont supérieurs. Les méthodes *Adaptative* 99% et 99, 9% sont plus de 4 fois plus rapides lorsque les index sont en mémoire, et de 1.47 à 3 fois plus rapides lorsqu'ils sont sur disque.

En prenant en compte la diversité des contenus, la Figure 4.5 montre que *adaptive* 99% et 99, 9% sont de 15% à plus de 12 fois plus rapides sur *Flickr* et *Delicious* respectivement lorsque les index sont en mémoire et de 15% à plus de deux fois plus rapides lorsqu'ils sont sur disque.

À grande échelle, les gains sont à nouveau supérieurs. Les méthodes *Adaptative* 99% et 99, 9% sont au moins deux fois plus rapides et le gain peut aller jusqu'à un facteur de 5.

Les méthodes adaptative 99% et 99, 9% entraînent cependant une légère perte de qualité qui est de 0.704% et 0.644% en termes de pertinence à petite échelle et de 1.02% et 0.82% à grande échelle. Notons que la diversification est accrue lorsque la pertinence est diminuée. En d'autres termes, les pertes sont minimes et ne concernent que des requêtes dont la fréquence d'accès à un index est inférieure à 0.1%. Notons qu'un index adaptatif pourrait être combiné à un index statique. Ainsi, lorsqu'une requête serait soumise, le choix de l'index se ferait dynamiquement en fonction des statistiques (*e.g.* fréquence), et les pertes seraient nulles.

Les méthodes de diversifications statiques d'index (*i.e.* globale ou locale à une liste) montrent de mauvais résultats en termes de performance. En effet, le gain n'excède pas 2% et se situe souvent autour de 0%. Qui plus est, la méthode de diversification locale à une liste entraîne une perte, en termes de pertinence, de 30% en moyenne.

Les gains du seuil raffiné sont très dépendants du jeu de données. Ainsi, ils sont proches de 0 sur le jeu de données *Flickr* mais dépassent les 20% sur *Delicious*. À cela s'ajoute le fait que la méthode n'entraîne aucune perte de qualité puisqu'elle ne fait que raffiner le seuil.

Enfin, limiter la taille de la liste des candidats en la bornant ou en filtrant permet d'obtenir des gains de quelques pour-cents à un facteur de deux. Lorsque les index sont sur disque, le gain est réduit et passe d'environ 2% à plus de 40%.

4.5 Travaux connexes

De nombreux travaux sur la diversité proposent des optimisations pour améliorer l'efficacité du calcul d'un ensemble de résultats divers.

Yu et al [176] proposent deux algorithmes pour calculer efficacement un *top-k*, dont les résultats sont composés d'objets pertinents et divers. Le premier d'entre eux, *Swap*, procède en deux étapes. Tout d'abord, les k objets les plus pertinents sont calculés. Ensuite, itérativement, l'objet de la liste de k éléments le plus redondant en est retiré pour être remplacé par l'élément le plus pertinent suivant (*e.g.* $k+1$, $k+2$, etc.), si ce dernier contribue à accroître la diversité. L'algorithme s'arrête lorsqu'un seuil indique que la perte de pertinence de la liste de résultats est trop importante. Le second algorithme, dit *glouton*, sélectionne itérativement les résultats les plus pertinents et ne les ajoute dans la liste de k éléments que si leur diversité est suffisante, étant donné un seuil. Le principal désavantage de ces solutions réside dans la définition du seuil, qui est arbitraire et peut varier suivant les utilisateurs.

Drosou et Pitoura [50] adoptent une approche différente de la diversité basée sur l'idée de *radius* (cf. Chapitre 2). Ici, les résultats les plus pertinents sont indexés dans une structure d'arbre de type *m-tree*, où les objets sont reliés en fonction de leur diversité. Cette structure oblige à indexer les résultats pertinents ultérieurement à l'exécution de la requête.

Angel et Koudas [17] proposent de focaliser les optimisations sur les accès dans les listes inversées. Ils adoptent une approche de *top-k*. À chaque accès, l'intuition est qu'il peut être ou non nécessaire de récupérer l'ensemble du vecteur décrivant l'objet, ou seulement une sous-partie. Ils décrivent une fonction permettant de choisir à la volée le type d'accès et montrent que si les vecteurs des objets sont grands, il est préférable de les récupérer en entier, ce qui n'est pas le cas si ils sont petits.

Dans [1], Abbar *et al.* proposent une technique afin de recommander des articles proches de l'article courant (*i.e.* lu par l'utilisateur), mais divers. L'algorithme de diversification s'appuie sur le hachage (*i.e. hashing*) *dLSH** et la requête est en réalité l'article lui-même. Lors de l'étape d'indexation par *hashing*, les articles sont regroupés par paniers en fonction de leur similarité (*i.e. LSH* signifiant *Locality Sensitive Hashing*), mais également suivant des critères de diversification (*i.e. dLSH* signifiant *diverse Locality Sensitive Hashing*). Il suffit donc de récupérer les paniers associés via le *hashing* de l'article courant afin de récupérer un sous-ensemble d'articles pertinents, et déjà divers. Le résultat final se calcule donc sur un sous-ensemble réduit extrait lors de l'opération de *hashing*.

Dans ce chapitre, nous nous concentrons sur d'autres aspects et essayons de limiter la difficulté du choix lorsque des seuils sont utilisés. Par exemple, le seuil permettant de limiter la taille de la liste des candidats est fixé comme un ratio, et même de larges valeurs de ce dernier permettent d'accroître les performances.

La méthode adaptative, de son côté, garantit des minimums de performances et de qualités en fonction d'un paramètre f_{max}. Ainsi, lorsque l'utilisateur choisit ce dernier, il est totalement conscient de son effet sur la qualité, ce qui limite l'arbitraire de ce choix.

4.6 Conclusion

Dans ce chapitre a été exposé un ensemble de techniques d'optimisation utiles afin d'accroître les performances de notre solution. Ces dernières se sont focalisées sur trois aspects :

1. simplification du modèle afin de permettre le pré-calcul ;
2. optimisation du seuil et score d'indexation diversifié afin de limiter le nombre d'accès aux listes inversées ;
3. sélection de candidats pour limiter le nombre de scores diversifiés à calculer.

À travers une évaluation expérimentale sur trois jeux de données, nous avons montré que nos optimisations peuvent réduire considérablement le temps de réponse – jusqu'à 12 fois plus petit –, que ce soit pour les scores intégrant la diversité des profils ou uniquement celle des contenus.

Les algorithmes présentés jusque-là sont centralisés. Par la suite, dans les deux prochains chapitres, nous montrons comment déployer la recherche et la recommandation sur une architecture distribuée.

CHAPITRE 5

Diversité dans la recherche et la recommandation distribuées

Résumé. Ce chapitre est consacré aux défis de la recherche et de la recommandation distribuées. Généralement, en recherche et en recommandation distribuées, chaque pair u *construit son propre* cluster *d'utilisateurs pertinents, qui seront ensuite utilisés pour traiter les requêtes soumises par* u*. Cependant, en ne considérant que la notion de pertinence dans la construction du* cluster*, les utilisateurs impliqués dans le traitement des requêtes seront redondants, tout comme les résultats qu'ils retourneront, ce qui limitera le niveau de rappel. Ce chapitre présente la manière dont la diversité combinée à la pertinence (i.e.* usefulness*, ou utilité) permet d'améliorer le voisinage de chaque pair, utilisé dans la recherche et la recommandation et donc les niveaux de rappel ; l'algorithme de* clustering *correspondant y est également introduit. Une stratégie de réplication, permettant d'accroître encore plus les niveaux de rappel, est aussi présentée. Enfin, nous validons notre approche en utilisant trois jeux de données basés sur MovieLens, Flickr et LastFM. En comparaison avec les solutions de l'état de l'art, nous obtenons un gain majeur avec des niveaux de rappel jusqu'à trois fois plus élevés lorsque notre score d'utilité (i.e.* usefulness*) est pris en compte, et ce, quel que soit le score de pertinence initialement utilisé.*

5.1 Introduction

Comme évoqué précédemment, les utilisateurs sont devenus des producteurs massifs de données diverses (*e.g.* photos, vidéos, données scientifiques, observations de botanique) qui peuvent être stockées dans une large variété de systèmes (*e.g.* serveurs centralisés, ordinateur local ou *smartphone*). Ces utilisateurs souhaitent malgré tout partager leurs données avec d'autres ayant des intérêts simi-

laires. Il s'agit donc de déployer les solutions de recherche et recommandation sur une architecture distribuée.

Nous adoptons une approche pair-à-pair de recherche et recommandation car celle-ci fournit des propriétés importantes, telles que le passage à l'échelle, la dynamicité, l'autonomie et le contrôle décentralisé. Chaque utilisateur u stocke les données qu'il souhaite partager sur son nœud local. Puis, étant donné u et une requête à mots clés q, l'objectif de notre approche de recherche et recommandation est de retourner à u l'ensemble des objets pertinents par rapport à q, partagés par d'autres utilisateurs similaires à u. En d'autres termes, nous combinons la recherche et la recommandation dans le sens où u soumet une requête q afin de chercher des objets recommandés par des utilisateurs qui lui sont similaires. Il s'agit, en réalité, de l'approche présentée au Chapitre 3.

La recherche et la recommandation distribuées ont reçu une attention importante [21, 37, 49, 171]. Généralement, chaque utilisateur construit un groupe d'utilisateurs « pertinents » (*e.g.* Jaccard) qu'il a rencontré sur le réseau. On appelle le *User Network* d'un utilisateur u (ou *U-Net*), ce groupe d'utilisateurs pertinents.

Pour construire le *U-Net*, nous nous appuyons sur des protocoles de *gossip* qui ont l'avantage d'être très résistants, évolutifs et convergent rapidement [75], ce qui en fait une bonne alternative pour la recherche et la recommandation distribuées. Avec les protocoles de *gossip*, chaque utilisateur maintient une vue aléatoire du réseau qui évolue dynamiquement (cf. Chapitre 2). Les utilisateurs les plus pertinents, étant donné un score, rencontrés dans cette vue sont ensuite ajoutés au *U-Net* [37, 49, 81, 82, 171].

Lorsqu'un utilisateur u soumet une requête q, celle-ci est transmise à tous les utilisateurs de son *U-Net*, qui re-transmettent q récursivement jusqu'à ce que le nombre de sauts effectués par q atteigne une valeur définie par le système, nommée TTL. Les utilisateurs qui reçoivent la requête q retournent à u leurs résultats les plus pertinents étant donné q et une fonction de score.

Généralement, ce groupe d'utilisateurs, le *U-Net*, est construit étant donné une mesure de similarité (*e.g.* Jaccard). Malheureusement, en ne considérant que la similarité dans ce processus de construction, le *U-Net* se retrouve rempli d'utilisateurs redondants, et lorsqu'un utilisateur u soumet une requête, ceux du *U-Net* qui sont impliqués pour répondre à la requête ont une forte probabilité de retourner les mêmes résultats. Cela aura pour effet de limiter la capacité du système à retrouver l'ensemble des résultats pertinents par rapport aux requêtes.

En *recherche d'information*, l'utilité (*i.e. usefulness*), en combinant pertinence et diversité, permet de limiter la redondance qu'il peut y avoir dans un ensemble d'objets [17, 39]. Dans ce chapitre, nous proposons une approche utilisant les protocoles de *gossip* pour la recherche et la recommandation, introduisant un nouveau score de *regroupement*, appelé *utilité* ou *usefulness*, combinant pertinence et diversité. Attention, cette utilisation de la diversification peut porter

à confusion et nécessite une clarification. Dans notre contexte, nous proposons d'utiliser *l'utilité* (*i.e.* pertinence combinée à la diversité), non pas pour limiter la redondance des résultats, mais bien dans le processus de construction du *U-Net*. Ainsi, les utilisateurs seront beaucoup plus divers et leur couverture sera augmentée – et par extension, les niveaux de rappel (*i.e.* capacité du système à retrouver l'ensemble des résultats pertinents, étant donné les requêtes soumises par les utilisateurs) seront également augmentés.

Comme nous le montrons expérimentalement, ce score permet d'accroître significativement le rappel et donc la qualité des recommandations faites par le système. Cependant, les algorithmes de *regroupement* généralement utilisés en pair-à-pair ne sont pas adaptés à la diversité puisqu'ils sont faits pour maximiser la pertinence uniquement. Nous proposons donc un nouvel algorithme, spécialement conçu pour le score d'*utilité*. Néanmoins, certains profils d'utilisateurs n'intègreront donc pas le *U-Net* car ils sont trop redondants, alors même qu'ils peuvent être pertinents : les objets partagés uniquement par ces utilisateurs redondants ne pourront pas être récupérés. Nous faisons référence à ces objets sous le nom d'*objets manquants*. Pour accroître davantage le rappel, nous introduisons également une approche de réplication afin de compenser l'effet des *objets manquants*.

En résumé, ce chapitre regroupe les contributions suivantes :

1. nous montrons que l'utilité, en combinant pertinence et diversité, est une bonne manière d'accroître le rappel et qu'il devrait être exprimé au travers d'un score de diversification probabiliste [17, 39] ;
2. nous proposons un algorithme de *regroupement* permettant de maintenir un *U-Net* grâce aux protocoles de *gossip*, en utilisant l'*utilité* ;
3. nous proposons une approche de réplication qui, combinée à la diversité, permet de limiter l'effet des *objets manquants* ;
4. nous validons notre approche grâce à une évaluation expérimentale sur trois jeux de données : *MovieLens*, *Flickr* et *LastFM*. Nous observons que la diversité permet un grand gain en termes de rappel quelle que soit la fonction de pertinence utilisée. Comparée à des solutions exposées dans l'état de l'art, la diversité permet d'atteindre des gains jusqu'à facteur trois. Notre proposition de réplication réduit l'effet des *objets manquants* et donc d'obtenir des valeurs de rappel dignes d'un système centralisé.

Le reste de ce chapitre est organisé comme suit. La Section 5.2 introduit un ensemble de concepts de bases nécessaires à la compréhension de ce travail, et présente la définition du problème. En Section 5.3, nous décrivons notre nouveau score de *regroupement* et nous montrons les détails de l'algorithme permettant de maintenir un *U-Net* utile (*i.e. useful*). La Section 5.5 présente l'évaluation expérimentale. Enfin, en Section 5.7, la conclusion et les perspectives de recherche.

5.2 Concepts de base et définition du problème

Cette section introduit les bases nécessaires pour comprendre le problème que nous adressons.

Représentation des objets et des profils : nous utilisons un modèle vectoriel afin de représenter les objets [141]. Plus précisément, chaque objet *it* peut être modelé par un vecteur creux ne contenant que les poids des mots-clés qui lui sont rattachés $k_1, ..., k_z$. Ces éléments sont calculés en utilisant $tf \times idf$. Ici chaque nœud calcule $tf \times idf$ localement, mais des protocoles distribués peuvent être utilisés pour calculer efficacement $tf \times idf$; les détails de l'implémentation d'une solution distribuée pour $tf \times idf$ sont présentés au Chapitre 6. Enfin, le profil de chaque utilisateur est déduit des objets qu'il partage, notés I_u. Ainsi, ce dernier peut être la liste ou la moyenne des vecteurs de chaque objet partagé par l'utilisateur courant u.

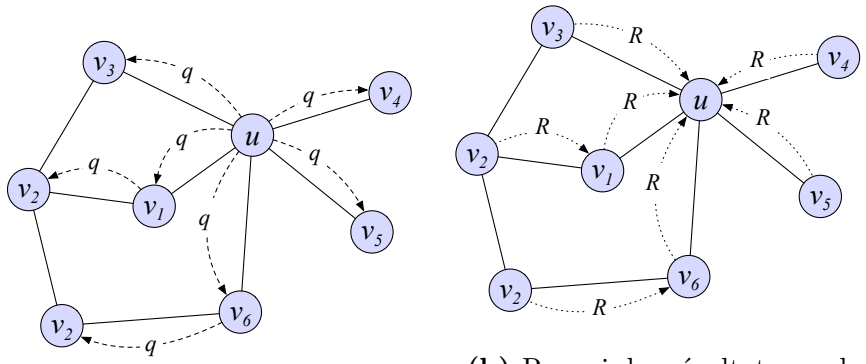
Modèle du réseau *P2P* : notre modèle pair-à-pair peut s'exprimer à partir d'un graphe $G = (U, I, E)$, où $U = \{u_1, ..., u_n\}$ est l'ensemble des utilisateurs disponibles sur le réseau, $I = \{i_1, ..., i_m\}$ tous les objets partagés, et $E = \{e_1, ..., e_k\}$ les arêtes entre utilisateurs et reliant ces derniers avec leurs objets. Ce modèle est très générique et, dans notre cas, les utilisateurs représentent des nœuds du réseau *P2P*.

Définition 5 (*User Network*)**.**

Étant donné un utilisateur u, son *User Network*, ou *U-Net*, réfère au groupe d'utilisateurs que u a regroupé dynamiquement au travers d'échanges périodiques. Le *U-Net* représente les utilisateurs qui seront utilisés pour traiter les requêtes de u. Il y a un lien $e(u, v)$ dans le graphe entre u et v si v est dans le *U-Net* de u.

Construction du réseau *P2P* : En utilisant le *gossip* aléatoire [75], chaque pair maintient localement une vue aléatoire et dynamique d'utilisateurs disponibles sur le réseau. Chacune des entrées de cette vue aléatoire correspond au profil de l'un d'entre eux. Périodiquement, chaque pair va choisir aléatoirement un contact dans cette vue afin de bavarder (*i.e. gossip*). Ces deux pairs vont ensuite échanger un sous-ensemble de leur vue et la mettre à jour à partir de celui qu'ils ont reçu de l'autre pair. Puis, après chaque bavardage, la vue aléatoire est utilisée pour mettre à jour le *U-Net*. Dans la suite de ce chapitre, nous utilisons *Jaccard* comme mesure de pertinence entre deux utilisateurs :

$$\mathbf{Jaccard(u, v)} = |I_u \cap I_v| \ / \ |I_u \cup I_v| \tag{5.1}$$



(a) Propagation de la requête q.

(b) Renvoi des résultats par le même chemin que la requête q.

Figure 5.1: Traitement des requêtes en *P2P*.

Où I_u et I_v sont les attributs des profils de u et v respectivement. Cependant, d'autres méthodes de pertinence sont utilisables.

Traitement des requêtes en *P2P* : comme cela a été mentionné auparavant et illustré dans la Figure 5.1, lorsqu'un utilisateur u soumet une requête q, celle-ci est transmise à un sous-ensemble d'utilisateurs, que nous appelons *U-Net*. Ces derniers vont retourner leurs résultats pertinents à u et vont récursivement transmettre q aux utilisateurs de leur propre *U-Net*, jusqu'à ce que le nombre de sauts atteigne la valeur TTL. Chaque fois qu'un utilisateur v reçoit une requête q, il en calcule les *top-k* objets les plus pertinents via l'application d'une fonction de score, puis les retourne à u. Une recommandation est identifiée par son vecteur $tf \times idf$, l'identifiant de v et son profil. Une fois que u reçoit l'ensemble des recommandations provenant de $v_1, ..., v_n$, il les trie, toujours par le biais d'une fonction de score, et par rapport à la requête q :

$$rec_q = rank(rec_q^1(it_1, ...) \cup ... \cup rec_q^n(it_p, ...)) \tag{5.2}$$

Où $rec_q^i(it_1, ...)$ sont toutes les recommandations provenant d'un utilisateur v_i.

Évaluation des résultats : Afin d'évaluer la capacité à retrouver les objets pertinents, étant donné une requête q, nous utilisons la mesure de *rappel* :

$$recall(q) = \frac{|R^q \cap Rel^q|}{|Rel^q|} \tag{5.3}$$

Où R^q représente l'ensemble des objets récupérés lors du traitement de la requête q et Rel^q représente tous les objets pertinents présents dans le système.

On appelle *couverture* d'un pair u la probabilité que son voisinage puisse répondre aux requêtes soumises par u. La mesure de *rappel* permet donc d'évaluer

cette couverture. Une définition plus précise de la couverture est introduite en section suivante.

Définition du problème : Étant donné un utilisateur $u \in U$, une requête q, les objets I dans G, et le *U-Net* de chaque pair, l'objectif est de maximiser le nombre d'objets pertinents qu'il est possible de récupérer lorsque q est soumise, ou, en d'autres termes, de maximiser la couverture de chaque pair, tout en minimisant le TTL.

5.3 Score et algorithme de regroupement diversifié

Dans cette section, nous montrons que l'utilité, ou *usefulness*, en combinant la pertinence avec la diversité, est un excellent moyen d'accroître la couverture de chaque pair, et donc, le niveau de rappel dans les solutions de recherche et recommandation utilisant les protocoles de *gossip*. En premier lieu, nous montrons formellement que pour augmenter le rappel, le score de *regroupement* doit être exprimé en termes d'utilité, combinant pertinence et diversité. Ensuite, nous présentons l'algorithme de *regroupement Useful U-Net* utilisant les protocoles de *gossip*.

5.3.1 Score d'utilité, ou *usefulness*

Le score d'utilité doit être conçu de manière à maximiser la probabilité que u récupère les objets pertinents aux requêtes qu'il soumet : il s'agit de la probabilité de couverture ou *coverage*. En d'autres termes, le voisinage de u, noté $v_1, ..., v_n \in G$, doit être choisi de manière à maximiser le nombre d'objets pertinents (par rapport aux requêtes que u soumettra) qu'il peut renvoyer.

Soit Q, l'ensemble inconnu de toutes les requêtes possibles et $\mathcal{P}(Q_v)$ la probabilité qu'un utilisateur v retourne au moins un élément pertinent étant donné une requête $q \in Q$. Il est possible d'étendre le raisonnement en utilisant $\mathcal{P}(Q_v^u)$, la probabilité que l'utilisateur v puisse répondre à une requête soumise par u ; en effet, toutes les requêtes $q \in Q$ n'ont pas la même probabilité d'être soumises par u. Par la suite, nous définissons strictement la couverture en fonction de $\textit{U-Net}_u = \{v_1, ..., v_n\}$. Cette notion de couverture permet d'exprimer ce qu'est l'utilité d'un utilisateur par rapport aux autres utilisateurs du *U-Net*.

Définition 6 (Couverture ou *coverage*)**.**

Étant donné Q et $U\text{-}Net_u = \{v_1, ..., v_n\}$, les utilisateurs présents dans le *U-Net* de u, la couverture est la probabilité qu'au moins un de ces derniers $\{v_1, ..., v_n\}$ puisse retourner un objet pertinent étant donné une requête $q \in Q$. Cette probabilité peut se formuler de la manière suivante : $\mathcal{P}(Q^u_{v_1} \cup Q^u_{v_2} \cup ... \cup Q^u_{v_n})$.

Les profils utilisateurs $v_1, ..., v_n$ doivent donc être sélectionnés de façon à maximiser cette probabilité. L'Équation 5.4 la développe en fonction de chaque utilisateur dans le *U-Net* de u.

$$\mathcal{P}(Q^u_{v_1} \cup ... \cup Q^u_{v_n}) = \sum_{j \in 1,...,n} (\mathcal{P}(Q^u_{v_j}) - \mathcal{P}(Q^u_{v_j} \cap (Q^u_{v_{j+1}} \cup ... \cup Q^u_{v_n}))) \tag{5.4}$$

$\mathcal{P}(Q^u_{v_j}) - \mathcal{P}(Q^u_{v_j} \cap (Q^u_{v_{j+1}} \cup ... \cup Q^u_{v_n}))$ représente la couverture ajoutée par un utilisateur v_j par rapport aux utilisateurs $v_{j+1}, ..., v_n$. Notons que lorsque $j = n$, seule la probabilité $\mathcal{P}(Q^u_{v_j})$ est calculée puisqu'il n'y a pas d'autres utilisateurs avec qui comparer v_j.

Nous définissons maintenant l'utilité de l'utilisateur v_j, déduite à partir de la probabilité de couverture.

Définition 7 (Utilité ou *usefulness*)**.**

Étant donné le *U-Net* de u et une requête q, l'utilité d'un profil v_j est la probabilité qu'il puisse retourner des objets pertinents par rapport à q, qu'aucun autre utilisateur du *U-Net* ne pouvait retourner. En d'autres termes, cela se définit comme suit :

$$\begin{aligned} &usefulness(u, v_j | v_{j+1}, ..., v_n) = \\ &\qquad \mathcal{P}(Q^u_{v_j}) - \mathcal{P}(Q^u_{v_j} \cap (Q^u_{v_{j+1}} \cup ... \cup Q^u_{v_n})) \end{aligned} \tag{5.5}$$

L'Équation 5.5 montre que l'utilité doit considérer la pertinence d'un utilisateur $\mathcal{P}(Q^u_{v_j})$ mais aussi sa redondance par rapport aux autres utilisateurs $\mathcal{P}(Q^u_{v_j} \cap (Q^u_{v_{j+1}} \cup ... \cup Q^u_{v_n}))$.

Dans la suite, nous montrons que cette formule d'utilité, notée $usefulness(u, v_j | v_{j+1}, ..., v_n)$, peut se développer en un modèle probabiliste de diversification connu [17, 39]. Dans l'Équation 5.6, nous transformons l'utilité en une probabilité conditionnelle :

$$\begin{aligned} &\mathcal{P}(Q^u_{v_j}) - \mathcal{P}(Q^u_{v_j} \cap (Q^u_{v_{j+1}} \cup ... \cup Q^u_{v_n})) \\ &\qquad = \mathcal{P}(Q^u_{v_j}) \times (1 - \mathcal{P}(Q^u_{v_{j+1}} \cup ... \cup Q^u_{v_n} | Q^u_{v_j})) \\ &\qquad = \mathcal{P}(Q^u_{v_j}) \times \mathcal{P}(\bar{Q}^u_{v_{j+1}} \cap ... \cap \bar{Q}^u_{v_n} | Q^u_{v_j}) \end{aligned} \tag{5.6}$$

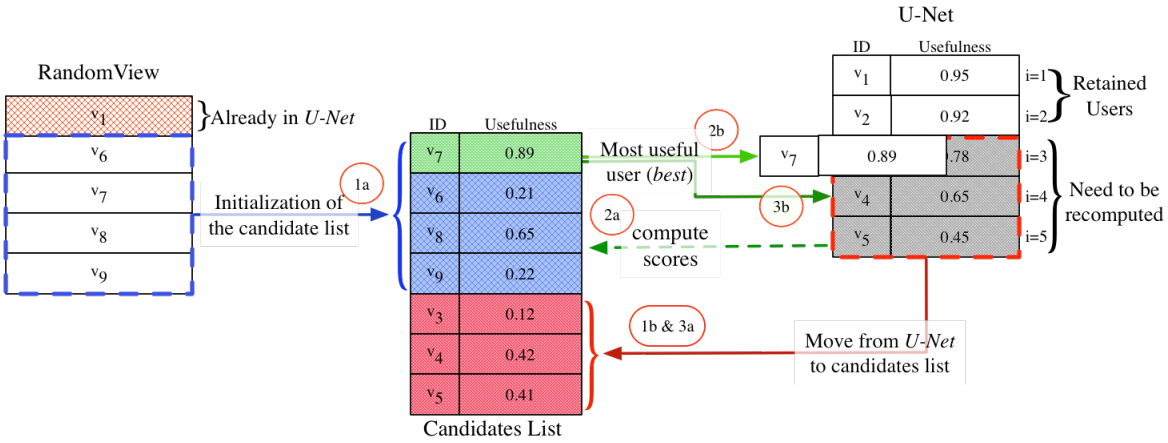


Figure 5.2: Exemple de l'exécution de l'algorithme *Useful-U-Net*.

De manière similaire à [17, 39, 36], nous supposons que la redondance entre deux utilisateurs est indépendante de leur redondance avec les autres, et nous en déduisons l'Équation 5.7.

$$\mathcal{P}(Q^u_{v_j}) \times \mathcal{P}(\bar{Q}^u_{v_{j+1}} \cap ... \cap \bar{Q}^u_{v_n} | Q^u_{v_j}) = \mathcal{P}(Q^u_{v_j}) \times \prod_{i \in j+1,...,n} (1 - \mathcal{P}(Q^u_{v_i} | Q^u_{v_j})) \quad (5.7)$$

Finalement, on peut observer que l'utilité d'un utilisateur est clairement similaire au problème de diversification probabiliste utilisé dans [17, 39] et peut se représenter comme dans l'Équation 5.8.

$$usefulness(u, v_j | v_{j+1}, ..., v_n) = rel(u, v_j) \times \prod_{i \in j+1,...,n} (1 - red(v_j, v_i)) \quad (5.8)$$

Où $rel(u, v_j) = \mathcal{P}(Q^u_{v_j})$ est la pertinence du profil de v_j et $red(v_j, v_i) = \mathcal{P}(Q^u_{v_i} | Q^u_{v_j})$ sa redondance par rapport au profil de l'utilisateur v_i. Attention, ici la diversité n'est pas utilisée dans le processus de recommandation, mais bien dans celui de *regroupement*.

5.3.2 Algorithme de regroupement *Useful U-Net*

Notre algorithme de *regroupement* est présenté ici. Il permet de maintenir un *U-Net* maximisant l'utilité de chacun de ses utilisateurs, en utilisant le score précédemment proposé.

Étant donné l'ensemble des utilisateurs dans la vue aléatoire, le but de l'algorithme de *regroupement* est de calculer l'utilité de chacun d'entre eux, en tenant compte de ceux déjà regroupés dans le *U-Net*, et de le mettre à jour en conséquence.

En utilisant le *gossip* aléatoire [75], chaque utilisateur u maintient un ensemble aléatoire d'utilisateurs disponibles sur le réseau. Périodiquement, chacun échange

un sous-ensemble de cette vue. Après la phase de fusion où les sous-ensembles des deux utilisateurs sont fusionnés avec leur vue aléatoire propre, l'algorithme de *regroupement* est déclenché. En réalité, en tenant compte du dernier bavardage, l'algorithme va sélectionner les utilisateurs les plus utiles de cette vue, afin de maximiser la couverture du *U-Net*.

L'algorithme exploite trois structures de données : la vue aléatoire, le *U-Net* et une liste des candidats – celle-ci a la même fonction que dans l'algorithme de *top-k* diversifié présenté au Chapitre 3. Les deux premières structures de données sont initialisées lorsque u rejoint le réseau, alors que la liste des candidats l'est lorsque l'algorithme de *regroupement* est déclenché, et contient les profils qui pourront potentiellement intégrer le *U-Net* par la suite.

Nous présentons avec plus de détails cet algorithme en nous appuyant sur l'exemple présenté dans la Figure 5.2. La vue aléatoire correspond aux profils des utilisateurs v_1, v_6, v_7, v_8, v_9. Les précédents utilisateurs présents dans le *U-Net* sont v_1, v_2, v_3, v_4, v_5. En supposant que l'algorithme soit exécuté sur le nœud de l'utilisateur u, il prend comme entrée le profil de u, sa vue aléatoire, notée $RandomView_u$, et son $U\text{-}Net_u$. La structure de données utilisée pour le *U-Net* est un tableau de taille N de profils d'utilisateurs, associés à leur score d'utilité et triés via ce dernier, de manière décroissante. La sortie de l'algorithme est le *U-Net* mis à jour. L'algorithme *Useful U-Net* se décompose en trois étapes :

1. Premièrement (lignes 1 à 6), l'utilisateur le plus utile (*i.e. usefull*) de la vue aléatoire doit être trouvé. Puis, la position i à laquelle il sera inséré dans le *U-Net* doit être calculée (rappelons que le score d'utilité d'un utilisateur dépend de sa position dans le *U-Net*). Cela permettra de limiter la mise à jour du *U-Net* aux utilisateurs dont la position s'échelonne de i à N. Afin de trouver l'utilisateur le plus utile de la vue aléatoire, l'algorithme va d'abord initialiser la liste des candidats avec l'ensemble de ceux provenant de la vue aléatoire, sauf pour ceux déjà présents dans le *U-Net* (ligne 2). Dans la Figure 5.2, v_1 est déjà dans le *U-Net*, et la liste des candidats est initialisée avec les utilisateurs v_6, v_7, v_8, v_9 ($1a$). Pour chaque position i (en partant de 1) dans le *U-Net*, le score d'utilité de chaque candidat est calculé en utilisant l'Équation 5.8 et en prenant en compte des utilisateurs du *U-Net* aux positions $1, ..., i-1$; étant donné i, si le meilleur candidat a un score d'utilité supérieur à celui de l'utilisateur $U\text{-}Net_u[i]$, alors cette étape se termine (ligne 6). Si plusieurs candidats obtiennent le meilleur score, le choix se fait de manière aléatoire. Dans la Figure 5.2, v_7 est plus utile que v_3 à la troisième position dans le *U-Net* car l'utilité de v_3 est de 0.78 alors que celle de v_7 est de 0.89 ($1b$). Si aucun candidat possède à un moment donné un score supérieur à un utilisateur du *U-Net* et que la position N est atteinte, alors l'algorithme s'arrête complètement.
2. La seconde partie (lignes 7 à 10) déplace les profils des utilisateurs restant dans le *U-Net* (de la position i à N) vers la liste des candidats ($2a$) ; leur

score d'utilité doit être recalculé (cf. Équation 5.8) en tenant compte du candidat le plus utile (calculé en étape 1). Ensuite, ce candidat est inséré en position i dans le *U-Net*. Dans l'exemple de la Figure 5.2, les profils des utilisateurs v_3, v_4, v_5 sont déplacés vers la liste des candidats et l'utilisateur v_7 est inséré en position 3 dans le *U-Net* ($2a$ and $2b$).

3. Enfin, dans la dernière partie (lignes 11 à 15), l'algorithme calcule itérativement, pour chaque position vide i du *U-Net* (*i.e.* les positions vidées dans l'étape 2), le score d'utilité de chaque candidat, en fonction des utilisateurs aux positions 1 à $i-1$ (lignes 12 et 13, et étape $3a$ dans la figure). Ensuite, le candidat le plus utile est inséré à la position i et supprimé de la liste des candidats (ligne 15 et étape $3b$ dans la figure). L'algorithme répète ces étapes jusqu'à ce que toutes les positions du *U-Net* soient remplies (ligne 11).

Rappelons que les protocoles de *gossip* convergent rapidement [49]. Ce sera ainsi le cas du *U-Net* qui se stabilisera alors en s'arrêtant de plus en plus à l'étape $1b$, puisqu'il sera déjà composé des utilisateurs les plus *utiles*. Rappelons que la convergence indique que le système se dirige vers un état de stabilité (*i.e.* les *U-Net* ne sont plus du tout mis à jour car ils contiennent les utilisateurs les plus pertinents et divers) ; le système a convergé lorsqu'il est devenu stable.

5.4 Réplication

La réplication des objets, en augmentant leur disponibilité sur le réseau, permet d'accroître la couverture de chaque pair, et donc, les niveaux de rappel. Dans cette section, nous présentons notre stratégie *best similarity replication* (*i.e. BSR*) qui choisit sélectivement les *objets manquants* entre utilisateurs très similaires afin de les répliquer et permet ainsi de compenser la perte d'objets pertinents provenant d'utilisateurs redondants. Rappelons que les *objets manquants*, étant donné u, représentent tous les objets provenant d'utilisateurs similaires à u mais qui ne sont pas dans le *U-Net* de u car ils sont trop redondants par rapport aux autres utilisateurs. Nous proposons ensuite une solution de réplication hybride, combinant *BSR* et *Query Path Replication* (*i.e. QPR*), technique qui réplique les objets retrouvés par rapport aux requêtes sur les chemins qu'elles ont parcourus. En d'autres termes, nous combinons la rapidité de réplication de *BSR* aux très hauts niveaux de rappel offerts par *QPR*.

5.4.1 *Best Similarity Replication*

L'utilisation de l'*utilité* comme score de *regroupement* implique que deux utilisateurs $v_i, v_j \in \textit{U-Net}_u$ ont une forte probabilité d'être divers. Ainsi, étant donné un utilisateur v_l très similaire à v_i, ce dernier a peu de chance de se retrouver

Algorithme 8: Useful *U-Net*

Input: u profile, $U\text{-}Net_u$ (array[1..N]), $RandomView_u$
Output: $U\text{-}Net_u$ is updated with respect to the $RandomView$

```
1  candidates : unsorted list of user profiles;
2  candidates ← RandomView_u − U-Net_u;
3  best ← ∅;
4  i ← 0;
5  repeat
6      i++;
7      for each c_j ∈ candidates do
       score(c_j) ← usefulness(c_j,u,U-Net_u[1..i − 1]) ;
8      best ← arg max_{c∈candidates}(score(c));
9  until i=N or score(best) > score( U-Net[i]);
10 if score(best) > score( U-Net[i]) then
11     after← U-Net_u[i..N];
12     U-Net_u[i] ← best;
13     i++;
14     candidates ← candidates − best;
15     candidates ← after ∪ candidates;
16     U-Net_u ← U-Net_u − after;
17     while i < N and candidates≠ ∅ do
18         for each c_j ∈ candidates do
19             score(c_j) ← usefulness(c_j, u, U-Net_u[1..i − 1]);
20         best ← arg max_{c∈candidates}(score(c));
21         U-Net_u[i] ← best;
22         candidates ← candidates − best;
23         i++;
```

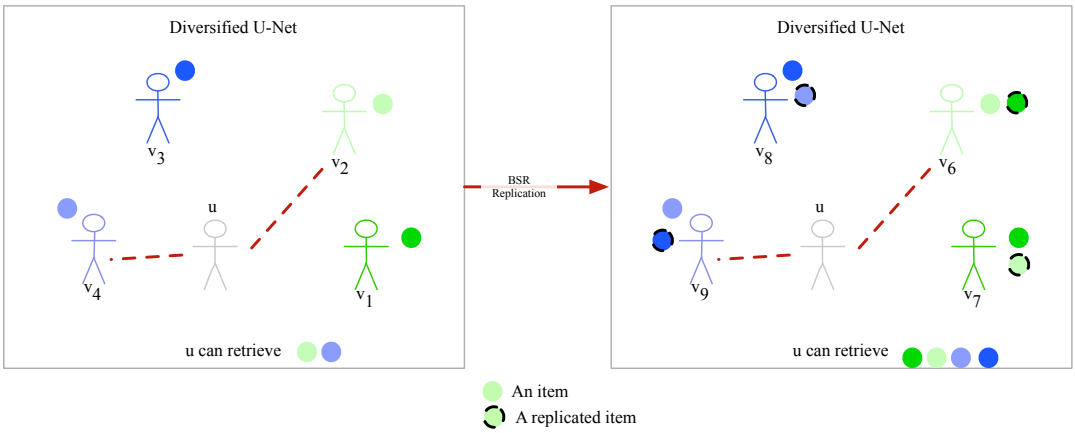


Figure 5.3: Bénéfices de *BFS* lorsque les *U-Net* sont diversifiés.

dans $U\text{-}Net_u$, et les objets qu'il partage, mais qui ne le sont pas par v_l, ont une plus faible probabilité d'être retrouvés par u ; cela se produit alors même que v_l peut être très pertinent pour u. Ces objets sont dits *manquants*.

Afin de limiter cette perte de pertinence impliquée par le score d'utilité, nous proposons la stratégie *Best Similarity Replication* (*BSR*). Étant donné un utilisateur v_i, *BSR* va garder k réplicas correspondants aux *missing items* partagés par les utilisateurs qui lui sont le plus similaires, où k est un paramètre défini par le système. La Figure 5.3 illustre cette idée. On peut observer que les objets répliqués par les utilisateurs du *U-Net* proviennent d'autres utilisateurs qui ont une faible probabilité de se retrouver dans ce même *U-Net* puisqu'ils sont redondants. Dans notre cas, un réplica n'est qu'une référence indiquant chez quel utilisateur se trouve réellement l'objet. Il est donc représenté par la structure suivante :

$$replica = \langle item_{ID}, item_{tf \times idf}, user_{ID}, user_{profile} \rangle \tag{5.9}$$

Où $item_{ID}$ est l'identifiant de l'objet, $item_{tf \times idf}$ son vecteur $tf \times idf$, $user_{ID}$ l'identifiant (*i.e.* adresse IP) de l'utilisateur partageant l'objet et $user_{profile}$ est son profil (*i.e.* vector $tf \times idf$).

Ainsi, si un utilisateur transmet une requête q à v_i, ce dernier va calculer les *top-k* objets les plus pertinents, en tenant compte de ceux qu'il partage lui-même mais aussi de ceux qu'il a répliqués, provenant d'utilisateurs qui lui sont similaires. u sera donc capable de retrouver des résultats provenant d'utilisateurs trop redondants pour être gardés dans son *U-Net*.

Chaque utilisateur u est donc associé à un *cache de réplication* et à ses propres objets, notés $items_u$. Le premier élément liste les réplicas et chacun d'entre eux y est associé à un *timestamp* indiquant la date à laquelle il y a été inséré. Si un réplica n'est pas apparu sur le réseau depuis trop longtemps (défini par le système), il en est supprimé.

Étant donné deux utilisateurs u et v, le nombre maximum de réplicas que u peut stocker, provenant de v, noté max_v, est calculé comme suit :

$$max_v = r_u - |RepC_u| + replace(u, RepC_u, v) \tag{5.10}$$

Où r_u est la taille maximale du cache de réplication et $|RepC_u|$ la taille actuelle du cache. $replace(u, RepC_u, v)$ est le nombre de réplicas présent dans le cache de réplication de u, noté $RepC_u$, qui peuvent être remplacés par ceux provenant de v. Cette valeur est calculée comme suit :

$$\begin{aligned} replace(u, RepC_u, v) = |\{item| \\ score(item) < sim(u, v), item \in RepC_u\}| \end{aligned} \tag{5.11}$$

Où $score(item)$ est la similarité entre l'utilisateur partageant $item$ et u. En d'autres termes, $replace(u, RepC_u, v)$ définit le nombre de réplicas qui proviennent d'utilisateurs moins similaires à u que v, et donc qui peuvent être remplacés par ceux provenant de v.

L'algorithme *BSR*, en utilisant les protocoles de *gossip*, se compose d'une partie active et d'une autre, passive. La première est exécutée par u qui demande à recevoir de nouveaux réplicas ; la seconde le sera par les utilisateurs qui vont fournir des réplicas à u. L'Algorithme 9 présente la partie active. Celle-ci est déclenchée à chaque fois que la vue aléatoire est mise à jour. Une liste, notée *users*, est alors générée avec tous les utilisateurs présents dans la vue aléatoire.

Étant donné u, l'entrée de l'algorithme 9 est $items_u$, *users* et le cache de réplication de u, noté $RepC_u$. La sortie est le cache mise à jour.

BSR trie tout d'abord les utilisateurs *users* par leur similarité par rapport à u (*e.g. Jaccard*) de manière décroissante (ligne 1). Ensuite, l'algorithme va itérativement calculer pour chaque utilisateur $v \in users$ le nombre maximum de réplicas max_v que u peut garder de v (lignes 3 et 9). Si ce nombre est positif, *BSR* initie un échange avec l'utilisateur v. Pendant cet échange, u envoie à v un message ayant la structure suivante :

$$\langle items_u, RepC_u, max_v \rangle$$

Afin d'éviter le transfert d'un nombre trop important de données entre u et v, $items_u$ et $RepC_u$ peuvent être implémentés comme des filtres de *Bloom* [29]. Enfin, v retourne jusqu'à max_v réplicas qui ne sont pas dans $items_u$ mais qui peuvent déjà se trouver dans le cache de réplication de u. Ainsi, ce cache pourra associer plusieurs utilisateurs à un unique réplica qui y restera donc disponible plus longtemps. L'algorithme s'arrête lorsque le nombre de réplicas que u peut garder de v est égal à 0 (ligne 7). Les utilisateurs suivants étant encore moins similaires, ce nombre sera aussi égal à 0.

Algorithme 9: BSR (Active User)

Input: $items_u$, $users$, $RepC_u$
Output: the updated replication cache.
sort $users$ in decreasing order of similarity with respect to u;
$v \Leftarrow first(users)$;
repeat
  $max_v \Leftarrow r_u - |RepC_u| + replace(u, RepC_u, v)$;
  send max_v, $items_u$, $RepC_u$ **to** v;
  $replicas \Leftarrow$ **receive items replicas from** v;
  add $replicas$ **to** $RepC_u$;
  $v \Leftarrow next(users)$;
until $v = \emptyset$ ***or*** $max_v = 0$;

5.4.2 Stratégie hybride

Query Path Replication (*QPR*) [44] est une méthode de réplication conçue pour les réseaux non-structurés. Lorsqu'une requête q est soumise, il s'agit de répliquer ses résultats sur le chemin de q. Cependant, la rapidité du processus de réplication dépend de la quantité de requêtes soumises et du TTL, et peut donc prendre un certain temps.

Nous proposons ici une solution hybride exploitant les caractéristiques intéressantes de *BSR* et de *QPR*. Avec la première d'entre elles, les réplicas sont propagés périodiquement, avec la seconde, ils le sont à chaque requête. Cependant, si le cache est plein et que *QPR* souhaite ajouter de nouveaux réplicas, ceux provenant des utilisateurs les moins similaires et issus de *BSR* sont retirés. En effet, les réplicas de *QPR* ont une plus forte probabilité de contribuer à une augmentation de la couverture de chaque pair, et donc, du rappel. En effet, *QPR* réplique les objets qui répondent aux requêtes des utilisateurs et donc, par définition, qui sont populaires. Si *QPR* souhaite insérer de nouveaux réplicas, et qu'il n'y en a déjà plus en provenance de *BSR*, le réplica *QPR* le plus ancien est ôté.

Ainsi, comme cela est montré en Section 5.5, *BSR* permet une croissance rapide du rappel grâce aux propriétés de dissémination propres au *gossip*, puis, en parallèle, *QPR* permettra d'accroître, certes plus doucement, mais à des niveaux encore plus élevés, le rappel grâce aux effets de popularités implicites (*i.e.* les objets les plus populaires seront les plus répliqués).

5.5 Évaluation Expérimentale

Dans cette section, nous fournissons l'évaluation expérimentale de nos contributions, que nous comparons avec d'autres solutions de l'état de l'art. Nous avons mené nos expériences sur trois jeux de données correspondant à *MovieLens*, *Flickr*

et *LastFM*. Nous introduisons tout d'abord, en Section 5.5.1, le protocole expérimental de notre évaluation. Puis, en Section 5.5.2, nous présentons et discutons les résultats de nos expériences.

5.5.1 Mise en place des expériences

Nous avons réalisé nos expériences via le simulateur *PeerSim*[1]. Trois jeux de données ont été utilisés : *MovieLens*, *Flickr* et *LastFM*, chacun possédant des caractéristiques différentes, présentées par la suite. Le premier est composé d'utilisateurs ayant évalué des films, le second d'utilisateurs ayant partagé des photos, ou les ayant ajouté à leurs favoris, et le dernier d'utilisateurs écoutant des musiques associées à des artistes, pour lesquels ils peuvent soumettre des étiquettes (*i.e. tags*). Chaque jeu de données possède des caractéristiques différentes : les utilisateurs sont plus ou moins redondants et le nombre d'objets partagés par chacun plus ou moins grands. Les caractéristiques de chaque jeu de données sont résumées dans le Tableau suivant :

dataset	**items**	**# items**	**# users**	**avg items/user**
MovieLens	Movies	3,900	6,040	166
Flickr	Pictures	2,029	2,000	3.7
LastFM	Artists	23,346	2,000	98

Les requêtes utilisées dans nos expériences sont construites comme suit. Premièrement, pour *MovieLens*, étant donné un utilisateur u, une partie aléatoire des films partagés par ce dernier sera utilisée comme des requêtes, une seconde sera réellement partagée dans les expériences. Plus précisément, les mots extraits des titres des films à chercher composent les mots clés des requêtes. Dans *Flickr* et *LastFM*, les requêtes sont définies comme l'association aléatoire d'étiquettes issues d'un objet particulier.

Une expérience se déroule en deux temps. Les utilisateurs vont tout d'abord effectuer des échanges de *gossip* pendant 400 tours afin que le réseau converge (*i.e.* atteigne un état stable où les *U-Net* des utilisateurs n'évoluent plus). Cette valeur est choisie expérimentalement et permet d'avoir la garantie que le réseau a réellement convergé. Ensuite, chaque 20 tours de *gossip*, tous les utilisateurs soumettent leurs requêtes. L'expérience s'arrête après 500 tours de *gossip*. Nous calculons alors le rappel moyen au cours de l'expérience. Il s'agit d'une mesure permettant d'évaluer la fraction des objets que le système a réussi à recommander avec succès, comme cela est présenté en Section 5.2. Sur le jeu de données *MovieLens*, le rappel prend la valeur de 1 si le film est trouvé, et de 0 si ce n'est pas le cas. Sur *Flickr* ou *LastFM*, le rappel est calculé comme le pourcentage des images ou des artistes pertinents, c'est-à-dire contenant tous les mots de la requête, qui sont retournés à l'utilisateur. Sur les expériences relatives aux jeux de données *Flick* et

1. `www.peersim.sourceforge.net`

LastFM, nous avons également mesuré la *variance* qui permet d'évaluer la variabilité du rappel et dont la formule est la suivante : $V(X) = 1/N \times \sum_{i=1}^{n}(x_i - m)^2$ où m est le rappel moyen.

Les scores suivants ont été utilisés dans nos expériences :

$\mathbf{overlap}(\mathbf{u}, \mathbf{v}) = \lvert I_u \cap I_v \rvert$	$\mathbf{over\_big}(\mathbf{u}, \mathbf{v}) = \lvert I_u \cap I_v \rvert + \lvert I_v \rvert$
$\mathbf{Jaccard}(\mathbf{u}, \mathbf{v}) = \lvert I_u \cap I_v \rvert \ / \ \lvert I_u \cup I_v \rvert$	$\mathbf{cosine}(\mathbf{u}, \mathbf{v}) = Ir_u \times Ir_v \ / \ \lvert\lvert Ir_u \rvert\rvert \times \lvert\lvert Ir_v \rvert\rvert$

Où I_u et I_v sont les objets partagés par u et v respectivement, et où Ir_u et Ir_v constituent les ensembles de notes que u et v ont respectivement données aux objets qu'ils partagent. Nous avons fixé la taille du *U-Net* à 16 et le TTL à 3. D'autres valeurs montrant des résultats similaires ont été testées. La taille de la vue aléatoire (5 dans notre cas) n'est pas importante car elle ne modifie que la vitesse de convergence. En effet, la taille de la vue aléatoire ne change que le nombre d'utilisateurs présents sur le réseau que u pourra découvrir à chaque tour de *gossip*.

Discussion sur le rappel et la précision : l'utilisation du rappel comme mesure en *P2P* (*rappel-P2P*) n'a pas la même utilité que cette même mesure en recherche d'information. En effet, en *P2P*, lorsqu'une requête est soumise par un utilisateur u, celle-ci n'atteint pas tous les utilisateurs du réseau ; le nombre d'utilisateurs du réseau qui recevront q dépend du TTL (*i.e.* nombre de sauts que peut faire la requête sur le réseau) et de la taille du *U-Net* (*i.e.* nombre d'utilisateurs à chaque saut qui recevront la requête).

La Figure 5.4a présente cette idée où tous les utilisateurs ne reçoivent pas la requête q, étant donné l'utilisateur u et *TTL=2*.

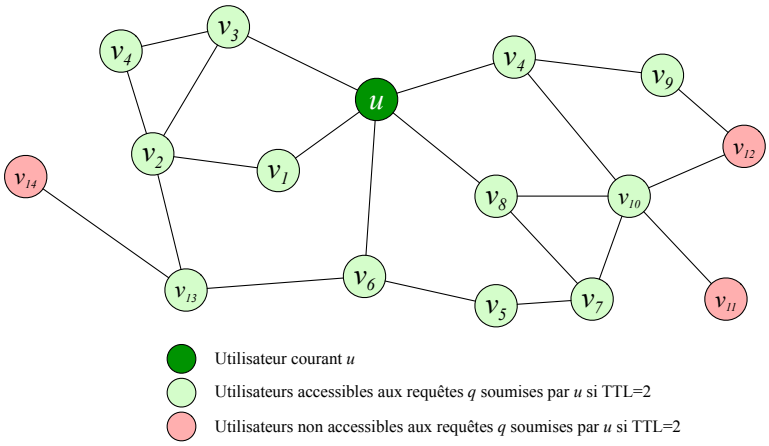
La Figure 5.4b montre l'utilisation du rappel avec la précision, comme cela est le cas en recherche ou en recommandation centralisées, et la Figure 5.4c montre son utilisation avec le TTL ou avec la taille du *U-Net* – les deux éléments ont un impact sur le rappel. Nos expériences consistent à fixer la valeur du TTL et du *U-Net* et à observer l'impact des méthodes utilisées dans le calcul du voisinage sur le rappel. Dans la Figure 5.4c, par exemple, nous avons fixé le TTL à 3 et le *U-Net* à 16 et, lorsque la diversité est prise en compte, le rappel monte jusqu'à $0,98$ alors qu'il n'était que de $0,58$ sans.

Le *rappel-P2P* (ou tout simplement rappel) n'a donc pas besoin d'être combiné avec la précision, puisqu'il est contrebalancé par le TTL et la structure du réseau (*i.e.* taille du *U-Net*).

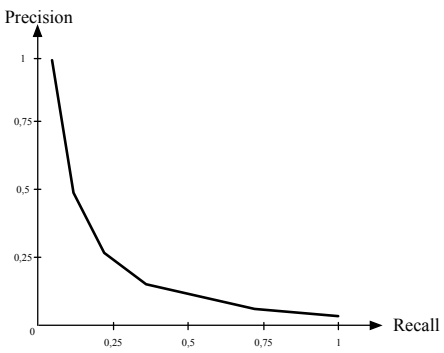
5.5.2 Expériences

Effet de la diversité

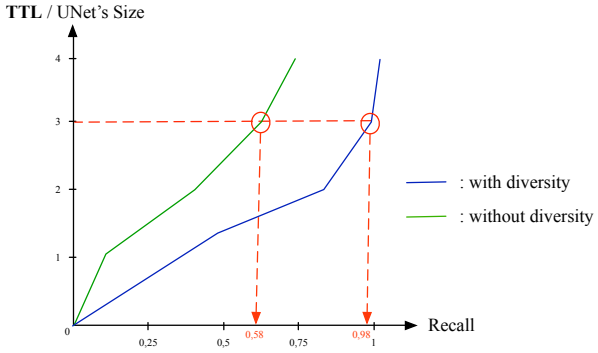
La Figure 5.5 présente les résultats de nos expériences sur l'effet de la diversité. Les Figures 5.5a, 5.5b et 5.5c comparent plus précisément les valeurs de rappels, lorsque différentes mesures de pertinence sont utilisées avec et sans *utilité*. Les



(a) Réseau *P2P* avec mise en avant des pairs qui sont atteignables par les requêtes de u si TTL=*2*.



(b) Utilisation du rappel avec la précision.



(c) Utilisation du rappel avec le TTL ou avec la taille du *U-Net*.

Figure 5.4: Utilisation du rappel dans les expériences *P2P*.

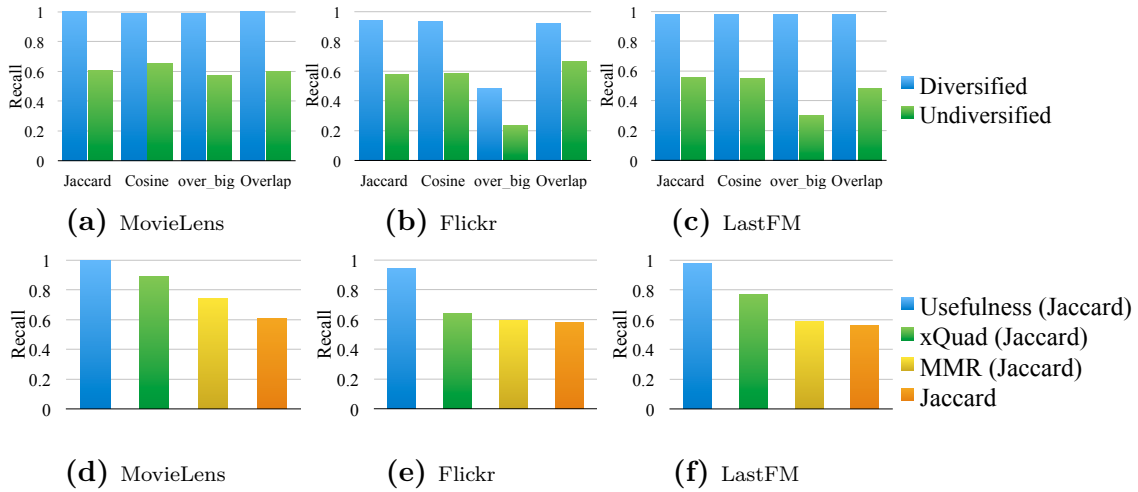


Figure 5.5: Effet de la diversification sur le rappel.

Figures 5.5d, 5.5e et 5.5f, quant à elles, analysent les effets de plusieurs méthodes de diversification dans le score d'utilité.

Sans surprise, diversifier le *U-Net* permet à chaque score de pertinence d'augmenter significativement son rappel. Sur le jeu de données *MovieLens*, les valeurs de rappels non diversifiées s'échelonnent entre $0,58$ et $0,62$, alors qu'elles augmentent entre $0,978$ et $0,999$ avec la diversification. Sur le jeu de données *Flickr*, les gains sont légèrement inférieurs car les profils des utilisateurs qui le composent sont déjà beaucoup plus divers que sur les autres jeux de données. Ainsi, la diversification a moins d'impact. Enfin, sur le jeu de données *LastFM*, le rappel monte jusqu'à $3,26$ fois sa valeur initiale lorsque le score d'utilité est utilisé.

À cela s'ajoute le fait que le score d'utilité permet même de réduire la variabilité du rappel en fonction des requêtes. En effet, sur *Flickr*, la variance diminue de $0,116$ à $0,013$ lorsque la mesure de pertinence est `Jaccard`. Cela peut être expliqué par le fait que dans les solutions non-diversifiées, les utilisateurs du *U-Net* sont très similaires. En conséquence, lorsqu'une requête est soumise, ils peuvent soit, tous y répondre, soit aucun. La diversification augmente la couverture et donc la probabilité qu'au moins un utilisateur du *U-Net* puisse retourner des objets pour toutes les requêtes q soumises.

Nous avons ensuite mesuré l'effet que pouvait avoir une modification de la taille du *U-Net* ou de la valeur du TTL. Par exemple, sur le jeu de données *MovieLens*, avec un *U-Net* de taille 5 et un TTL de taille 2, le rappel est en moyenne $2,37$ fois plus élevé lorsque le score d'utilité est utilisé. En effet, sans diversification, les valeurs de rappel tournent autour de $0,26$ alors qu'elles atteignent environ $0,61$ avec.

Nous avons aussi comparé trois différentes méthodes de diversification. La première est l'*utilité*, présentée dans l'Équation 5.8. La seconde est *Maximal Marginal*

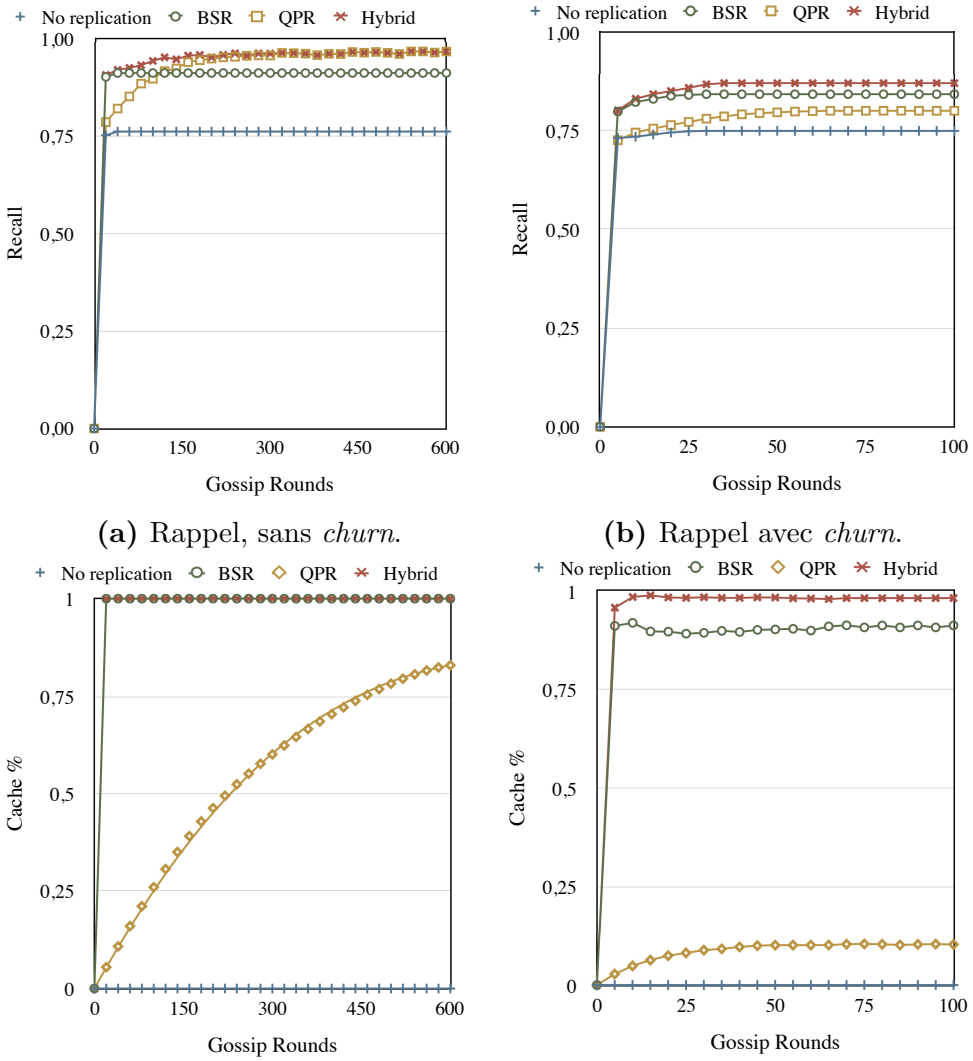
Relevance, méthode abordée au Chapitre 2, connue sous le nom de *MMR* [36]. Celle-ci choisit les utilisateurs qui minimisent la similarité maximale qui peut exister étant donné chaque couple d'utilisateurs du *U-Net*. Enfin, la dernière méthode est *Explicit Query Aspect Diversification* plus fréquemment appelée *xQuad* [142]. Cette dernière choisit des utilisateurs de telle manière que chacun d'entre eux soit similaire à u d'une manière différente. Supposons par exemple que u partage i_1 et i_2. Si v_1 est dans le *U-Net* de u et lui est similaire parce qu'il partage aussi i_1, alors *xQuad* sélectionnera un utilisateur v_2 tel qu'il est similaire à u du fait qu'il partage l'objet i_2. Dans cette expérience, *Jaccard* est la mesure de pertinence.

Les Figures 5.5d, 5.5e et 5.5f montrent que chaque méthode de diversification permet de maximiser les valeurs de rappel comparées aux méthodes non diversifiées. Parmi elles, l'*utilité* obtient les meilleurs gains, rapidement suivie par *xQuad*. *MMR* montre quant à lui les pires gains de rappel. En effet, cette méthode choisit des utilisateurs qui minimisent la similarité maximale qui peut exister entre chaque couple d'utilisateurs du *U-Net*. Ainsi, *MMR* préfère un groupe d'utilisateurs tous moyennement similaires, ce qui limitera le gain de rappel.

Effet de la réplication

La Figure 5.6 montre les résultats de nos expériences sur la réplication. Nous étudions notamment l'effet de la réplication sur le rappel. Plusieurs stratégies de réplication sont comparées. Ici, la valeur du TTL est descendue à 2. Les résultats montrent les valeurs de rappel et le pourcentage de cache rempli par des réplicas. En effet, bien que la taille du cache soit de 50, les différentes stratégies ne convergent pas à la même vitesse. Dans ces expériences, nous examinons également l'effet du *churn*, c'est-à-dire le fait que des utilisateurs se déconnectent et se re-connectent. Celui-ci est fixé à 0.05% des utilisateurs quittant et joignant le réseau à chaque tour de *gossip*. Cette valeur, bien qu'identique pour tous les utilisateurs, ce qui n'est pas le cas en réalité, est cohérente avec le *churn* observé sur *Bittorrent* [158], populaire réseau *P2P*. Le score de *regroupement* est l'*utilité*.

De manière non surprenante, la réplication permet d'augmenter la couverture, et donc, le rappel. Dans la Figure 5.6a, sans *churn*, *BSR* permet un rappel maximal de 0.91 en 30 tours de *gossip*. En comparaison, les valeurs de rappel de *QPR* sont encore en train de croître après 600 tours de *gossip*. Cependant, cette valeur gagne 0.91 après 120 tours de *gossip* et 0.964 après 500 tours. En combinant chaque méthode (stratégie hybride), de meilleures valeurs sont atteintes pendant toute l'expérience. La solution hybride dépasse 0.91 de rappel après 25 tours de *gossip* seulement, 0.96 après 200 et 0.965 après 500.
Les Figures 5.6b et 5.6d présentent la même expérience sous *churn*. Chacune des méthodes atteint sa valeur maximale de rappel avant la fin de l'expérience. *BSR* obtient 0.84 après 30 tours de *gossip*, et *QPR*, 0.8 après 65 tours. Ainsi, *QPR* est à la fois plus lente à accroître le rappel et atteint des niveaux plus faibles que ceux de *BSR*. En combinant les deux méthodes, le rappel monte à 0.87 après 35 tours



(a) Rappel, sans *churn*.

(b) Rappel avec *churn*.

(c) % de cache rempli sans *churn*.

(d) % de cache rempli avec *churn*.

Figure 5.6: Effet de la réplication sur le rappel et le niveau de remplissage moyen des caches.

de *gossip*. La Figure 5.6d montre que *QPR* ne consomme que 15% de l'espace de réplication. Ceci explique les faibles gains de rappel. *BSR* et la méthode hybride peuvent quant à elles exploiter jusqu'à plus de 90% de cet espace.

Des résultats similaires peuvent être atteints avec d'autres valeurs pour la taille de l'espace de stockage.

5.6 Travaux connexes

La recommandation et la recherche distribuées pour les données du web, s'appuyant sur des techniques de filtrage collaboratif ont fait l'objet de divers travaux tout en offrant de nombreuses perspectives de recherche. Cette section se propose de comparer ces différentes contributions avec notre approche.

Loupasakis et Ntarmos ont proposé [99] une approche décentralisée pour le réseautage social avec trois objectifs en tête : protection de la vie privée, rentabilité et disponibilité. Ils développent ainsi une architecture s'appuyant sur une *DHT* permettant la recherche à mots clés. Puisque les *DHT*s sont plus adaptées aux calculs de requêtes exactes (*i.e.* clé/valeur), les auteurs proposent de décomposer chaque requête en mots simples devenant ainsi chacun, des requêtes exactes. Cependant, le principal défaut de cette approche est que des requêtes offrant un score moyen pour chacun des mots clés, mais un score élevé lors de leur combinaison (*i.e.* requête complète) ont une forte probabilité d'être raté.

P2PRec [49] est une solution de recherche et de recommandation à base de *gossip* où le profil de chaque utilisateur u est représenté comme un ensemble de thèmes (*i.e. topics*) calculé à partir des objets qu'il partage. Ensuite, l'utilisation des protocoles de *gossip* permet de relier chaque utilisateur à ceux qui lui sont le plus similaires – étape dite de *regroupement* à ne pas confondre avec les approches *P2P* à base de *classification*. Ce dernier point est repris dans notre approche. Cependant, puisque que les auteurs ne tiennent pas compte de la diversité dans ce processus de *regroupement*, chaque *cluster* peut être composé d'utilisateurs redondants et le rappel reste limité.

Kermarrec *et al.* [81] se focalisent sur la recommandation et proposent de combiner les algorithmes de *gossip* et les marches aléatoires. Ainsi, chaque utilisateur crée son *cluster* d'utilisateurs similaires – comme *P2PRec* – et ce voisinage est exploité pour propager les marches aléatoires afin de calculer la matrice « *Users* $\times$ *Items* ». Ces marches aléatoires sont exécutées localement en utilisant une matrice transitoire de similarité dont la complexité de calculs dépend de la taille du voisinage. Cela réduit la capacité de passage à l'échelle de cette approche. Bien que les auteurs n'introduisent pas de diversité explicitement, les marches aléatoires peuvent être vues comme une manière d'atténuer l'effet d'un voisinage trop similaire.

Toujours dans l'optique d'accroître le rappel, Kermarrec *et al.* [82] affirment que puisque les utilisateurs sont hétérogènes, il est nécessaire d'introduire de l'hétérogénéité dans le calcul du voisinage de chacun. Cependant, contrairement à nous, l'idée proposée par les auteurs est que chaque utilisateur utilise une fonction de similarité optimisée pour son profil. Les *clusters* ainsi créés restent redondants et le rappel limité.

Bai *et al.* [21] proposent une solution pour le traitement de requêtes *top-k* en *P2P* dans le cadre d'un système collaboratif d'étiquetage, nommé *P4Q*. Dans cette

solution, les utilisateurs sont regroupés en fonction de leur similarité en utilisant des protocoles de *gossip*. Deux groupes sont alors définis : 1) les c plus proches de l'utilisateur u dont sont répliqués toutes les méta-données relatives aux objets qu'ils partagent (*i.e.* action d'étiquetage) et 2) les n utilisateurs moins proches de u dont seul le profil est connu. La diversité ici n'est pas prise en compte ce qui limite l'effet de réplication des données les plus similaires et n'aborde pas l'effet des *clusters* trop redondants. Dans notre approche, les méta-données répliquées devaient être complémentaires au profil de l'utilisateur u afin d'être combinées à la diversité.

5.7 Conclusion et perspectives de recherche

Dans ce chapitre, nous avons présenté une approche à base de *gossip* pour la recherche et la recommandation. De nouvelles mesures et techniques ont été introduites. Tout d'abord, l'*utilité*, en combinant pertinence et diversité, est une manière effective d'augmenter considérablement la couverture, et en conséquence, les niveaux de rappel.

Pour accroître encore plus la couverture de chaque pair, et donc, le rappel, nous avons proposé une stratégie de réplication : *BSR*. Lorsque *BSR* est combinée à *QPR*, elle permet également une augmentation rapide et non négligeable du rappel.

Nous avons validé nos propositions avec une évaluation expérimentale sur plusieurs jeux de données, montrant des gains très importants. La diversité seule permet d'obtenir des niveaux de rappel jusqu'à 3 fois supérieurs aux solutions non diversifiées. Quant à la réplication, celle-ci atteint un gain significatif du rappel permettant, avec de très faibles valeurs de TTL (*i.e.* 2), d'obtenir des niveaux de rappels proches de 1 sans *churn* et autour de $0,95\%$ avec *churn*.

Comme cela a été présenté en introduction (cf. Chapitre 1, en première page), les données de botanique et de phénotypage sont partagées sur un ensemble de sites hétérogènes (*i.e.* ordinateur personnel, serveur, *cloud*). Il convient donc d'enrichir la présente contribution avec une approche plus axée sur le multisite. Cela est présenté dans le prochain Chapitre.

Par ailleurs, des perspectives de recherche se dessinent également. Il peut être intéressant, par exemple, d'étudier l'effet général de la diversité sur les réseaux distribués en recherche et en recommandation, et non pas uniquement sur les techniques utilisant les protocoles de *gossip*. Un défi consisterait à maximiser la couverture global du *U-Net*, et non pas de maximiser itérativement la pertinence et la diversité de chaque pair.

CHAPITRE 6

Prototype multisite de recherche et de recommandation

Résumé. Ce chapitre présente nos contributions pour la recherche et recommandation multisites. Nous y introduisons à cette fin le concept de nœud virtuel ainsi qu'un protocole d'indexation distribué. L'architecture générique de notre approche y est présentée. Un prototype composé de deux versions a été réalisé. Les versions PlantRT *et* DocRT *adressent respectivement les cas d'application de la botanique et celui du phénotypage. Nous abordons en particulier un* use-case *et le déploiement de ce dernier. Nous évaluons expérimentalement nos contributions ; nous montrons notamment, qu'en tenant compte de la diversité – telle que présentée dans le chapitre précédent – et de la notion de nœud virtuel, il est possible d'obtenir des niveaux de rappels élevés tout en minimisant le TTL.*

6.1 Introduction

Dans la continuité des chapitres précédents, nous illustrons celui-ci par les cas d'application sur la botanique et sur les données de phénotypage. Rappelons que dans le premier cas, les utilisateurs produisent et souhaitent partager une grande quantité d'observations de plantes. Celles-ci sont représentées par des images de la plante, ainsi que par un ensemble de méta-données telles que la famille, le genre et l'espèce de la plante, mais aussi la position *GPS* de la plante, etc. Dans le second cas d'application, la communauté visée est celle des scientifiques. Ces derniers produisent une grande quantité de documents scientifiques qu'ils souhaitent partager.

Le chapitre précédent exploite l'idée de réseau *P2P* [49], où chaque utilisateur représente un nœud du réseau. Lorsqu'une requête q est soumise par un utili-

sateur, celle-ci est propagée entre les nœuds via un recouvrement. Pour plus de détails se référer à la Section 2.2 du Chapitre 2 sur l'état de l'art ou au Chapitre 5.

Nous proposons une solution générique multisite pour la recherche et la recommandation, où chaque site peut regrouper 1 à n utilisateurs (*e.g.* une plateforme dans le nuage peut ainsi représenter un site).

Par ailleurs, nous avons introduit au Chapitre 3 une méthode de recherche de recommandations diversifiée [149] qui, en plus de la pertinence, prend également en compte la diversité des objets et des profils des utilisateurs les partageant. Dans le contexte de la recommandation de données issues de la botanique, les profils sont déduits de leurs observation et la diversification permet, par exemple, de découvrir des plantes provenant de plusieurs espèces originales d'une même famille, ou d'une même zone géographique. Ceci est fondamental afin de comprendre correctement la biodiversité des plantes. Dans le cadre du phénotypage, les profils sont issus des documents scientifiques partagés par chaque scientifique. La diversité permet ici de recommander des données provenant de communautés pertinentes mais diverses afin de mettre en relation des jeux de données inattendus.

En résumé, nous proposons une solution générique multisite pour la recherche de recommandations diversifiées, où chaque site peut regrouper 1 à n utilisateurs (*e.g.* une plateforme dans le nuage peut ainsi représenter un site). Un prototype composé de deux versions est issu de cette approche générique :

- ***Plant Recommendation Tool* (*i.e.* PlantRT)** : cette version adresse le contexte de la botanique en permettant le partage d'observations de plantes ;
- ***Document Recommendation Tool* (*i.e.* DocRT)** : celle-ci adresse le contexte du phénotypage en permettant le partage de documents scientifiques.

Les deux versions s'adaptent notamment au données différentes issues des cas d'application.

Lorsque les sites sont composés de plusieurs utilisateurs, la personnalisation est calculée de la manière suivante : chacun possède son propre profil issu des données qu'il partage, et les utilisateurs similaires sont regroupés ensemble (*i.e. clustering*). Les groupes d'utilisateurs issus de ce *clustering* définissent l'ensemble des nœuds virtuels du site. Il s'agit d'une représentation logique d'un groupe d'utilisateurs similaires, pour lesquels un profil commun est créé, ainsi que quelques méta-données supplémentaires. Les nœuds virtuels sont utiles pour la création du recouvrement pour la recherche et la recommandation distribuées tout autant que pour l'indexation personnalisée et efficace des objets [9].

Dans ce chapitre, nous présentons en détails notre architecture multisite. Celle-ci est générique et peut être appliquée à plusieurs cas d'application. Notre prototype PlantRT a été déployé en utilisant un jeu de données fourni par *ImageCLEF* [74] et *Pl@ntNet* en collaboration avec *Tela Botanica*. Il contient plus

de 10 000 observations produites par 1 500 bénévoles. Afin d'évaluer le passage à l'échelle ainsi que le niveau rappel, tout autant que l'efficacité de nos protocoles, nous avons réalisé des simulations jusqu'à 100 sites. Les utilisateurs étaient répartis aléatoirement sur ces sites. À plus petite échelle, ce prototype a été instancié sur 5 machines virtuelles fournies par *Microsoft Azure*.

À propos de DocRT, nous disposions cette fois-ci d'environ 300 documents de biologie (*e.g.* génétique, phénotypage) mais aussi d'informatique, partagés par une trentaine de volontaires de ces différentes disciplines. Il a été déployé sur 3 machines du *LIRMM*.

À notre connaissance, il s'agit de la première démonstration d'une approche multisite utilisant de vraies données scientifiques.

En résumé, nous proposons les contributions suivantes :

1. le concept de nœud virtuel est introduit afin de construire un recouvrement entre sites hétérogènes étant donné une fonction de score personnalisée ;
2. un protocole d'indexation entièrement distribué est proposé ;
3. un cas d'utilisation et le déploiement du prototype sont discutés ;
4. enfin, afin de valider nos propositions, une évaluation expérimentale basée sur un jeu de données réel composé de 10 000 observations de botanique partagées par 1 500 utilisateurs est présentée. Celle-ci confirme l'intérêt des nœuds virtuels ainsi que les performances de notre protocole d'indexation.

La suite de ce chapitre est structurée de la manière suivante. Tout d'abord, la Section 6.2 introduit les concepts de base ainsi que le problème que nous abordons. La Section 6.3 présente en détail l'architecture de notre approche multisite. Ensuite, la Section 6.5 décrit les différents cas d'utilisation en prenant pour exemple PlantRT. La Section 6.6 discute de l'évaluation expérimentale de nos contributions. Enfin, la Section 6.8 conclut le chapitre.

6.2 Concepts de base et définition du problème

Cette section, contient les bases nécessaires pour comprendre le problème que nous adressons. La définition du problème est également introduite.

Dans notre approche de recherche et de recommandation distribuées, lorsqu'un utilisateur u soumet une requête q, celle-ci doit être transmise à un sous-ensemble de sites qui retourneront leurs résultats les plus pertinents en fonction de q et de u. Nous disposons donc d'un ensemble de sites $S = s_1, ..., s_p$ et d'utilisateurs $U = u_1, ..., u_n$. Chacun peut partager jusqu'à m objets $I = it_1, ..., it_m$ (*e.g.* observations, contenus scientifiques).

Un objet, ou contenu, est représenté de manière vectorielle [107, 141]. En utilisant $tf \times idf$, un objet est comme une liste de mots clés $k_1, ..., k_z$, et le vecteur représente le poids de chacun de ces derniers pour l'objet en question, étant donné le corpus global. Le profil d'un utilisateur exprime ses intérêts et est calculé à

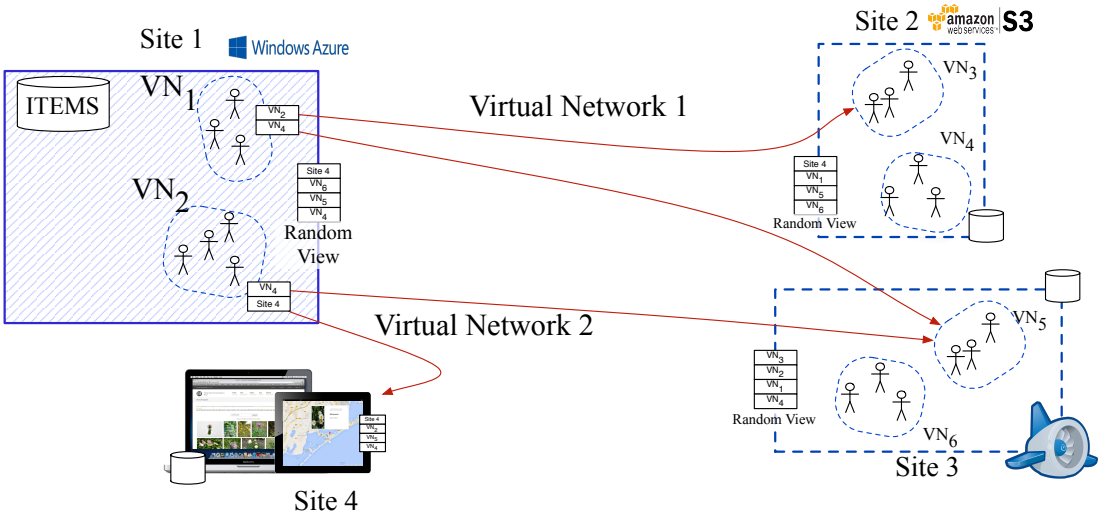


Figure 6.1: Exemple d'un réseau multisite avec 4 sites.

partir des objets qu'il partage I_u. De manière plus précise, le profil d'un utilisateur est la moyenne des vecteurs $tf \times idf$ des objets qu'il partage. Les requêtes sont exprimées, quant à elles, par une liste de mots clés $k_1, ..., k_z$.

Définition du problème : Étant donné un ensemble d'utilisateurs U, un ensemble d'objets I, de sites S et une requête à mots clés q soumise par un utilisateur u, le problème que nous adressons est le suivant : il s'agit de recommander efficacement à u les *top-k* objets les plus pertinents et divers $R^q \in I$ quelles que soient les infrastructures S où ils sont stockés.

6.3 Aperçu de l'architecture multisite

Cette section présente, en détail, l'architecture de notre approche multisite, illustrée en Figure 6.2. La sous-section 6.3.1 décrit tout d'abord les éléments relatifs au concept de nœud virtuel. La sous-section 6.3.2 expose ensuite comment la recherche et recommandation sont implémentées en multisite. Enfin, nous détaillons en sous-section 6.3.3 notre solution pour l'indexation distribuée des données.

Comme cela est présenté dans la Figure 6.1, notre solution multisite est composée d'un ensemble de sites hétérogènes $s_1, ..., s_p$ (*e.g.* serveur, PCs, *cloud*). Chacun héberge 1 à n nœuds virtuels faisant référence à un groupe de 1 à m utilisateurs au profil similaire. Ces derniers partagent des données dont le contenu, ou les méta-données, sont utilisés pour calculer leur profil [149]. Dans PlantRT, les observations de plantes sont associées à des méta-données indiquant la famille, l'espèce et le genre de la plante, ainsi que la position GPS de l'observation et une description. Dans DocRT, les documents scientifiques le sont à un titre, un

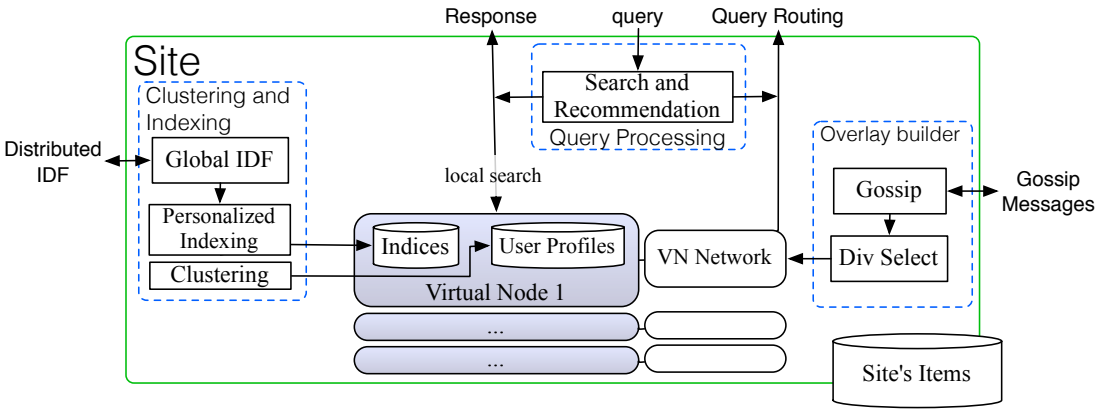


Figure 6.2: Architecture de recherche et de recommandation distribuée multisite.

ensemble d'étiquettes et au contenu textuel du document. Nous utilisons $tf \times idf$ (*i.e. Term Frequency* $\times$ *Inverse Document Frequency*) [141] afin de représenter chaque donnée, et chaque profil utilisateur.

D'un point de vue architectural, notre approche multisite possède trois composants (cf. Figure 6.2). Le premier, *Clustering and Indexing*, est responsable de la construction des nœuds virtuels mais aussi de leur index associé et personnalisé. Le second, *Overlay Builder*, est chargé de l'établissement du recouvrement entre les nœuds virtuels, parmi les différents sites $s_1, ..., s_p$. Finalement, le composant *Query Processing*, s'occupe de la recherche et recommandation.

Par la suite, nous utilisons I pour faire référence au corpus d'objets de tous les sites, et I_s pour le corpus local à un site s.

6.3.1 Nœud virtuel

Un élément clé de notre architecture est le *nœud virtuel*. Il s'agit d'une représentation logique d'un groupe d'utilisateurs similaires. Cela permet à la fois une indexation personnalisée des objets et la construction du recouvrement multisite utilisé par la suite pour guider les recommandations. Un nœud virtuel est défini comme suit :

$$VN_i = \{ID, Index_i, U_i, Profile_i, VN\text{-}Network_i\}$$

Où ID est l'identifiant unique du nœud, permettant de lui associer l'ensemble des structures qui lui sont attachées, telles que l'index. Lorsqu'une requête doit être exécutée, cet identifiant est par exemple utilisé pour déterminer le nœud qui en aura la charge. $Index_i$ est l'ensemble personnalisé des listes inversées de chaque nœud virtuel VN_i, calculées en fonction des utilisateurs U_i associés à ce dernier (*i.e.* VN_i), et des objets I_s. Comme cela est présenté dans les chapitres précédents, les listes inversées associent chaque mot clé du corpus à l'ensemble des documents le contenant. Celles-ci sont utilisées dans le calcul d'un *top-k* afin de récupérer

les résultats pertinents [7]. Le détail de ce *top-k* est présenté au Chapitre 2 sur l'état de l'art, en Section 9.

Une application directe d'un algorithme de *top-k*, lorsque le score est personnalisé consisterait à créer un ensemble de listes inversées pour chaque utilisateur, solution impossible en réalité (*i.e.* trop de mémoire/stockage serait nécessaire). Afin d'éviter cela, Amer-Yahia *et al.* [9] proposent que les utilisateurs similaires soient regroupés entre eux (*e.g. k-means* [104]) et que chaque groupe se voit assigner un ensemble de listes inversées : il s'agit pour nous des nœuds virtuels. Le score d'un objet dans une liste est finalement calculé de la manière suivante :

$$score(it, VN_i) = \max_{u \in U_i} score(it, u) \tag{6.1}$$

Où U_i correspond aux utilisateurs du nœud virtuel VN_i. En d'autres termes, le score d'un objet dans un groupe est le score maximum qu'il aurait pu obtenir étant donné chaque utilisateur de ce groupe. Au moment du traitement de la requête, le score exact de l'objet est calculé. Puisque celui-ci est nécessairement inférieur à son score dans la liste, il est toujours possible d'utiliser le seuil δ des algorithmes de *top-k*.

L'élément *Personalized Indexing* du composant *Clustering and Indexing* (cf. Figure 6.2) est responsable de la génération et de la mise à jour de ces listes inversées dynamiquement (*e.g.* lorsqu'un nouvel objet est partagé, les listes inversées dans lesquels il sera référencé sont mises à jour).

U_i fait ainsi référence aux utilisateurs similaires regroupés sur le nœud virtuel via *k-means* [104]. Rappelons que les profils des utilisateurs sont exprimés par un vecteur $tf \times idf$. Ainsi, $Profile_i$ représente la moyenne de ces vecteurs et constitue le profil du nœud virtuel. Dans la Figure 6.2, *k-means* est exécuté par l'élément *clustering*.

Enfin, chaque nœud virtuel vn_i est associé à une structure de données, appelée $VN\text{-}Network_i$ – dont le comportement est similaire au *User Network* du chapitre précédent –, faisant référence à l'ensemble divers de nœuds virtuels distants provenant des sites $s_1, ..., s_p$ dont vn_i a connaissance. Attention, ici la diversité est utilisé dans un sens similaire à celui du Chapitre 5 : il s'agit d'accroître la couverture de chaque $VN\text{-}Network_i$ afin de maximiser le rappel lorsqu'une requête est soumise. Pour plus de détails sur l'impact de la diversité sur le rappel dans les réseaux distribués, se référer au Chapitre 5. Chaque entrée de cette structure $VN\text{-}Network_i$ se compose des informations suivantes :

$$\langle ID, Profile_i, IP \rangle \tag{6.2}$$

Où *IP* est l'adresse du site auquel appartient le nœud. Dans la Figure 6.1, par exemple, VN_1 est connecté à deux autres nœuds virtuels provenant des sites 2 et 3.

6.3.2 Recherche et recommandation diversifiées

Lorsqu'une requête est soumise par un utilisateur sur un site, les différents *VN-Networks* sont utilisés afin de guider la recherche et la recommandation distribuées. Dans cette section, nous montrons comment les *VN-Networks* sont construits, maintenus et utilisés pour la recherche et la recommandation.

Notre approche s'appuie sur des échanges épidémiques (*i.e. gossip*) [75] entre sites afin, de propager les profils des différents nœuds virtuels présents sur le réseau. Ainsi, chaque site s_i maintient localement une vue aléatoire de nœuds disponibles sur le réseau, en utilisant le même format que décrit précédemment (cf. Équation 6.2). Périodiquement, deux sites donnés vont échanger un sous-ensemble de leur vue afin de la mettre à jour à partir de celui reçu (*i.e.* le sous-ensemble). Cela permet de découvrir de nouveaux nœuds virtuels provenant de sites distants. L'élément *Gossip* du composant *Overlay Builder* dans la Figure 6.2 est responsable de la vue aléatoire de même que des échanges de *gossip* entre les sites $s_1, ..., s_n$.

Après tout échange épidémique, le *VN-Network* de chaque nœud virtuel est mis à jour via un algorithme de *clustering* diversifié [150] (cf. Chapitre 5), que l'on nomme *Div Select*, dans la Figure 6.2. La diversification est ici utilisée dans l'optique d'accroître la couverture de chaque *VN-Network*; cela permet, lorsqu'une requête est soumise, d'obtenir des niveaux de rappel élevés (*i.e.* fraction des objets présents sur le réseau et pertinents pour une requête que le système est capable de récupérer). Celui-ci prend en entrée les nœuds de la vue aléatoire et du *VN-Network* actuel qu'il actualise en conséquence – rappelons que l'algorithme utilise la notion de liste des candidats.

Lorsqu'une requête q est soumise par un utilisateur u à son nœud d'appartenance, celui-ci la transmet à l'ensemble des nœuds virtuels présents dans son *VN-network*. L'ensemble des nœuds l'ayant reçue répète cette même opération jusqu'à ce qu'un *TTL* soit atteint.

Finalement, chaque nœud impliqué, incluant celui de u, retourne ses résultats les plus pertinents et divers, en fonction de q et de u, en utilisant un score [149] (cf. Chapitre 3). Attention, ici la notion de diversité est celle introduite dans le Chapitre 3. Il s'agit de la définition utilisée dans les modèles de recherche d'information ou de recommandation. Le calcul de ces résultats se fait par l'utilisation d'un *top-k* [7] sur $index_i$ (*i.e.* listes inversées).

6.3.3 Indexation $tf \times idf$ distribuée

Comme cela a été présenté dans la Section 6.3, les objets sont associés à un ensemble de mots clés. Ainsi, afin de les indexer dans des listes inversées, chacun d'eux est représenté par un vecteur $tf \times idf$ [141].

Nous présentons au sein de cette section notre solution pour le calcul de $tf \times idf$. Cette contribution est entièrement décentralisée et ne dépend donc d'aucune

Before		
Sites	*tdc_avg*	*dc_avg*
s	$\langle t_1, 2\rangle, \langle \boldsymbol{t_2, 1}\rangle, \langle \boldsymbol{t_3, 3}\rangle$	5
s'	$\langle t_1, 1\rangle, \langle t_2, 2\rangle, \langle t_4, 3\rangle$	8
After		
s	$\langle t_1, 2\rangle, \langle \boldsymbol{t_2, 1.5}\rangle, \langle \boldsymbol{t_3, 1.5}\rangle$	6.5
s'	$\langle t_1, 1\rangle, \langle \boldsymbol{t_2, 1.5}\rangle, \langle \boldsymbol{t_3, 1.5}\rangle, \langle t_4, 13\rangle$	6.5

Table 6.1: Exemple de l'état des structures utilisées pour le calcul d'IDF avant et après un échange du protocole *Distributed-IDF*

entité centralisée. Rappelons tout de même que *tf* × *idf* est un score définissant l'importance d'un mot pour un objet étant donné leur corpus global, qui, dans notre cas, est entièrement réparti sur les sites. L'idée sur laquelle repose notre approche est que seules des statistiques moyennes relatives à ce dernier sont nécessaires pour le calcul de ce score [160].

Plus précisément, le score *tf* × *idf* d'un terme (ou mot clé) $t \in it$, où it est un objet, est obtenu en multipliant la fréquence de ce terme au sein de it (TF) par l'inverse de sa fréquence dans le corpus global I (IDF) qui se calcule généralement de la manière suivante :

$$IDF(t, I) = log\frac{|I|}{F_{t/I}} \tag{6.3}$$

Où $F_{t/D}$ est le nombre d'objets du corpus contenant le terme t. La première partie de ce score peut être obtenue localement. La seconde, à l'inverse, s'appuyant sur le corpus global distribué, nécessite un algorithme distribué. À cette fin, nous nous appuyons sur un protocole épidémique (*i.e. gossip*).

Le but de notre approche distribuée est de calculer le nombre moyen d'objets par site, noté $|I|/n$, ainsi que pour chaque terme, le nombre moyen d'objets par site le contenant, noté $\frac{F_{t/D}}{n}$, où n représente la quantité totale de sites. Il suffit d'exploiter les propriétés de convergence des protocoles épidémiques [89] afin de déterminer ces moyennes qui sont ensuite utilisées dans le calcul d'IDF. La convergence signifie que le réseau va se stabiliser dans un état où les vecteurs IDF calculés seront proches de ce qu'ils auraient été en centralisés.

À cette fin, chaque site s maintient un compte du nombre de documents par termes :

$$tdc\_avg_s = \langle t_1, c_1\rangle, ..., \langle t_n, c_n\rangle \tag{6.4}$$

Où t_i est un terme et c_i est initialisé au nombre local de documents le contenant, mais converge vers la moyenne $\frac{F_{t_i/D}}{n}$. À cela s'ajoute une seconde structure maintenant le nombre moyen de documents par site, notée dc_avg_s. Celle-ci est initialisée au nombre local de documents et converge vers la moyenne par site.

Les Algorithmes 10 et 11 présentent les parties active (*i.e.* celui qui initie le protocole) et passive du protocole distribué du calcul d'IDF. Un échange est

initié périodiquement par chaque site s. L'objectif est de faire converger tdc_avg_s et dc_avg_s vers leur valeur moyenne au sein des sites. Un exemple d'exécution est présenté dans le Tableau 6.1.

Chaque échange est composé de trois étapes :

1. le site s initiant l'échange, commence en exécutant le premier pas de l'Algorithme 10. Il s'agit d'échantillonner le vecteur tdc_avg_s en un sous-ensemble appelé tdc_{sample} (Algorithme 10, ligne 2). L'échantillonnage est nécessaire car échanger systématiquement le vecteur tdc_avg_s serait trop lourd en termes de charge réseau. En d'autres mots, le vecteur tdc_avg_s ne sera jamais échangé complètement à chaque étape de *gossip*, mais seul un sous-ensemble aléatoire le sera. Dans le Tableau 6.1, l'échantillon de s est t_2 et t_3. Le site s choisit ensuite aléatoirement un autre site dans sa vue aléatoire, noté s' (Algorithme 10, ligne 3) auquel il envoie tdc_{sample} et dc_avg_s (Algorithme 10, lignes 4 et 5) ;
2. la seconde étape, décrite par l'Algorithme 11, est exécutée par le site s'. Après avoir reçu le message de s, s' va calculer la moyenne entre chaque terme de tdc_{sample} et ceux de sa structure $tdc\_avg_{s'}$, ainsi qu'entre dc_avg_s et $dc\_avg_{s'}$ (Algorithme 11, lignes 3, 4 et 6). Dans le Tableau 6.1, s' ne connaît pas le terme t_3, ce qui revient à supposer que son poids est de 0. Une fois la moyenne calculée, le site s' renvoie les résultats à s ;
3. la dernière étape est effectuée par le site s. Celle-ci consiste à sauvegarder localement les résultats du calcul de la moyenne de l'étape précédente (Algorithme 10, lignes 7 à 9).

Lorsque les différences entre l'état des structures avant et après un échange sont inférieures à un seuil pendant c cycles consécutifs (c est défini par le système), le site peut considérer que le vecteur IDF a convergé. Le vecteur peut donc être utilisé localement pour calculer des *top-k*. De manière plus précise, la condition d'arrêt se calcule comme suit :

$$\frac{\sum_{t \in tdc\_avg'} |tdc\_avg[t] - tdc\_avg'[t]|}{|tdc\_avg|} < \delta \tag{6.5}$$

Où tdc_avg est le vecteur avant l'échange et tdc_avg' est sa valeur après l'échange. δ est le seuil défini par le système.

6.4 Caractéristiques des versions du prototype

Cette section se propose de donner une idée des fonctionnalités implémentées qui distinguent PlantRT de DocRT. Ces dernières dépendent bien évidemment du cas d'application correspondant : les observations de plantes en botanique et les données de phénotypage en biologie. La sous-section 6.4.1 présente les fonctionnalités relatives à PlantRT et la sous-section 6.4.2, celles relatives à DocRT.

Algorithm 10: Initiator *Distributed-IDF*

Input: tdc_avg_s, dc_avg_s, Random *view*
Output: The *tdc* vector and *dc* will be updated with new values

repeat
  $tdc_{sample} \leftarrow subset(tdc\_avg_s)$;
  $v \leftarrow random\_selection(view)$;
  $message \leftarrow \langle tdc_{sample}, dc\_avg_s \rangle$;
  send *message* **to** v;
  receive $message'$ **from** v;
  for $t_i \in message'.tdc_{sample}$ **do**
    $tdc\_avg_s[t_i] \leftarrow message'.tdc_{sample}[t_i]$;
  $dc\_avg_s \leftarrow message'.dc\_avg_s$;
  wait *gossip period*;
until tdc_avg_s *and* dc_avg_s *has converged*;

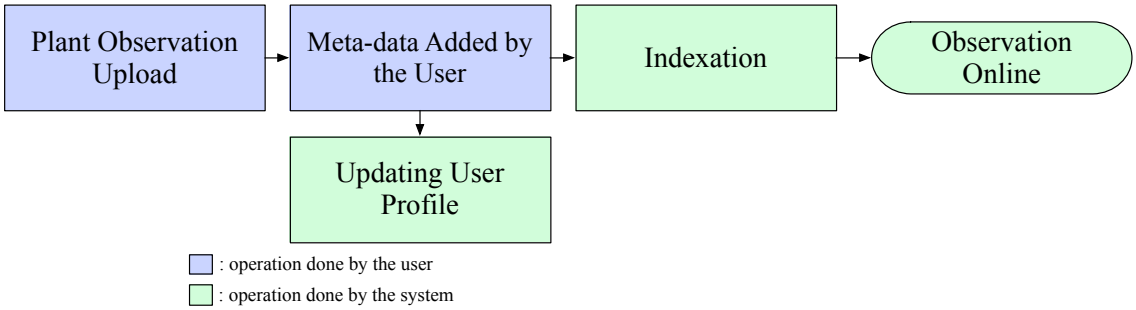


Figure 6.3: Processus d'indexation de PlantRT.

6.4.1 Caractéristiques de PlantRT

Type des données : il s'agit d'observations de plantes, et donc d'images. L'utilisateur a la possibilité d'accompagner ces dernières avec un ensemble méta-données qui sont : la famille, l'espèce et le genre de la plante, la position *GPS* de l'observation et le nom de la position (*e.g.* Montpellier) ainsi qu'une description.

Indexation : celle-ci suit le processus présenté en Figure 6.3. Cependant, deux points peuvent être mis en avant. L'ensemble des méta-données est indexé au niveau du document et du profil de l'utilisateur.

Recherche et recommandation : les utilisateurs peuvent soit soumettre des requêtes à mots clés, auquel cas, les observations sont choisies à partir du modèle présenté au Chapitre 3, soit soumettre leur position *GPS*. Dans ce dernier cas, le même modèle est appliqué, mais seules les observations faites dans un rayon

Algorithm 11: Recipient *Distributed-IDF*

Input: tdc_avg_v, dc_avg_v, Random *view*
Output: The *tdc* vector and *dc* will be updated with new values

```
1 repeat
2     receive message from u;
      /* message ← ⟨tdc_sample, dc_avg_s⟩                    */
3     for t_i ∈ message.tdc_sample do
4         tdc_avg_v[t_i] ← (message.tdc_sample[t_i] + tdc_avg_v[t_i]) / 2;
5         message.tdc_sample[t_i] ← tdc_avg_v[t_i];
6     dc_avg_v ← (dc_avg_v + message.dc_avg_s) / 2;
7     message.dc_avg_s ← dc_avg_v;
8     send message to u;
9 until ever;
```

$q = asteraceae$		
Un-diversified	Profile Diversity	
Élodie Dujardin	Marie Dupont	Pierre Durand
		
		
		

Table 6.2: Exemple de recherche et recommandation avec la diversité des profils.

défini par le système sont prises en compte. La position *GPS* est automatiquement calculée grâce aux *API* présentes dans les navigateurs Web modernes.

6.4.2 Caractéristiques de DocRT

Type des données : il s'agit de documents scientifiques. Ces derniers doivent être au format *PDF* et doivent contenir du texte, et non des images uniquement

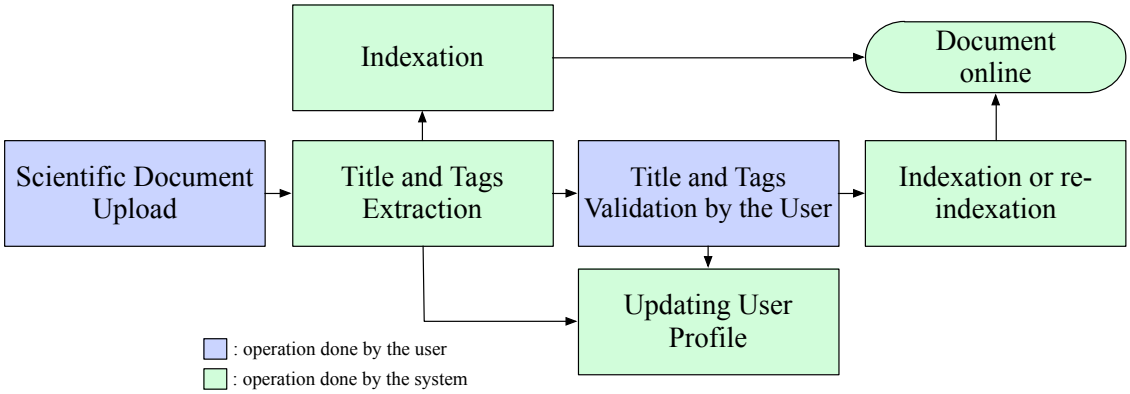


Figure 6.4: Processus d'indexation de DocRT.

(*e.g.* il ne peut pas s'agir d'un résultat de scan par exemple).

Indexation : celle-ci suit le processus présenté en Figure 6.3, qui est plus complexe que pour PlantRT. En effet, le système va, dans un premier temps, extraire le titre du document, son contenu et des étiquettes – calculées comme les mots ayant un score $tf \times idf$ le plus élevé. À ce stade, le document est ajouté au *buffer* pour indexation. L'utilisateur a cependant le choix de modifier les données de type *titre* ou *étiquette*. Dans le cas où il validerait une modification, le document est soit uniquement modifié dans le *buffer*, soit re-indexé, s'il l'avait déjà été entre-temps. Seules les étiquettes sont utilisées dans le calcul du profil de chaque utilisateur.

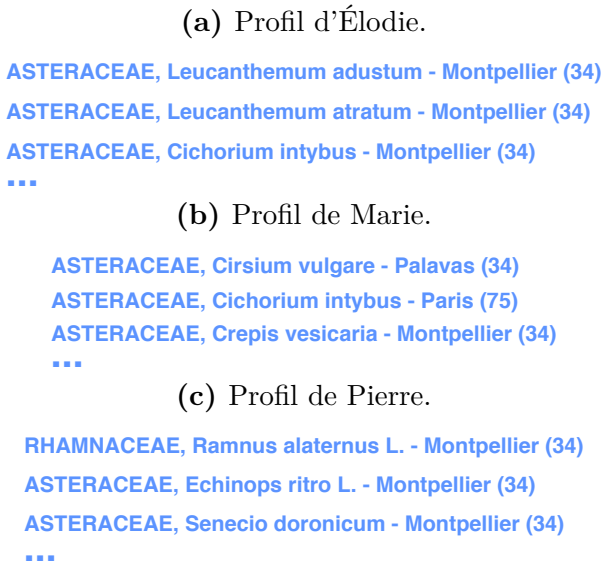
Recherche et recommandation : les utilisateurs peuvent soumettre des requêtes à mots clés et les documents sont choisis à partir du modèle présenté au Chapitre 3.

6.5 Zoom sur PlantRT

Cette section présente deux points importants de notre prototype PlantRT. Tout d'abord, plusieurs cas d'utilisation détaillés sont discutés en Section 6.5.1. Puis, en Section 6.5.2, sont fournis les détails relatifs au déploiement du prototype.

6.5.1 Cas d'utilisation de PlantRT

Rappelons que nous nous appuyons sur la recherche de recommandations diversifiée qui, lorsqu'une requête q est soumise, tient compte de q, du profil de son initiateur, de la diversification des contenus des objets ainsi que des profils des utilisateurs les partageant. Dans cette section, la notion de diversité est la même



(a) Profil d'Élodie.

(b) Profil de Marie.

(c) Profil de Pierre.

Figure 6.5: Les profils utilisateurs de l'exemple présenté en Tableau 6.2.

que celle introduite au Chapitre 3. Il s'agit donc de la diversité telle qu'utilisée en recherche d'information ou en recommandation.

Nous montrons ici le comportement de PlantRT au moyen d'une plateforme distribuée avec différents types de requêtes, soumises par des utilisateurs aux profils variés. La diversité permet aux utilisateurs de récupérer des objets (*i.e.* des observations de plantes), qui en plus d'être pertinents, sont également différents les uns des autres, améliorant ainsi la qualité des résultats. Ces objets sont également partagés par des utilisateurs aux profils similaires à celui de l'utilisateur courant (cf. Chapitre 3).

Le Tableau 6.2 présente les résultats obtenus lorsque la requête « asteraceae » – qui correspond à une grande famille de plantes, incluant la marguerite – a été exécutée sur notre jeu de données de botanique. Nous analysons les résultats de trois utilisateurs dont le profil est décrit dans la Figure 6.5 – ces derniers étant déduits des observations qu'ils ont soumises, chaque ligne correspond à une observation.

Chaque colonne du tableau représente les résultats obtenus par un utilisateur. Dans la première, correspondant à Élodie Dujardin, les résultats ne sont pas diversifiés. La méthode utilisée est simplement la similarité entre la requête et les objets indexés. Dans les deux autres, associées à Marie Dupont et à Pierre Durand, ils ont été obtenus en utilisant la diversification des profils [149] (cf. Chapitre 3).

La première colonne ne montre ainsi que des observations très redondantes, du fait de la non-diversification. D'un autre côté, la diversification a permis de récupérer un spectre large d'observations. Le premier résultat de Marie Dupont est par exemple une *Cirsium vulgare (Savi) Ten*, alors que le second provient de

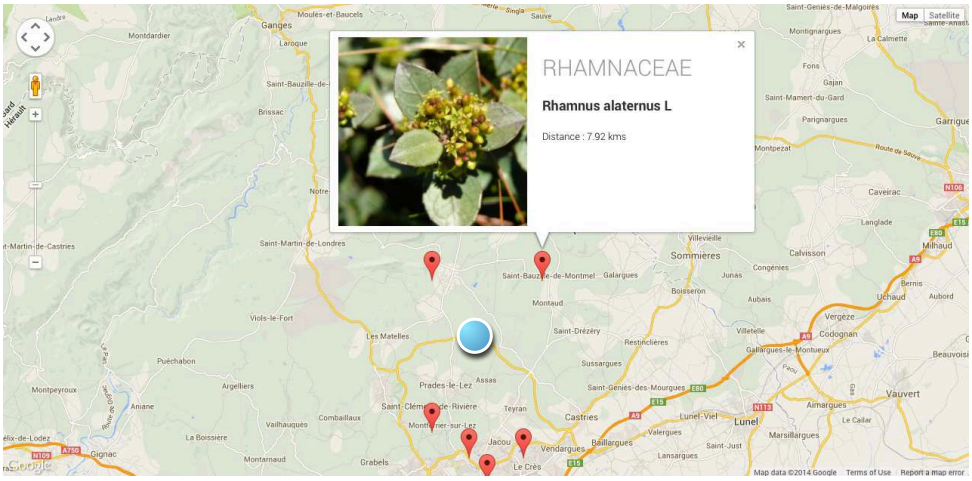


Figure 6.6: Exemple de géo-recommandation sur PlantRT autour de Montpellier, France.

l'espèce *Crepis vesicaria*. De même, le premier résultat renvoyé à Pierre est une plante de l'espèce *Senecio doronicum*, alors que le second est une *Echinops ritro*. Ces espèces peuvent également être observées dans les profils respectifs de chaque utilisateur dans la Figure 6.5.

Un second cas d'utilisation est le suivant. Un utilisateur u peut souhaiter retrouver des observations faites autour de lui, dans la même zone géographique. Dans ce scénario, la requête est constituée par sa position géographique, et les objets les plus pertinents sont les plus proches de u, mais également les plus divers : l'objectif est que l'utilisateur puisse avoir une idée de la biodiversité présente autour de lui, étant donné son profil (*i.e.* ses intérêts).

Dans la Figure 6.6, Pierre Durand s'est par exemple promené autour de Montpellier : il est représenté par le point bleu au centre, faisant donc référence à la requête $q = \langle latitude : 43.74 \rangle, \langle longitude : 3.9 \rangle$. Les pin's rouges indiquent les résultats de la requête. Il s'agit d'observations diverses faites dans la zone autour de u.

6.5.2 Déploiement de PlantRT

Cette section se propose d'introduire aux étapes nécessaires au déploiement de PlantRT sur une architecture multisite. Nous avons réalisé ce déploiement sur *Microsoft Azure*, où nous avons alloué 5 nœuds. Chacun d'entre eux possède les caractéristiques suivantes :

- **RAM :** $7GB$;
- **Nombre de cœurs :** 4 ;
- **Fréquence** CPU **:** $2,6Ghz$;

- **Taille disque :** environ $400GB$.

Microsoft Azure possède plusieurs centres de données, nommés en fonction de leur position géographique. Les nœuds PlantRT que nous avons déployés l'ont été sur les sites *north europe* et *east us*.

Une architecture possible avec *Microsoft Azure* est présentée en Figure 6.7. Chaque machine virtuelle est associée à un *cloud service*. Cela permet généralement, et par exemple si plusieurs machines virtuelles sont associées à un même *cloud service*, d'effectuer du *load balancing* (*i.e.* rediriger une requête web de l'utilisateur en fonction de la charge présente sur les machines virtuelles).

Chaque machine virtuelle d'un *cloud service* est associée à la même adresse *IP* publique. Lors de notre déploiement, le nombre de *cloud service* étant limité, chaque machine virtuelle lui appartenant devait en réalité représenter un site du réseau multisite. Pour cela, nous avons associé à chaque site un groupe de ports – pour le protocole de *gossip*, de traitement des requêtes, etc. – distincts des autres machines rattachées à son propre *cloud service* (*e.g. tomcat* utilise le port 8080 sur la VM_1 et 8081 sur la VM_2).

Le déploiement de notre approche se structure donc en deux étapes : construction d'une image qui servira de modèle à l'ensemble des machines virtuelles et instanciation de l'ensemble des machines virtuelles.

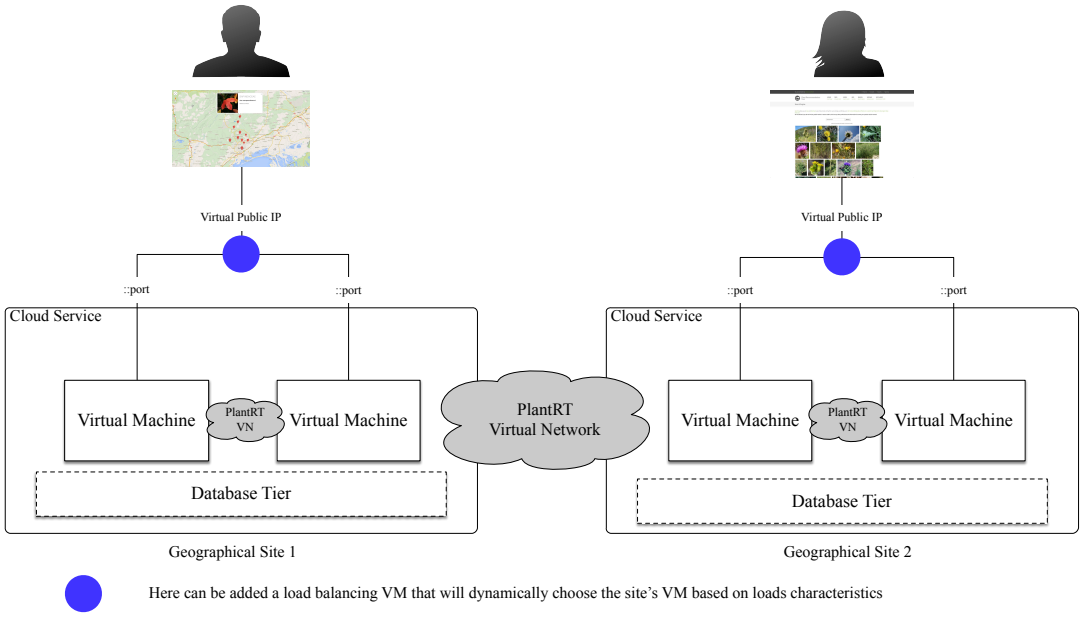


Figure 6.7: Architecture multisite sur *Microsoft Azure*.

Lors de la construction de l'image, deux applications doivent tout d'abord être installées : une base de données et un serveur d'application. La première est utilisée pour stocker des méta-données comme les profils utilisateurs ou la liste des documents partagés. Il s'agit dans notre cas de *MongoDB* [118], une base

de données orientée documents. Chaque utilisateur u est ainsi représenté par un document dans la base ; celui-ci regroupe la liste des documents partagés par u, son profil et toute autre information utile (*e.g.* adresse, date de naissance). Le serveur d'application, ici *tomcat* [18], permet d'exécuter chaque fonctionnalité de PlantRT dans un environnement web.

Afin de faciliter le déploiement du prototype, nous avons réalisé un script qui, en plus de démarrer les applications nécessaires dans le bon ordre (*i.e. mongodb* doit être démarré avant que *PlantRT* ne soit déployé dans *tomcat*), exploitent les *API* de *Microsoft Azure* pour détecter d'autres instances de PlantRT déjà démarrées dans le *cloud*. Plus précisément, il est possible de récupérer les caractéristiques de l'ensemble des machines virtuelles créées à partir de la commande suivante :

```
$ azure vm list --dns-name msproto-bota.cloudapp.net
  --json > $HOME/network.json
```

où l'attribut *dns-name* permet de cibler un *cloud service* donné, ici nommé *msproto-bota*. Cette commande renvoie au format `JSON` les caractéristiques de machines virtuelles ; il suffit de *parser*, puis de sélectionner une machine PlantRT, démarrée, différente de la machine locale, que le site courant pourra rejoindre pour former un réseau. *Microsoft Azure* permet facilement, à partir d'une machine virtuelle entièrement configurée, de créer une image (*i.e.* via la commande *capture*). Cette image est ensuite utilisée comme base à toutes les autres machines virtuelles.

Enfin, nous avons instancié 5 machines virtuelles à partir de l'image qui ont alors formé un réseau de partage PlantRT.

6.6 Évaluation expérimentale

Cette section présente l'évaluation expérimentale faite afin de valider les choix techniques proposés pour notre prototype multisite. La section 6.6.1 présente la mise en place des expériences dont les résultats sont discutés en section 6.6.2.

6.6.1 Mise en place des expériences

Bien que notre prototype aie été réellement déployé (*e.g.* dans le *cloud*), et afin de déployer nos algorithmes sur un grand nombre de nœuds, les expériences sont réalisées par des simulations. Nos expériences ont été menées sur le jeu de données de botanique, composé de 10 000 observations faites par 1 500 utilisateurs. Ces derniers sont répartis aléatoirement sur 100 sites, soit une moyenne de 15 utilisateurs par site.

L'objectif des expériences est de valider le concept de nœud virtuel et de réseau virtuel (*i.e. virtual network*), puis de confirmer le comportement de notre

protocole d'indexation distribué *Distributed-IDF* – les autres points (*e.g.* score de diversité des profils) ayant été validés aux chapitres précédents.

Pour ce faire, chaque utilisateur soumet un ensemble de requêtes afin de calculer les *top-10* objets les satisfaisant. Chacune est construite comme l'association aléatoire de mots clés extraits d'un objet donné. Dans notre cas, ces derniers correspondent au genre, à l'espèce ou à la famille d'une plante. Un résultat répond à la requête s'il contient l'ensemble de ces mots. Dans nos expériences, nous mesurons le rappel en fonction du nombre maximum de nœuds virtuels par site. Il s'agit de la même définition de rappel que celle discutée au Chapitre 5.

Le prototype effectue dans un premier temps 400 tours de *gossip* afin que le réseau virtuel converge. Cette valeur de 400 est choisie expérimentalement. Puis les requêtes sont soumises pendant 100 tours de *gossip*. Le résultat mesuré est le rappel moyen au cours de l'expérience. La notion de diversité utilisée dans la suite des expériences est celle introduite au Chapitre 5, c'est-à-dire celle permettant de construire le réseau virtuel via les protocoles de *gossip*.

Dans le but de confirmer notre protocole d'indexation distribué, nous mesurons l'erreur moyenne par termes, parmi les sites, du score IDF, à chaque tour de *gossip*. Cela nous permet d'observer la rapidité de convergence du protocole. Rappelons que la convergence est une caractéristique impliquant que le vecteur IDF de chaque site se stabilise vers une valeur proche de celle qui aurait été calculée en centralisé. Au cours de cette expérience, plusieurs tailles de messages de *gossip* sont comparées. Nous considérons qu'un tour de *gossip* est effectué toutes les 10 secondes.

6.6.2 Résultats expérimentaux

La première expérience, dont les résultats sont présentés dans la Figure 6.8, évalue l'effet des nœuds virtuels sur le rappel.

Deux comportements peuvent être observés. Tout d'abord, l'augmentation du nombre de nœuds virtuels accroît le niveau du rappel lorsque le réseau virtuel est diversifié. À l'inverse, lorsque la diversité n'est pas prise en compte, l'augmentation du nombre de nœuds virtuels entraîne une baisse du rappel. Cela peut s'expliquer à partir des conclusions du chapitre précédent où nous avons fait remarquer que le manque de diversité dans la construction d'un recouvrement entraîne une baisse du rappel. Ici, la diversité n'est pas prise en compte, mais regrouper les profils de plusieurs utilisateurs au sein d'un unique nœud virtuel peut être compris comme tel.

En second lieu, augmenter la taille du *virtual network*, c'est-à-dire du nombre de nœuds dont chacun est au courant, permet d'accroître le niveau du rappel. On peut observer que lorsque cette taille est fixée à 16 et que le nombre de nœuds virtuels et suffisamment élevé, les niveaux de rappel sont extrêmement proches de 1.

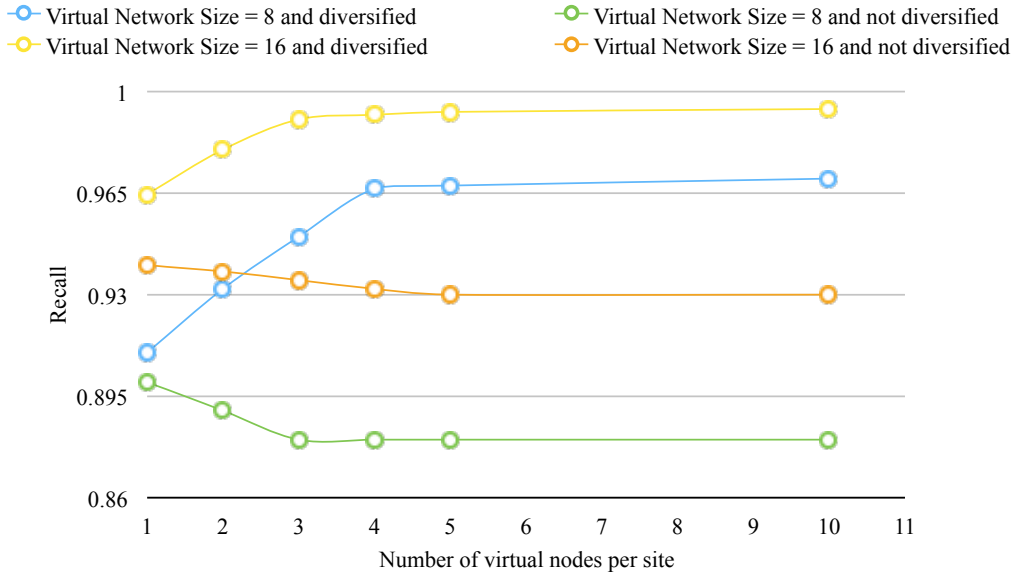


Figure 6.8: Niveau de rappel en fonction du nombre de nœuds virtuels, de la taille et du type de recouvrement (*i.e.* réseau virtuel).

La seconde expérience, dont les résultats sont présentés dans la Figure 6.9, montre le comportement de notre protocole d'indexation distribué. Nous avons fixé la taille de chaque message de *gossip* à 10, 50, 100 et 200. Sans surprise, la vitesse de convergence dépend de la taille des messages échangés. En effet, plus le nombre de termes (*i.e.* le poids associé au terme) échangés est important, plus la convergence sera rapide. Cependant, il peut être observé que même des tailles relativement réduites, telles que 10 ou 50, entraînent des temps de calcul très faibles. Par exemple, avec 50, une vingtaine de minutes est nécessaire au calcul du vecteur IDF. Cela confirme que notre protocole peut être effectivement déployé à grande échelle.

6.7 Travaux connexes

Les approches multisites ont été très récemment proposées comme un point intermédiaire entre les systèmes *P2P* et ceux centralisés. Il s'agit par exemple de combiner les possibilités de réduction de coût offertes par les premiers avec la fiabilité des seconds [20, 28, 35, 80].

Ces approches partent du postulat que le nombre de sites est réduit, leur disponibilité et leur fiabilité forte.

Certains travaux se focalisent sur la propagation des requêtes [19, 35, 58]. L'objectif est de déterminer dans quels cas doit-on transmettre ou pas une requête q d'un site s_1 à un site s_2 ? Si les *top-k* documents permettant de répondre à une

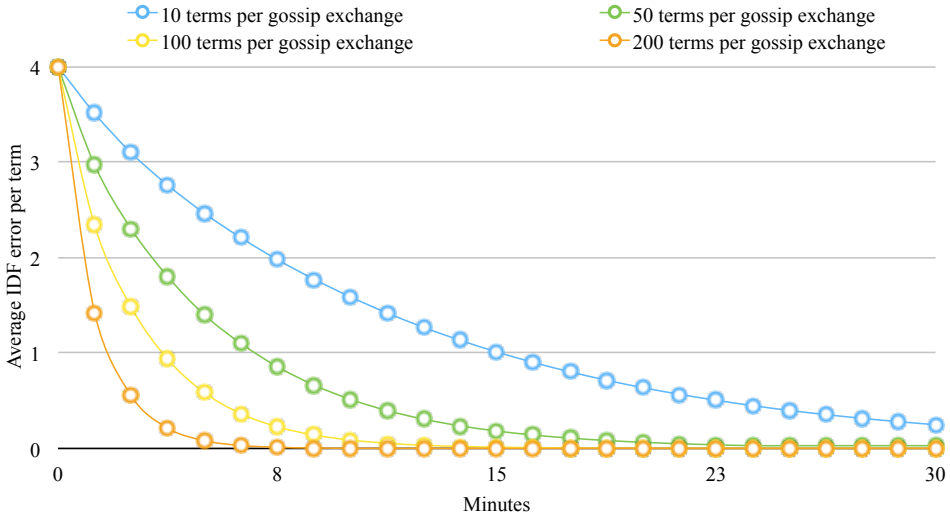


Figure 6.9: Convergence des valeurs IDF en fonction de la taille d'un message de *gossip*.

requête sont stockés et indexés sur le site s_1, alors il n'est pas utile d'envoyer la requête au site s_2. L'utilisation de seuil comme celui présenté dans le chapitre 2 ou des techniques de *machine learning* permettent ce choix.

Afin d'optimiser le calcul local d'une requête il est nécessaire de bien choisir les sites qui stockeront les documents [28]. En effet, puisqu'un objet donné ne doit pas être indexé par tous les sites, il doit se voir attribuer un *maître* (*i.e. master*) afin de toujours rester disponible si un site décide de le supprimer. L'objectif devient donc de définir un score pour chaque site en fonction d'un objet : $score(o, s_i)$. Une manière simple de définir ce score est d'évaluer la langue utilisée dans le contenu de l'objet et celles que les utilisateurs du site utilisent. Le maître sera le site où les utilisateurs parlant la langue de l'objet seront les plus nombreux. Une manière plus pertinente consiste à exploiter les documents déjà présents sur chaque site et les requêtes qui leur ont récemment été soumises afin d'évaluer la probabilité que le nouveau document intéresse les utilisateurs du site s_i. En d'autres termes, on s'intéresse à la probabilité que ce nouvel objet corresponde aux contenus déjà sur le site et répondant aux requêtes soumises par les utilisateurs de ce dernier.

Un autre point de travail est de maximiser le nombre de requêtes qui peuvent être traitées localement [80] dans le site d'appartenance des utilisateurs les soumettant en utilisant des techniques de réplication. Celle-ci doit se soumettre à deux contraintes : le nombre de réplicas d'un objet, au niveau global, ne doit pas être supérieur à un seuil prédéfini ; le nombre de réplicas présents sur un site ne doit pas excéder un seuil local prédéfini.

Cependant, ces approches supposent que les sites sont homogènes et ne prennent pas en compte la recommandation.

6.8 Conclusion

Ce chapitre présente donc une analyse d'une approche multisite, articulée autour d'un prototype composé de deux versions. PlantRT permet la recherche et recommandation pour les sciences citoyennes et, plus précisément, pour les botanistes. DocRT est une approche dont le but est la collaboration entre chercheurs partageant des documents scientifiques orientés sur les données de phénotypage.

Nous avons proposé une approche originale en utilisant le concept de nœuds virtuels. Cette dernière est décrite au sein d'une architecture générique et est illustrée par plusieurs cas d'utilisation. Enfin, une description du déploiement de notre prototype, instanciant cette approche, a été fournie.

PlantRT utilise un jeu de données composé de 10 000 observations faites par plus de 1 500 citoyens et son comportement a été validé expérimentalement. Nous montrons notamment les bénéfices dus aux nœuds virtuels et du réseau virtuel diversifié, permettant d'obtenir des niveaux de rappel proches des solutions centralisées. De la même manière, nous avons montré expérimentalement la convergence rapide de notre protocole d'indexation $tf \times idf$ distribué. DocRT a été déployé en interne parmi certains doctorants et ingénieurs de l'équipe Zenith et offre des temps de réponses très réduits à petite échelle.

Des impressions écran des deux versions du prototype sont disponibles en Annexe C.

Les codes des deux versions du prototype sont disponibles aux adresses suivantes :

- PlantRT : `http://www2.lirmm.fr/~servajean/prototypes/plant-sharing/plant-rt.html` ;
- DocRT : `http://www2.lirmm.fr/~servajean/prototypes/docs-sharing/doc-rt.html`.

CHAPITRE 7

Conclusion

Résumé. Ce chapitre synthétise les contributions présentées dans cette thèse et discute des perspectives de recherche qui en découlent.

7.1 Synthèse des contributions

Nous présentons dans cette section un résumé des contributions faites dans le cadre de cette thèse. Celles-ci sont au nombre de trois :

Diversité des profils : la première contribution se focalise sur la diversité et se situe dans un contexte centralisé. Nous introduisons tout d'abord l'idée de diversité des profils (cf. Chapitre 3, page 59) qui permet d'aborder les limitations dues à la mauvaise description des contenus ainsi qu'aux mots clés ambigus utilisés dans la description des objets. Nous adoptons ici l'idée de recherche de recommandations : les utilisateurs peuvent soumettre des requêtes à mots clés pour rechercher des objets partagés par des utilisateurs pertinents. La pertinence d'un utilisateur est évaluée étant donné l'idée de filtrage hybride (*i.e.* filtrage collaboratif combiné à celui basé sur les contenus). Les résultats doivent par ailleurs être divers à la fois en termes de contenu, mais également en fonction des profils des utilisateurs partageant les objets. Cette diversité des profils permet de limiter la redondance due aux descriptions incomplètes ou ambigües. Notre approche s'appuie sur un modèle probabiliste [17, 39] et permet d'adapter le niveau de diversification en fonction de *feedbacks* utilisateurs (*e.g.* si un utilisateur n'est pas satisfait des résultats présentés, les calculs futurs adapteront la diversité en conséquence). Nous avons proposé un ensemble d'algorithmes centralisés, s'appuyant sur des listes inversées (*i.e.* ces dernières associent chaque mot clé aux documents le contenant, triés par score décroissant, par exemple en utilisant la

similarité avec le mot clé donné) et sur la notion de liste de candidats afin d'être le plus efficace possible. En particulier, nous avons proposé des optimisations (cf. Chapitre 4, page 83) afin de limiter le nombre d'accès dans les listes inversées, le nombre de candidats à considérer et améliorer le calcul du score de chaque objet. Nos expériences montrent la préférence des utilisateurs pour la diversité des profils dans plus de 75% des cas. Par ailleurs, nos optimisations permettent des temps de réponse jusqu'à 12 fois plus faibles qu'un algorithme glouton. Cette contribution a été acceptée pour publication [149] dans *WWW Companion*, pour présentation [148] à *BDA'13* et pour publication en version enrichie [146] dans le journal *Information Systems*.

Diversité dans la construction d'un voisinage *P2P* : en second lieu, nous exploitons la diversité dans les réseaux *P2P* (cf. Chapitre 5, page 103). Nous proposons une nouvelle méthode de regroupement des utilisateurs (*i.e.* calcul du voisinage de chaque pair) basée sur un score combinant pertinence et diversité, nommé *usefulness*. Cette notion de diversité est une application différente de celle présentée précédemment, en centralisé. Alors que la diversité est traditionnellement utilisée dans les modèles de recherche et de recommandation – et c'est le cas pour notre contribution précédente –, nous appliquons ici cette notion dans l'étape de regroupement des pairs (*i.e.* calcul du voisinage de chaque pair) du réseau *P2P*. En diversifiant le voisinage de chaque pair, la probabilité qu'ils retournent les mêmes résultats à chaque requête soumise se réduit, et la capacité du système à retrouver des données pertinentes, étant donné une requête q, augmente. Nous montrons une stratégie de réplication accroissant encore plus la probabilité de retrouver des objets pertinents. Il en résulte que le score de regroupement diversifié permet d'obtenir des gains majeurs en termes de rappel (*i.e.* fraction des résultats pertinents que le système est capable de récupérer), jusqu'à 3 fois plus élevé que lorsque la diversité n'est pas prise en compte, tout en limitant les valeurs de *TTL* (*i.e. Time-To-Live*). La réplication permet rapidement d'obtenir des gains encore plus élevés. Les niveaux de rappel atteignent alors des valeurs quasi centralisées. Cette contribution a été acceptée pour publication [150] dans *Globe'14* et pour présentation [151] à *BDA'14*.

Plateformes multisites pour la recherche et la recommandation : en combinant les deux précédentes contributions, nous proposons une approche générique multisite pour sites hétérogènes (*e.g.* ordinateur personnel, serveur, *cloud*), permettant la recherche et la recommandation diversifiées (cf. Chapitre 6, page 125). Nous introduisons l'idée de nœud virtuel : il s'agit d'un regroupement d'utilisateurs similaires dans un site. Le voisinage de chaque site est réalisé via l'utilisation des nœuds virtuels en utilisant le score de *usefulness* précédemment introduit. Lorsqu'une requête est soumise, elle est propagée au travers de ce voisinage et les résultats retournés sont choisis via notre score incluant la diversité des profils.

Nous avons également proposé un algorithme entièrement distribué afin d'indexer correctement les objets dans des listes inversées, en tenant compte de statistiques globales à tous les sites (*e.g.* fréquence d'un mot dans l'ensemble des documents). Nos expériences montrent l'intérêt de la notion de nœuds virtuels ainsi que la capacité de passage à l'échelle de notre algorithme d'indexation distribué. Cette dernière contribution a été acceptée pour publication [152] à *BDA'14* et a fait l'objet d'un prototype composé de deux versions :

- PlantRT : `http://www2.lirmm.fr/~servajean/prototypes/plant-sharing/plant-rt.html` ;
- DocRT : `http://www2.lirmm.fr/~servajean/prototypes/documents-sharing/doc-rt.html`.

7.2 Perspectives de recherche

Les thématiques abordées au sein de cette thèse débouchent sur des perspectives de recherche qui constituent autant de défis. Quatre d'entre elles sont évoquées ci-bas.

7.2.1 Diversité et avis (*i.e. feedbacks*) des utilisateurs

Les problèmes de diversification montrent un fort attrait de la part de la communauté scientifique. En effet, la diversité permet d'accroître considérablement la satisfaction des utilisateurs lorsqu'ils se voient présenter des résultats. La diversité est souvent combinée à la pertinence et un poids permettant d'adapter le niveau de diversification. Ainsi, dans l'exemple suivant (*i.e. MMR*), λ est ce poids :

$$score(it, q|it_i, ..., it_{i-1}) = \arg\max_{it \in I \setminus R}[\lambda \times sim(it_i, q) - (1 - \lambda) \times \max_{it_j \in R}(sim(it_i, it_j))] \tag{7.1}$$

Dans l'équation suivante, il s'agit de ω :

$$score(it, q, R) = sim(it, q) \times \prod_{j \in 1,...,i-1} (1 - sim(it, it_j))^{\omega} \tag{7.2}$$

En déterminer la meilleure valeur est complexe. Nous avons abordé cette question au Chapitre 3, à la page 59, en utilisant les avis des utilisateurs. L'objectif consiste, étant donné l'ensemble des utilisateurs, des descriptions des objets et d'une requête q, à déterminer le meilleur poids de la diversité (*i.e.* la meilleure valeur de ω). Le problème est très proche des techniques de prédiction qui permettraient alors d'obtenir la meilleure satisfaction de l'utilisateur étant donné un objet. L'utilisation de plusieurs techniques issues du domaine de la recommandation, en détectant les corrélations entre mots clés et utilisateurs, permettrait alors d'affiner au maximum la valeur de ce poids, et donc, la satisfaction de l'utilisateur.

7.2.2 Diversité et *crowdsourcing*

En botanique, nous exploitons la recherche et la recommandation afin que les botanistes puissent rechercher et se voir recommander des observations des plantes qu'ils pourront ensuite identifier. Les plateformes de partage de botanique gagneraient à ce que ce nombre d'identifications soit le plus grand possible. Plusieurs points restent à exploiter à cette fin. Premièrement, surcharger certains experts d'observations que d'autres utilisateurs auraient pu identifier reviendrait à gaspiller les connaissances de plantes rares que ces experts ont : après avoir identifié un trop grand nombre de marguerites par exemple, l'expert peut refuser d'identifier une plante plus rare et laisser son identification à un autre jour.

Deuxièmement, il existe des outils automatiques d'identification de plantes (*e.g.* l'application Pl@ntNet) qui associent une liste possible de familles, d'espèces et de genres à une observation en fonction de sa ressemblance visuelle avec d'autres plantes de la base de données. L'expertise des utilisateurs devient utile lorsque l'application n'a pas réussi à identifier l'observation. Il peut être intéressant d'utiliser la liste des identifications possibles, calculées par l'application – même si aucune identification n'a été retenue – afin de rediriger l'observation vers les meilleurs utilisateurs (*i.e.* ceux qui ont la plus forte probabilité de pouvoir identifier la plante).

Troisièmement, plus les utilisateurs de ces plateformes de partage d'observations auront une connaissance de la biodiversité, plus il sera facile d'identifier des plantes. Pour cela, il peut être pertinent d'exploiter de la diversité afin de retourner un ensemble d'observations diverses et proches du profil de l'utilisateur, de manière à lui faire connaître de plus en plus de variétés de plantes.

7.2.3 Diversité et réseaux *P2P*

Nous avons introduit, au Chapitre 5, à la page 103, l'idée d'exploiter la diversité dans l'étape de construction du voisinage de chaque pair, afin de maximiser le rappel. Nous nous appuyons alors sur les protocoles de bavardage dans la construction de ce voisinage. Il peut être intéressant d'analyser l'impact de la diversité sur tous les types d'architectures et de topologies *P2P* de recherche d'information et de recommandation que nous avons présenté au Chapitre 2, à la page 13. Chacune de ces architectures et topologies utilisant des algorithmes qui leur sont propres pour la construction du voisinage, il conviendra alors de les adapter en tenant compte de la diversité.

7.2.4 Réplication et transmission de requêtes en multisite

Les approches multisites, en combinant les avantages des infrastructures centralisées et du *P2P*, deviennent de plus en plus populaires. Nous avons proposé une architecture permettant la collaboration de sites hétérogènes (*e.g.* ordina-

teurs personnels, serveurs). En proposant le concept de nœud virtuel, nous avons par exemple permis l'utilisation des techniques *P2P* de traitement des requêtes en les propageant sur le réseau de nœuds virtuels. Néanmoins, plusieurs contributions, en exploitant la grande capacité des sites – ou du moins de certains sites –, montrent qu'il est possible pour une majeure partie des requêtes, de les traiter localement. Ainsi, être capable de transmettre les requêtes uniquement lorsque cela est nécessaire, de répliquer les objets maximisant les calculs locaux à un site, et ce, lorsque le score est personnalisé, apparaît comme une perspective de recherche intéressante et particulièrement efficace pour réduire les temps de réponses et donc, améliorer la satisfaction des utilisateurs.

7.2.5 Recommandation de requêtes

La recommandation de requêtes est généralement abordée dans le cadre de la recherche ; l'utilisateur soumet déjà des requêtes, et le but est de lui en proposer d'autres, peut-être plus pertinentes. Dans un contexte où de nombreux documents textuels (*e.g.* contenus scientifiques, articles de journaux) sont présentés à l'utilisateur, être capable, en croisant le profil de l'utilisateur avec certains des contenus qu'il consulte, d'en extraire les thèmes et mots clés pertinents afin de construire et de lui recommander de nouvelles requêtes est un défi très intéressant qui augmenterait considérablement la satisfaction de ce dernier. En botanique, par exemple, cela permettrait à l'utilisateur de retrouver des plantes proches de celles qu'il consulte, tout en demeurant dans son champ de connaissance (*i.e.* proches de son profil).

Bibliographie

[1] S. Abbar, S. Amer-Yahia, P. Indyk et S. Mahabadi, « Real-Time Recommendation of Diverse Related Articles », dans *Proceedings of the 22nd International Conference on World Wide Web*, 2013, p. 1–12.

[2] Z. Abbassi, S. Amer-Yahia, L. V. Lakshmanan, S. Vassilvitskii et C. Yu, « Getting Recommender Systems to Think Outside The Box », dans *Proceedings of the Third ACM Conference on Recommender Systems*, 2009, p. 285–288.

[3] G. Adomavicius et 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*, t. 17, n° 6, p. 734–749, 2005.

[4] R. Agrawal, S. Gollapudi, A. Halverson et S. Ieong, « Diversifying Search Results », dans *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, 2009, p. 5–14.

[5] A. M. Ahmad Wasfi, « Collecting User Access Patterns for Building User Profiles and Collaborative Filtering », dans *Proceedings of the 4th International Conference on Intelligent User Interfaces*, 1998, p. 57–64.

[6] H. J. Ahn, « A New Similarity Measure for Collaborative Filtering to Alleviate the New User Cold-Starting Problem », *Information Sciences*, t. 178, n° 1, p. 37–51, 2008.

[7] R. Akbarinia, E. Pacitti et P. Valduriez, « Best Position Algorithms for Top-k Queries », dans *Proceedings of the 33rd International Conference on Very Large Data Bases*, 2007, p. 495–506.

[8] R. Akbarinia, M. Tlili, E. Pacitti, P. Valduriez et A. A. Lima, « Replication in DHTs Using Dynamic Groups », dans *Transactions on Large-Scale Data and Knowledge-Centered Systems III*, 2011, p. 1–19.

[9] S. Amer-Yahia, M. Benedikt, L. V. Lakshmanan et J. Stoyanovich, « Efficient Network Aware Search in Collaborative Tagging Sites », *Proceedings of the VLDB Endowment*, t. 1, n° 1, p. 710–721, 2008.

[10] S. Amer-Yahia, J. Huang et C. Yu, « Building Community-Centric Information Exploration Applications on Social Content Sites », dans *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, 2009, p. 947–952.

[11] S. Amer-Yahia, L. V. Lakshmanan, S. Vassilvitskii et C. Yu, « Battling Predictability and Overconcentration in Recommender Systems », *IEEE Data Engineering Bulletin*, t. 32, n° 4, p. 33–40, 2009.

[12] S. Amer-Yahia, L. Lakshmanan et C. Yu, « SocialScope : Enabling Information Discovery on Social Content Sites », dans *Conference on Innovative Data Systems Research*, 2009.

[13] S. Amer-Yahia, J. Shanmugasundaram, U. Srivastava, E. Vee et P. Bhat, *Efficient Online Computation of Diverse Query Results*, US Patent 8,001,117, 2011.

[14] S. Amer-Yahia et C. Yu, « Leveraging Communities in Social Content Sites », dans *Proceedings of the 2009 EDBT/ICDT Workshops*, 2009, p. 1.

[15] A. Anagnostopoulos, A. Z. Broder et D. Carmel, « Sampling Search-Engine Results », *World Wide Web*, t. 9, n° 4, p. 397–429, 2006.

[16] C. Anderson, *The Long Tail : Why the Future of Business Is Selling Less of More*. Hyperion, 2006.

[17] A. Angel et N. Koudas, « Efficient Diversity-Aware Search », dans *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, 2011, p. 781–792.

[18] *Apache Tomcat*, `http://tomcat.apache.org`, 2014.

[19] I. Arapakis, X. Bai et B. B. Cambazoglu, « Impact of Response Latency on User Behavior in Web Search », dans *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2014, p. 103–112.

[20] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Plachouras et L. Telloli, « On the Feasibility of Multi-Site Web Search Engines », dans *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, 2009, p. 425–434.

[21] X. Bai, R. Guerraoui, A.-M. Kermarrec et V. Leroy, « Collaborative Personalized Top-k Processing », *ACM Transactions on Database Systems*, t. 36, n° 4, p. 26, 2011.

[22] M. Balabanović et Y. Shoham, « Fab : Content-Based, Collaborative Recommendation », *Communications of the ACM*, t. 40, n° 3, p. 66–72, 1997.

[23] E Barnett, « Facebook Cuts Six Degrees of Separation to Four », *Telegraph*, 2011.

[24] M. Bawa, G. S. Manku et P. Raghavan, « SETS : Search Enhanced by Topic Segmentation », dans *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2003, p. 306–313.

[25] P. Bedi, H. Kaur et S. Marwaha, « Trust Based Recommender System for Semantic Web », dans *International Joint Conferences on Artificial Intelligence*, t. 7, 2007, p. 2677–2682.

[26] M. Bender, S. Michel, P. Triantafillou, G. Weikum et C. Zimmer, « Minerva : Collaborative P2P Search », dans *Proceedings of the 31st International Conference on Very Large Data Bases*, 2005, p. 1263–1266.

[27] D. Billsus et M. J. Pazzani, « Learning Collaborative Information Filters », dans *International Conference on Machine Learning*, t. 98, 1998, p. 46–54.

[28] R. Blanco, B. B. Cambazoglu, F. P. Junqueira, I. Kelly et V. Leroy, « Assigning Documents to Master Sites in Distributed Search », dans *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, 2011, p. 67–76.

[29] B. H. Bloom, « Space/Time Trade-Offs in Hash Coding with Allowable Errors », *Communications of the ACM*, t. 13, n° 7, p. 422–426, 1970.

[30] J. Bobadilla, F. Ortega, A. Hernando et J. Bernal, « A Collaborative Filtering Approach to Mitigate the New User Cold Start Problem », *Knowledge-Based Systems*, t. 26, p. 225–238, 2012.

[31] J. Bobadilla, F. Ortega, A. Hernando et A. Gutiérrez, « Recommender Systems Survey », *Knowledge-Based Systems*, t. 46, p. 109–132, 2013.

[32] N. Borch, « Social Peer-to-Peer for Social People », dans *The International Conference on Internet Technologies and Applications*, 2005.

[33] Y. Busnel et A.-M. Kermarrec, « PROXSEM : Interest-Based Proximity Measure to Improve Search Efficiency in P2P Systems », dans *European Conference on Universal Multiservice Networks*, 2007, p. 62–74.

[34] S. Büttcher, C. L. Clarke et G. V. Cormack, *Information Retrieval : Implementing and Evaluating Search Engines*. MIT Press, 2010.

[35] B. B. Cambazoglu, E. Varol, E. Kayaaslan, C. Aykanat et R. Baeza-Yates, « Query Forwarding in Geographically Distributed Search Engines », dans *Proceedings of the 33rd International ACM SIGIR Conference on Research and Sevelopment in Information Retrieval*, 2010, p. 90–97.

[36] J. Carbonell et J. Goldstein, « The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries », dans *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1998, p. 335–336.

[37] J. Carretero, F. Isaila, A.-M. Kermarrec, F. Taïani et J. M. Tirado, « Geology : Modular Georecommendation in Gossip-Based Social Networks », dans *Proceeding of the 32nd International Conference on Distributed Computing Systems*, 2012, p. 637–646.

[38] S. H. S. Chee, J. Han et K. Wang, « Rectree : an Efficient Collaborative Filtering Method », dans *Data Warehousing and Knowledge Discovery*, 2001, p. 141–151.

[39] H. Chen et D. R. Karger, « Less is More : Probabilistic Models for Retrieving Fewer Relevant Documents », dans *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006, p. 429–436.

[40] Z. Chen et T. Li, « Addressing Diverse User Preferences in SQL-Query-Result Navigation », dans *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, 2007, p. 641–652.

[41] P.-A. Chirita, W. Nejdl et O. Scurtu, « Knowing Where to Search : Personalized Search Strategies for Peers in P2P Networks. », dans *Workshop on Peer-to-Peer Information Retrieval*, 2004.

[42] V. Cholvi, P. Felber et E. Biersack, « Efficient Search in Unstructured Peer-to-Peer Networks », *European Transactions on Telecommunications*, t. 15, n° 6, p. 535–548, 2004.

[43] I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg et B. Wiley, « Protecting Free Expression Online with Freenet », *Internet Computing, IEEE*, t. 6, n° 1, p. 40–49, 2002.

[44] E. Cohen et S. Shenker, « Replication Strategies in Unstructured Peer-to-Peer Networks », dans *ACM SIGCOMM Computer Communication Review*, t. 32, 2002, p. 177–190.

[45] A. Crespo et H. Garcia-Molina, « Routing Indices for Peer-to-Peer Systems », dans *Proceeding of the 22nd International Conference on Distributed Computing Systems*, 2002, p. 23–32.

[46] A. Crespo et H. Garcia-Molina, « Semantic Overlay Networks for P2P Systems », dans *Agents and Peer-to-Peer Computing*, 2005, p. 1–13.

[47] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall et W. Vogels, « Dynamo : Amazon's Highly Available Key-Value Store », dans *ACM SIGOPS Operating Systems Review*, t. 41, 2007, p. 205–220.

[48] M. Deshpande et G. Karypis, « Item-Based Top-N Recommendation Algorithms », *ACM Transactions on Information Systems*, t. 22, n° 1, p. 143–177, 2004.

[49] F. Draidi, E. Pacitti, D. Parigot et G. Verger, « P2Prec : a Social-Based P2P Recommendation System », dans *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, 2011, p. 2593–2596.

[50] M. Drosou et E. Pitoura, « Disc Diversity : Result Diversification Based on Dissimilarity and Coverage », *Proceedings of the VLDB Endowment*, t. 6, n° 1, p. 13–24, 2012.

[51] R. O. Duda, P. E. Hart et D. G. Stork, *Pattern Classification*. John Wiley & Sons, 2012.

[52] K. El-Arini, G. Veda, D. Shahaf et C. Guestrin, « Turning Down the Noise in the Blogosphere », dans *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009, p. 289–298.

[53] M. El Dick, E. Pacitti et B. Kemme, « Flower-CDN : a Hybrid P2P Overlay for Efficient Query Processing in CDN », dans *Proceedings of the 12th International Conference on Extending Database Technology : Advances in Database Technology*, 2009, p. 427–438.

[54] R. Fagin, A. Lotem et M. Naor, « Optimal Aggregation Algorithms for Middleware », *Journal of Computer and System Sciences*, t. 66, n° 4, p. 614–656, 2003.

[55] A. Fast, D. Jensen et B. N. Levine, « Creating Social Networks to Improve Peer-to-Peer Networking », dans *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 2005, p. 568–573.

[56] Y. B. Fernández, J. J. Pazos Arias, M. L. Nores, A. G. Solla et M. R. Cabrer, « AVATAR : an Improved Solution for Personalized TV Based on Semantic Inference », *IEEE Transactions on Consumer Electronics*, t. 52, n° 1, p. 223–231, 2006.

[57] E. Feuerstein, P. A. Heiber, J. Martinez-Viademonte et R. Baeza-Yates, « New Stochastic Algorithms for Scheduling Ads in Sponsored Search », dans *Latin American Web Conference*, 2007, p. 22–31.

[58] G. Francès, X. Bai, B. B. Cambazoglu et R. Baeza-Yates, « Improving the Efficiency of Multi-Site Web Search Engines », dans *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, 2014, p. 3–12.

[59] M. Ge, C. Delgado-Battenfeld et D. Jannach, « Beyond Accuracy : Evaluating Recommender Systems by Coverage and Serendipity », dans *Proceedings of the 4th ACM Conference on Recommender systems*, 2010, p. 257–260.

[60] M. Gibson et I. A. Pirwani, « Algorithms for Dominating Set in Disk Graphs : Breaking the Long Barrier », dans *European Symposium on Algorithms*, 2010, p. 243–254.

[61] D. Goldberg, D. Nichols, B. M. Oki et D. Terry, « Using Collaborative Filtering to Weave an Information Tapestry », *Communications of the ACM*, t. 35, n° 12, p. 61–70, 1992.

[62] S. Gollapudi et A. Sharma, « An Axiomatic Approach for Result Diversification », dans *Proceedings of the 18th International Conference on World Wide Web*, 2009, p. 381–390.

[63] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy et J. Zahorjan, « Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload », dans *ACM SIGOPS Operating Systems Review*, t. 37, 2003, p. 314–329.

[64] M. M. Halldórsson, « A Still Better Performance Guarantee for Approximate Graph Coloring », *Information Processing Letters*, t. 45, n° 1, p. 19–23, 1993.

[65] J. Han, M. Kamber et J. Pei, *Data Mining : Concepts and Techniques*. Morgan kaufmann, 2006.

[66] J. Han et C. Moraga, « The Influence of the Sigmoid Function Parameters on the Speed of Backpropagation Learning », dans *International Workshop on Artificial Neural Networks*, 1995, p. 195–201.

[67] P. Han, B. Xie, F. Yang et R. Shen, « A Scalable P2P Recommender System Based on Distributed Collaborative Filtering », *Expert Systems with Applications*, t. 27, n° 2, p. 203–210, 2004.

[68] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite et C. Kadie, « Dependency Networks for Inference, Collaborative Filtering, and Data Visualization », *The Journal of Machine Learning Research*, t. 1, p. 49–75, 2001.

[69] T. Hofmann et D Hartmann, « Collaborative Filtering with Privacy via Factor Analysis », dans *Proceedings of the 2005 ACM Symposium on Applied Computing*, 2005, p. 791–795.

[70] Z. Huang, H. Chen et D. Zeng, « Applying Associative Retrieval Techniques to Alleviate the Sparsity Problem in Collaborative Filtering », *ACM Transactions on Information Systems*, t. 22, n° 1, p. 116–142, 2004.

[71] A. Iamnitchi et I. Foster, « Interest-Aware Information Dissemination in Small-World Communities », dans *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing*, 2005, p. 167–175.

[72] A. Iamnitchi, M. Ripeanu et I. Foster, « Locating Data in (Small-World ?) Peer-to-Peer Scientific Collaborations », dans *Peer-to-Peer Systems*, 2002, p. 232–241.

[73] L. Iaquinta, M. De Gemmis, P. Lops, G. Semeraro, M. Filannino et P. Molino, « Introducing Serendipity in a Content-Based Recommender System », dans *Proceeding of the 8th International Conference on Hybrid Intelligent Systems*, 2008, p. 168–173.

[74] *ImageCLEF - The CLEF Cross Language Image Retrieval Track*, `http://www.imageclef.org`, 2014.

[75] M. Jelasity et O. Babaoglu, « T-Man : Gossip-Based Overlay Topology Management », dans *Engineering Self-Organising Systems*, 2006, p. 1–15.

[76] H. Jin, X. Ning et H. Chen, « Efficient Search for Peer-to-Peer Information Retrieval Using Semantic Small World », dans *Proceedings of the 15th International Conference on World Wide Web*, 2006, p. 1003–1004.

[77] A Joly, H Goëau, P Bonnet, B Vera, J Barbe, S Souheil, Y Itheri, J Carré, E Mouysset, J. F. Molino, N Boujemaa et D Barthélémy, « Interactive Plant Identification Based on Social Image Data », *Ecological Informatics*, t. 23, nº 0, p. 22–34, 2013, Special Issue on Multimedia in Ecology and Environment.

[78] V. Kalogeraki, D. Gunopulos et D. Zeinalipour-Yazti, « A Local Search Mechanism for Peer-to-Peer Networks », dans *Proceedings of the 11th International Conference on Information and Knowledge Management*, 2002, p. 300–307.

[79] G. Karypis, « Evaluation of Item-Based Top-N Recommendation Algorithms », dans *Proceedings of the 10th International Conference on Information and Knowledge Management*, 2001, p. 247–254.

[80] E. Kayaaslan, B. B. Cambazoglu et C. Aykanat, « Document Replication Strategies for Geographically Distributed Web Search Engines », *Information Processing and Management*, t. 49, nº 1, p. 51–66, 2013.

[81] A.-M. Kermarrec, V. Leroy, A. Moin et C. Thraves, « Application of Random Walks to Decentralized Recommender Systems », dans *Principles of Distributed Systems*, 2010, p. 48–63.

[82] A.-M. Kermarrec et F. Taïani, « Diverging Towards the Common Good : Heterogeneous Self-Organisation in Decentralised Recommenders », dans *Proceedings of the 5th Workshop on Social Network Systems*, 2012, p. 1.

[83] A.-M. Kermarrec et M. Van Steen, « Gossiping in Distributed Systems », *ACM SIGOPS Operating Systems Review*, t. 41, n° 5, p. 2–7, 2007.

[84] J. Kim et E. Chan-Tin, « Robust Object Replication in a DHT Ring », 2007.

[85] M. Kim et V. V. Raghavan, « Adaptive Concept-Based Retrieval Using a Neural Network », dans *Proceedings of the 23th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2000.

[86] I. A. Klampanos et J. M. Jose, « An Architecture for Information Retrieval over Semi-Collaborating Peer-to-Peer Networks », dans *Proceedings of the 2004 ACM Symposium on Applied Computing*, 2004, p. 1078–1083.

[87] M. Kochen, *The Small World*. Ablex Norwood, NJ, 1989.

[88] G. Koutrika, B. Bercovitz et H. Garcia-Molina, « FlexRecs : Expressing and Combining Flexible Recommendations », dans *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, 2009, p. 745–758.

[89] W. Kowalczyk, M. Jelasity et A Eiben, « Towards Data Mining in Large and Fully Distributed Peer-to-Peer Overlay Networks », dans *Proceedings of the 15th Benelux Conference on Artificial Intelligence*, 2003, p. 203–210.

[90] S. R. Kruk, S. Decker, A. Gzella et S. Grzonkowski, « Social Semantic Collaborative Filtering for Digital Libraries », *Journal of Digital Information, Special Issue on Personalization*, 2006.

[91] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal et R. Krishnapuram, « A Hierarchical Monothetic Document Clustering Algorithm for Summarization and Browsing Search Results », dans *Proceedings of the 13th International Conference on World Wide Web*, 2004, p. 658–665.

[92] X. N. Lam, T. Vu, T. D. Le et A. D. Duong, « Addressing Cold-Start Problem in Recommendation Systems », dans *Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication*, 2008, p. 208–211.

[93] N. Lathia, S. Hailes et L. Capra, « Trust-Based Collaborative Filtering », dans *Trust Management II*, 2008, p. 119–134.

[94] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. R. Karger et R. Morris, « On the Feasibility of Peer-to-Peer Web Indexing and Search », dans *Peer-to-Peer Systems II*, 2003, p. 207–215.

[95] Q. Li et B. M. Kim, « An Approach for Combining Content-Based and Collaborative Filters », dans *Proceedings of the 6th International Workshop on Information Retrieval with Asian Languages*, 2003, p. 17–24.

[96] G. Linden, B. Smith et J. York, « Amazon.com Recommendations : Item-to-Item Collaborative Filtering », *Internet Computing*, t. 7, n° 1, p. 76–80, 2003.

[97] F. Liu, F. Ma, M. Li et L. Huang, « Distributed Information Retrieval Based on Hierarchical Semantic Overlay Network », dans *Grid and Cooperative Computing*, 2004, p. 657–664.

[98] P. Lops, M. De Gemmis et G. Semeraro, « Content-Based Recommender Systems : State of the Art and Trends », dans *Recommender Systems Handbook*, 2011, p. 73–105.

[99] A. Loupasakis, N. Ntarmos, P. Triantafillou et D. Makreshanski, « eXO : Decentralized Autonomous Scalable Social Networking », dans *Proceeding of the 2011 Conference on Innovative Data Systems Research*, 2011, p. 85–95.

[100] Q. Lv, S. Ratnasamy et S. Shenker, « Can Heterogeneity Make Gnutella Scalable ? », dans *Peer-to-Peer Systems*, 2002, p. 94–103.

[101] H. Ma, I. King et M. R. Lyu, « Effective Missing Data Prediction for Collaborative Filtering », dans *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007, p. 39–46.

[102] H. Ma, H. Yang, M. R. Lyu et I. King, « SoRec : Social Recommendation Using Probabilistic Matrix Factorization », dans *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, 2008, p. 931–940.

[103] H. Ma, D. Zhou, C. Liu, M. R. Lyu et I. King, « Recommender Systems with Social Regularization », dans *Proceedings of the 4th ACM International Conference on Web Search and Data Mining*, 2011, p. 287–296.

[104] J. MacQueen et al., « Some Methods for Classification and Analysis of Multivariate Observations », dans *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, t. 1, 1967, p. 281–297.

[105] B. Magnini et C. Strapparava, « Experiments in Word Domain Disambiguation for Parallel Texts », dans *Proceedings of the Association for Computational Linguistics Workshop on Word Senses and Multi-Linguality*, 2000, p. 27–33.

[106] D. Malkhi, M. Naor et D. Ratajczak, « Viceroy : A Scalable and Dynamic Emulation of the Butterfly », dans *Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing*, 2002, p. 183–192.

[107] C. D. Manning, P. Raghavan et H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008, t. 1.

[108] M. E. Maron et J. L. Kuhns, « On Relevance, Probabilistic Indexing and Information Retrieval », *Journal of the ACM*, t. 7, n° 3, p. 216–244, 1960.

[109] S. Marti, P. Ganesan et H. Garcia-Molina, « SPROUT : P2P Routing with Social Networks », dans *Current Trends in Database Technology-EDBT Workshops*, 2004, p. 425–435.

[110] P. Massa et P. Avesani, « Trust-Aware Recommender Systems », dans *Proceedings of the 2007 ACM Conference on Recommender Systems*, 2007, p. 17–24.

[111] P. Maymounkov et D. Mazieres, « Kademlia : A Peer-to-Peer Information System Based on the XOR Metric », dans *Peer-to-Peer Systems*, 2002, p. 53–65.

[112] S. M. McNee, J. Riedl et J. A. Konstan, « Being Accurate is not Enough : How Accuracy Metrics Have Hurt Recommender Systems », dans *International Conference of Human-Computer Interaction*, 2006, p. 1097–1101.

[113] D. A. Menascé et L. Kanchanapalli, « Probabilistic Scalable P2P Resource Location Services », *ACM SIGMETRICS Performance Evaluation Review*, t. 30, n° 2, p. 48–58, 2002.

[114] S. Milgram, « The Small World Problem », *Psychology Today*, t. 2, n° 1, p. 60–67, 1967.

[115] B. N. Miller, J. A. Konstan et J. Riedl, « PocketLens : Toward a Personal Recommender System », *ACM Transactions on Information Systems*, t. 22, n° 3, p. 437–476, 2004.

[116] T. Miranda, M. Claypool, A. Gokhale, T. Mir, P. Murnikov, D. Netes et M. Sartin, « Combining Content-Based and Collaborative Filters in an Online Newspaper », dans *Proceedings of ACM SIGIR Workshop on Recommender Systems*, 1999.

[117] K. Miyahara et M. J. Pazzani, « Collaborative Filtering with the Simple Bayesian Classifier », dans *Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence*, 2000, p. 679–689.

[118] *MongoDB*, `http://www.mongodb.org`, 2014.

[119] S. A. Munson, D. X. Zhou et P. Resnick, « Sidelines : An Algorithm for Increasing Diversity in News and Opinion Aggregators. », dans *Proceeding of the 4th International Conference on Web and Social Media*, 2009.

[120] J. Noel, S. Sanner, K.-N. Tran, P. Christen, L. Xie, E. V. Bonilla, E. Abbasnejad et N. Della Penna, « New Objective Functions for Social Collaborative Filtering », dans *Proceedings of the 21st International Conference on World Wide Web*, 2012, p. 859–868.

[121] M. O'Connor et J. Herlocker, « Clustering Items for Collaborative Filtering », dans *Proceedings of the ACM SIGIR Workshop on Recommender Systems*, t. 128, 1999.

[122] Y. Ogawa, T. Morita et K. Kobayashi, « A Fuzzy Document Retrieval System Using the Keyword Connection Matrix and a Learning Method », *Fuzzy Sets and Systems*, t. 39, n° 2, p. 163–179, 1991.

[123] M. T. Özsu et P. Valduriez, *Principles of Distributed Database Systems*. Springer, 2011.

[124] E. Pacitti, R. Akbarinia et M. El-Dick, « P2P Techniques for Decentralized Applications », *Synthesis Lectures on Data Management*, t. 4, n° 3, p. 1–104, 2012.

[125] M. Papagelis, D. Plexousakis et T. Kutsuras, « Alleviating the Sparsity Problem of Collaborative Filtering Using Trust Inferences », dans *Trust Management*, 2005, p. 224–239.

[126] M. J. Pazzani et D. Billsus, « Content-Based Recommendation Systems », dans *The Adaptive Web*, 2007, p. 325–341.

[127] M. J. Pazzani, J. Muramatsu, D. Billsus et al., « Syskill & Webert : Identifying Interesting Web Sites », dans *Association for the Advancement of Artificial Intelligence*, t. 1, 1996, p. 54–61.

[128] M. Pazzani et D. Billsus, « Learning and Revising User Profiles : the Identification of Interesting Web Sites », *Machine Learning*, t. 27, n° 3, p. 313–331, 1997.

[129] J. Pearl, *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[130] *Pl@ntNet*, `http://www.plantnet-project.org`, 2014.

[131] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. Epema, M. Reinders, M. R. Van Steen et H. J. Sips, « TRIBLER : a Social-Based Peer-to-Peer System », *Concurrency and Computation : Practice and Experience*, t. 20, n° 2, p. 127–138, 2008.

[132] J. R. Quinlan, « Induction of Decision Trees », *Machine Learning*, t. 1, n° 1, p. 81–106, 1986.

[133] F. Radlinski et S. Dumais, « Improving Personalized Web Search Using Result Diversification », dans *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006, p. 691–692.

[134] S. Ratnasamy, I. Stoica et S. Shenker, « Routing Algorithms for DHTs : Some Open Questions », dans *Peer-to-Peer Systems*, 2002, p. 45–52.

[135] J. D. M. Rennie et N. Srebro, « Fast Maximum Margin Matrix Factorization for Collaborative Prediction », dans *Proceedings of the 22nd International Conference on Machine Learning*, 2005, p. 713–719.

[136] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom et J. Riedl, « GroupLens : an Open Architecture for Collaborative Filtering of Netnews », dans *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, 1994, p. 175–186.

[137] S. C. Rhea et J. Kubiatowicz, « Probabilistic Location and Routing », dans *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, t. 3, 2002, p. 1248–1257.

[138] J. Risson et T. Moors, « Survey of Research Towards Robust Peer-to-Peer Networks : Search Methods », *Computer Networks*, t. 50, n° 17, p. 3485–3521, 2006.

[139] A. Rowstron et P. Druschel, « Pastry : Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems », dans *Middleware 2001*, 2001, p. 329–350.

[140] O. D. Sahin, F. Emekçi, D. Agrawal et A. El Abbadi, « Content-Based Similarity Search over Peer-to-Peer Systems », dans *Databases, Information Systems, and Peer-to-Peer Computing*, 2005, p. 61–78.

[141] G. Salton, A. Wong et C.-S. Yang, « A Vector Space Model for Automatic Indexing », *Communications of the ACM*, t. 18, n° 11, p. 613–620, 1975.

[142] R. L. Santos, J. Peng, C. Macdonald et I. Ounis, « Explicit Search Result Diversification Through Sub-Queries », dans *Advances in Information Retrieval*, 2010, p. 87–99.

[143] S. Saroiu, P. K. Gummadi et S. D. Gribble, « Measurement Study of Peer-to-Peer File Sharing Systems », dans *Electronic Imaging 2002*, 2001, p. 156–170.

[144] B. M. Sarwar, G. Karypis, J. Konstan et J. Riedl, « Item-Based Collaborative Filtering Recommendation Algorithms », dans *Proceedings of the 10th International Conference on World Wide Web*, 2001, p. 285–295.

[145] B. M. Sarwar, G. Karypis, J. Konstan et J. Riedl, « Recommender Systems for Large-Scale e-Commerce : Scalable Neighborhood Formation Using Clustering », dans *Proceedings of the 5th International Conference on Computer and Information Technology*, t. 1, 2002.

[146] M. Servajean, R. Akbarinia, E. Pacitti et S. Amer-Yahia, « Profile Diversity for Query Processing using User Recommendations », *Information Systems (à paraître), 20 pages*, 2014.

[147] M. Servajean, E. Pacitti, S. Amer-Yahia et A. El Abbadi, « Increasing Coverage in Distributed Search and Recommendation with Profile Diversity », à soumettre.

[148] M. Servajean, E. Pacitti, S. Amer-Yahia et P. Neveu, « Phenotyping Data Search and Recommendation », dans *Bases de Données Avancées*, 2013.

[149] M. Servajean, E. Pacitti, S. Amer-Yahia et P. Neveu, « Profile Diversity in Search and Recommendation », dans *Proceedings of the 22nd International Conference on World Wide Web Companion*, 2013, p. 973–980.

[150] M. Servajean, E. Pacitti, M. Liroz-Gistau, S. Amer-Yahia et A. El Abbadi, « Exploiting Diversification in Gossip-Based Recommendation », dans *Proceeding of the 7th International Conference on Data Management in Cloud, Grid and P2P Systems*, 2014, p. 25–36.

[151] M. Servajean, E. Pacitti, M. Liroz-Gistau, S. Amer-Yahia et A. El Abbadi, « Exploiting Diversity in Distributed Recommendation », dans *Bases de Données Avancées*, 2014.

[152] M. Servajean, E. Pacitti, M. Liroz-Gistau, A. Joly et J. Champ, « PlantRT : Multi-Site Diversified Search and Recommendation for Citizen Sciences », dans *Bases de Données Avancées*, 2014.

[153] U. Shardanand et P. Maes, « Social Information Filtering : Algorithms for Automating "Word of Mouth" », dans *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1995, p. 210–217.

[154] A. Shepitsen, J. Gemmell, B. Mobasher et R. Burke, « Personalized Recommendation in Social Tagging Systems Using Hierarchical Clustering », dans *Proceedings of the 2008 ACM Conference on Recommender Systems*, 2008, p. 259–266.

[155] K. Sripanidkulchai, B. Maggs et H. Zhang, « Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems », dans *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, t. 3, 2003, p. 2166–2176.

[156] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek et H. Balakrishnan, « Chord : a Scalable Peer-to-Peer Lookup Protocol for Internet Applications », *IEEE/ACM Transactions on Networking*, t. 11, n° 1, p. 17–32, 2003.

[157] Strabon, *Geôgraphiká. Géographie de Strabon*, grec, trad. par A. Letrone. Imprimerie Impériale, 1819.

[158] D. Stutzbach et R. Rejaie, « Understanding Churn in Peer-to-Peer Networks », dans *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, 2006, p. 189–202.

[159] C. Tang, Z. Xu et S. Dwarkadas, « Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks », dans *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003, p. 175–186.

[160] C. Tang, Z. Xu et M. Mahalingam, « PeerSearch : Efficient Information Retrieval in Peer-to-Peer Networks », dans *Proceedings of the 1st ACM SIGCOMM Workshop on Hot Topics in Networks*, 2002.

[161] C. Tang, Z. Xu et M. Mahalingam, « pSearch : Information Retrieval in Structured Overlays », *ACM SIGCOMM Computer Communication Review*, t. 33, n° 1, p. 89–94, 2003.

[162] *Tela Botanica*, `http://www.tela-botanica.org`, 2014.

[163] R. Thiagarajan, G. Manjunath et M. Stumptner, « Computing Semantic Similarity Using Ontologies », dans *Proceeding of the 2008 International Semantic Web Conference*, 2008.

[164] K. H. L. Tso-Sutter, L. B. Marinho et L. Schmidt-Thieme, « Tag-Aware Recommender Systems by Fusion of Collaborative Filtering Algorithms », dans *Proceedings of the 2008 ACM Symposium on Applied Computing*, 2008, p. 1995–1999.

[165] D. Tsoumakos et N. Roussopoulos, « Adaptive Probabilistic Search for Peer-to-Peer Networks », dans *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, 2003, p. 102–109.

[166] A. Tveit, « Peer-to-Peer Based Recommendations for Mobile Commerce », dans *Proceedings of the 1st International Workshop on Mobile Commerce*, 2001, p. 26–29.

[167] B. Upadhyaya et E. Choi, « Social Overlay : P2P Infrastructure for Social Networks », dans *Proceedings of the 5th International Conference on Networked Computing, Advanced Information Management and Service, Digital Content, Multimedia Technology and its Applications*, 2009, p. 970–976.

[168] Y. Upadrashta, J. Vassileva et W. Grassmann, « Social Networks in Peer-to-Peer Systems », dans *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005, p. 200c–200c.

[169] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat et S. Amer-Yahia, « Efficient Computation of Diverse Query Results », dans *Proceedings of the 24th International IEEE Conference on Data Engineering*, 2008, p. 228–236.

[170] J Verhoeff, W. Goffman et J. Belzer, « Inefficiency of the Use of Boolean Functions for Information Retrieval Systems », *Communications of the ACM*, t. 4, n° 12, p. 557–558, 1961.

[171] S. Voulgaris et M. Van Steen, « Epidemic-Style Management of Semantic Overlays for Content-Based Searching », dans *Proceeding of the 2005 Euro-Par Conference on Parallel Processing*, 2005, p. 1143–1152.

[172] J. Wang, A. P. De Vries et M. J. Reinders, « Unifying User-Based and Item-Based Collaborative Filtering Approaches by Similarity Fusion », dans *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006, p. 501–508.

[173] R. Wilkinson et P. Hingston, « Using the Cosine Measure in a Neural Network for Document Retrieval », dans *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1991, p. 202–210.

[174] B. Yang et H. Garcia-Molina, « Efficient Search in Peer-to-Peer Networks », 2002.

[175] B. Yang et H. Garcia-Molina, « Improving Search in Peer-to-Peer Networks », dans *Proceeding of the 22nd International Conference on Distributed Computing Systems*, 2002, p. 5–14.

[176] C. Yu, L. Lakshmanan et S. Amer-Yahia, « It Takes Variety to Make a World : Diversification in Recommender Systems », dans *Proceedings of the 12th International Conference on Extending Database Technology : Advances in Database Technology*, 2009, p. 368–378.

[177] X. Zhu, A. B. Goldberg, J. Van Gael et D. Andrzejewski, « Improving Diversity in Ranking using Absorbing Random Walks. », dans *Proceeding of the 2007 Conference of the North American Chapter of the Association for Computational Linguistics*, 2007, p. 97–104.

[178] C.-N. Ziegler, G. Lausen et L. Schmidt-Thieme, « Taxonomy-Driven Computation of Product Recommendations », dans *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*, 2004, p. 406–415.

[179] C.-N. Ziegler, S. M. McNee, J. A. Konstan et G. Lausen, « Improving Recommendation Lists through Topic Diversification », dans *Proceedings of the 14th International Conference on World Wide Web*, 2005, p. 22–32.

[180] *eBird*, `http://ebird.org`, 2014.

[181] *xeno-canto : : Sharing bird sounds from around the world*, `http://www.xeno-canto.org`, 2014.

Annexes

ANNEXE A

Notations générales utilisées dans ce mémoire de thèse

Table A.1: Notations générales.

Notation	Description
k_i	Un mot clé.
K	L'ensemble des mots clés, de taille $\|K\| = z$.
q	Une requête, généralement à mots clés (*e.g.* $q = k_1, ..., k_t$) et de taille $\|q\| = t$.
it	Un objet du système.
I	L'ensemble des objets du système, au nombre de $\|I\| = m$.
I_u	Les objets partagés par un utilisateur u.
$profile(u)$	Le profil de l'utilisateur u, calculé en fonction des objets partagés par u, notés I_u.
R^q	Une liste de résultats, étant donné, une requête q, de taille k.
U	L'ensemble des utilisateurs du système, au nombre de $\|U\| = n$.
C	Liste des candidats, utilisée dans le calcul d'un ensemble divers.
L_k	Une liste inversée associant les objets pertinents étant donné le mot clé k.
L_q	Les listes inversées correspondant à chacun de mots $L_q = \{L_k \| k \in q\}$.

Table A.2: Notations générales utilisées dans les chapitres traitant de la distribution des données.

Notation	Description
$U\text{-}Net_u$	Ensemble d'utilisateurs, ou pairs, que u a choisi pour répondre à ses requêtes. Sa taille est fixée à $\|U\text{-}Net\| \leq N$.
$RepC_u$	Cache de réplication d'un utilisateur d'une taille de $\|RepC_u\| = r_u$.
VN_i	Un nœud virtuel. Il s'agit d'un regroupement d'utilisateurs similaires appartenant à un même site.
$VN\text{-}Network_i$	Le pendant du *U-Net* pour le multisite : il s'agit d'un groupe de nœuds virtuels distants choisis par le VN_i pour répondre aux requêtes futures.

ANNEXE B

Méthode adaptative de diversification des index

B.1 Adaptive Diversity Loss Analysis

In this section, we analyze the quality and performance of the adaptive approach. Section B.1.1 proves Theorem 1 and Section B.1.2 proves Theorem 2 (cf. Chapitre 4).

In the two following sections, we use the set of notations presented in Table B.1.

B.1.1 Quality Loss Analysis

In this subsection, we show that given a keywords query q and L_q the set of inverted lists used to compute q's results, if q represents at least $(1 - f_{max})\%$ of the queries that access each one of the inverted lists $L_k \in L_q$, then the adaptive approach returns the same results as the basic algorithm ($R_a^q = R_b^q$).

To simplify the demonstration, we first consider *h-frequent* queries. Then, we show that if a query has a frequency of h on the inverted list it accesses, it is necessarily *h-frequent* if $h \geq (1 - f_{max})$.

Définition 8 (*h-frequent* queries)**.**

Given a query q and L_q the set of inverted lists used to compute q's results R^q, the query q is said to be *h-frequent* if for each inverted list $L_k \in L_q$, the results of q that are in L_k are at least returned by $h\%$ of the queries that access L_k.

Remark 1.

If an item is accessed by $h\%$ of the queries, then its redundancy frequency cannot be higher than $1 - h\%$.

Notation	Description
f_{max}	A system defined value representing a redundancy frequency. All items whose redundancy frequency is inferior to this value have minimum quality and performance guarantees.
L_k / L_q	The inverted list corresponding to the keyword k or the set of inverted lists corresponding used to compute the query's q results list. If $q = \{k\}$, then $L_q = L_k$. $L_k[i]$ is the i^{th} item, while $L_k[it]$ is the position of item it.
$R^q_{a/b}$	The results list of query q computed using the adaptive (a) or in the basic (b) algorithm.
Δ	Given an inverted list, the variation between the indexation score of an item in the basic algorithm and its indexation score in the adaptive algorithm.
$rfreq(L_k, i)$	The redundancy frequency of the i^{th} item of the inverted list L_k.
$W(x)$	The weighed function that permits to decrease the indexation score of some items. It is continuous and monotonously decreasing.
$1 - l_{max}$	The worst value which $W(x)$ can have for items whose redundancy frequency is inferior to f_{max}. $W(f_{max}) = l_{max}$.
$th_{a/b}(s_1, ..., s_n)$	The threshold function used in the adaptive (a) or in the basic (b) algorithm.
$sa_{a/b}(L_k)[i]$	The value of the sorted access performed on the inverted list corresponding to the keyword k at position i (using the adaptive (a) or the basic (b) algorithm's index).
it	An item.
$rel(L_k, i)$	The relevance of the i^{th} item in L_k with respect to keyword k.
$nbSA^q_{a/b}$	The number of sorted accesses needed to compute R^q using respectively an adaptive index or a static index.

Table B.1: Notations used in the proofs.

First, we propose the following theorem:

Theorem 3.

An *h-frequent* single keyword query processed using the adaptive algorithm has the guarantee to have the same results as it would if it had been processed using the basic algorithm ($R_a^q = R_b^q$) if $h \geq 1 - f_{max}$.

Proof. Suppose a single keyword *h-frequent* query q where $h \geq 1 - f_{max}$. Suppose that q's results list R_a^q already contains $p-1$ items, so that $R_a^q[1..p-1] = R_b^q[1..p-1]$. The p^{th} item of R_b^q is noted it. If $R_a^q = R_b^q$, then the p^{th} item in R_a^q is it.

Recall that, in our approach, the items index score takes into account a weight that depends on the items redundancy frequency (cf. Algorithm 4):

$$sa_a(L_k)[i] = rel(L_k[i], L_k) \times W(rfreq(L_k[i], L_k)) \tag{B.1}$$

The threshold is evaluated based on the item's index score (cf. Algorithm 4):

$$th_a(sa_a(L_k)[i]) = \frac{rel(L_k[i], L_k) \times W(rfreq(L_k[i], L_k))}{(1 - l_{max})} \tag{B.2}$$

Finally, the stop condition of our approach is the following (cf. Algorithm 4):

$$\textbf{if } th_a(sa_a(L_k)[i]) > rel(L_k[i], L_k) \times W(rfreq(L_k[i], L_k)) \tag{B.3}$$

It means that when this condition is fulfilled, the algorithm stops and adds the best items accessed until now in the results list. Therefore, if the condition is fulfilled before accessing it, the results list R_a^q would not be identical to R_b^q.

At some point, if it has been accessed, we know that it will be added to R_a^q because it has the best score and R_a^q will remain identical to R_b^q.

However, if it has not been accessed yet, its position is necessarily after the last accessed item. Let us say that the last accessed item is the i^{th} item in L_k. Because scoring function is monotonous, we can define the next equation:

$$rel(L_k[i], L_k) \times W(rfreq(L_k[i], L_k)) \geq rel(it, L_k) \times W(rfreq(it, L_k)) \tag{B.4}$$

where $L_k[R_a^q[p-1]] < i < L_k[it]$. Since q is *h-frequent*, the next item it is also *h-frequent* and since $W(x)$ is a continuous monotonous decreasing function, if $h \geq 1 - f_{max}$ then $f_{max} \geq 1 - h$ and we can define the following equation:

$$rel(it, L_k) \times W(it, L_k) \geq rel(it, L_k) \times W(f_{max}) \tag{B.5}$$

This leads us to Equation B.6.

$$rel(it, L_k) \times W(it, L_k) \geq rel(it, L_k) \times (1 - l_{max}) \tag{B.6}$$

Finally, using Equations B.2, B.4 and B.6, we can define the following inequality:

$$th_a(sa_a(L_k)[i]) \geq rel(it, L_k) \geq rel(it, L_k) \times div(it|R_a^q) \tag{B.7}$$

Because $R_a^q \leq 1$ and since the item it is the next element that should be added to R_a^q, we know that it is not possible to find an item at position i where $L_k[R_a^q[p-1]] < i < L_k[it]$ so that

$$rel(L_k[i], L_k) \times div(L_k[i]|R_a^q) \geq rel(it, L_k) \times div(it|R_a^q)$$

and therefore its score is higher than the threshold until it is accessed. Therefore, it will necessarily be accessed and added to R_a^q which will remain identical to R_b^q. □

Theorem 4.

An *h-frequent* n keywords query is guaranteed to have 0 loss in terms of quality if $h \geq 1 - f_{max}$.

Proof. Let the aggregation operator for multiple keywords queries be the sum function. Then the relevance of an item is the sum of its relevance in each one of the inverted lists and the threshold is the sum of each threshold in each inverted list. We are given a n keywords query q and its results list R_a^q knowing that there are already $1-p$ items in R_a^q. The next item to add in the results list is noted it. If Theorem 3 cannot be extended for n keywords queries, it means that it is possible to find in one of the inverted lists an item at position i where $L_k[R_a^q[p-1]] < i < L_k[it]$ whose score is higher than the threshold. Since the threshold in each inverted list is higher than or equal to the next item it, the aggregation of the threshold is higher than the score of the next item it. Because it is the next item to add, it is also the item with the best score. Therefore, no item can have a score higher than the threshold computed until it is accessed. □

Theorem 5.

An *h-frequent* query q that has always been processed at a frequency of h where $h \geq 1 - f_{max}$ will be at least $(1 - f_{max})$-*frequent*.

Proof. Suppose that the query q is submitted and is at least $(1 - f_{max})$-*frequent*. Then, its results will be optimal (cf. Theorem 4) and will at least reflect the frequency of the query. Since all queries are at least $(1 - f_{max})$-*frequent* at first, then q will remain at least $(1 - f_{max})$-*frequent*. □

Remark 2.

The frequency of a query is very sensible to noise when the system has just started because the frequency is evaluated with respect to a very small number of queries. However, after a certain time, the *h-frequency* of the queries will become stable.

Remark 3.

It is possible to define values of f'_{max} minimizing the probability that an item of frequency $h > 1 - f_{max}$ at a given time reaches a frequency less than $1 - f'_{max}$.

As a conclusion, we guarantee that if the query's worst frequency is at least f_{max}, its results quality will remain optimal. The experiments show that, with our optimization techniques, even the queries that do not respect this property remain of high quality (*i.e.* their quality loss is very small).

B.1.2 Performance Analysis

In this subsection, we show that given a keywords query q and *indices* on the inverted lists used to compute q's results, if q represents at least $(1 - f_{max})\%$ of the queries (and therefore is at least $(1 - f_{max})$-*frequent*) that access *indices*, it is guaranteed to perform at maximum the same number of index accesses as the basic algorithm.

In the following, we show that the only cases that could make our approach to perform more accesses on the indices than the basic algorithm are impossible. More precisely, only two cases could incur such behavior. First, since we know that queries that are at least $(1 - f_{max})$-*frequent* are optimal compared to the basic algorithm, if the position in the index of the last result is increased, then, the number of sorted accesses needed to build the results list would be increased. Second, the threshold can no longer stop the algorithm at a specific rank in the index while it could stop it with the basic algorithm.

About the first case, generally, items in an inverted list follow a *Zipfian* distribution. In other words, there are a few items with very high scores and a lot of items with very small scores. Based on this idea, we make the following assumption:

Assumption 1.

An inverted list of items follows exactly a *Zipfian* distribution:

$$f(k; S, N) = \frac{1/k^s}{\sum_{n=1}^{N}(1/n^s)} \tag{B.8}$$

Where N is the number of items in the inverted list and k the rank of the item.

Based on this assumption, we propose the following theorem:

Theorem 6.

For all values of $f_{max} < 1$, it is possible to find a value of $l_{max} > 0$ such that it is not possible to find an item at least $(1 - f_{max})$-*frequent* whose position increases.

Proof. The position of an item increase if the score of a lower scored item increases, and therefore whose position is decreased. This is impossible because, at the beginning, $W(x)$ has its highest possible value and $W(x)$ is a monotonous continuous decreasing function. Thus, the score of an item can only decrease compared to their score at the beginning.

Therefore, the position of an item can only increase because its weight $W(x)$ decreases. In other words, given the set of items IT which are at least $(1-f_{max})$-*frequent*, then it is possible to find an item $it \in IT$ at rank r in the inverted list, whose position is increased if it is possible to find a solution to the following system:

$$\left.\begin{aligned} f(k; S, N) \times (1 - l_{max}) &= f(k'; S, N) \\ k' &\geq k + 1 \end{aligned}\right\} \tag{B.9}$$

In other words, the solution is the position of an item whose position could be increased. If this position is higher than the number of items in the inverted list, then, it would be impossible for our approach to perform worse than the basic algorithm. By solving the system, we can obtain the following inequality:

$$k' \geq \frac{1}{1 - (1 - l_{max})^{1/s}} \tag{B.10}$$

Therefore, it is possible to choose specific values of l_{max} that are small enough so that $k' > N$, which would imply that it is impossible to find an item at least $(1 - f_{max})$-*frequent* whose position k increases.

Moreover, if $l_{max} \to 0$, then $k' \to \infty$, which implies that the probability that the position of an item which is at least $(1 - f_{max})$-*frequent* increases, tends to 0. □

Assumption 2.

In the following, we presume the worst case, which means that all items which are at least $(1 - f_{max})$-*frequent* have not decreased in their position compared to the basic algorithm. This implies that the number of sorted accesses to compute the results list using the adaptive algorithm is at least equal to the basic algorithm.

Then we propose the following theorem:

Theorem 7.

If a query is h-*frequent* and $h \geq 1 - f_{max}$, then the threshold condition enables the algorithm to stop, in the worst case, on the same item as the basic algorithm.

Proof. Suppose a keywords query q at least $(1 - f_{max})$-*frequent* and its results list R_a^q. Suppose that R_a^q already contains $p - 1$ items. The next item to add is it and has already been accessed by our algorithm.

Recall that the smaller a threshold is, the faster the algorithm stops (cf. Equation B.3). In the following, we show that the threshold's value used in Equation 4.17 (cf. Chapitre 4) permits with a very high probability to stop on the same item as the basic algorithm.

Since $rel(L_k[i], L_k)$ is static, the highest value of the threshold is computed when $W(l'[i]) \simeq 1$ (cf. Equation B.2). For simplification, in the following, we suppose that $W(l'[i]) = 1$, which is the worst case for our algorithm.

Therefore, in the worst case, the difference between the basic algorithm threshold and our threshold follows this equation (according to Assumption 2 and Equation B.1):

$$th_a(sa_a(L_k)[i]) = \frac{th_b(sa_b(L_k)[i])}{1 - l_{max}} \tag{B.11}$$

Since we are in the worst case (cf. Assumption 2), the number of sorted accesses of our solution is either equal or higher than the basic algorithm. The only case where it would be higher is when the basic algorithm's threshold satisfies the stop condition while our solution's threshold does not. This leads to the following equation:

$$\frac{th_b(sa_b(L_k)[i])}{1 - l_{max}} > rel(it) \times div(it) \geq th_b(sa_b(L_k)[i]) \tag{B.12}$$

In the following, we compute the probability that an item satisfies such an equation. This probability is noted $\mathcal{P}(nbSA_a < nbSA_b)$. Given T as all the possible values for the threshold, $\mathcal{P}(nbSA_a < nbSA_b)$ is computed as follows for a one keyword query:

$$\mathcal{P}(nbSA_a < nbSA_b) = \sum_{t \in T} \int_{t \times (1 - l_{max})}^{t} f(x)dx \tag{B.13}$$

where $f(x)$ is the distribution of the final scores of the items whose range is in: $[0, 1]$. $f(x)$ is a continuous function. Equation B.13 calculates the probability to find an item *it* that satisfies Equation B.12, taking into account all possible threshold values. For a n keywords query, it is the intersection of this probability for each one of the keywords. Developing the integral of Equation B.13 leads to the following equation:

$$\int_{t \times (1 - l_{max})}^{t} f(x)dx = F(t) - F(t \times (1 - l_{max})) \tag{B.14}$$

where $F(x)$ is the primitive of $f(x)$. Since $f(x)$ is continuous between 0 and 1, $F(x)$ remains continuous between 0 and 1, and since $l_{max} \to 0$, then $t \times (1 - l_{max}) \to t$ and $\int_{t \times (1 - l_{max})}^{t} f(x)dx \to 0$. The finite sum of a set of probabilities which are equal to 0 is 0. This probability is for one keyword queries. For several keywords, the probability is the intersection of the probabilities of each keyword which is even smaller.

In other words, the probability follows Equation B.15.

$$\lim_{l_{max} \to 0} \mathcal{P}(nbSA_a < nbSA_b) = 1 \tag{B.15}$$

□

Remark 4.

Suppose that the previous probability's event happens, it would imply a unique additional sorted accesses. However, if the event happens several times, it will provoke as many additional sorted accesses, but the probability that this event happens is even less likely.

As a conclusion we have shown that in the worst case, the probability that our algorithm performs a few more index accesses tends toward 0.

ANNEXE C

Impressions écran de PlantRT et de DocRT

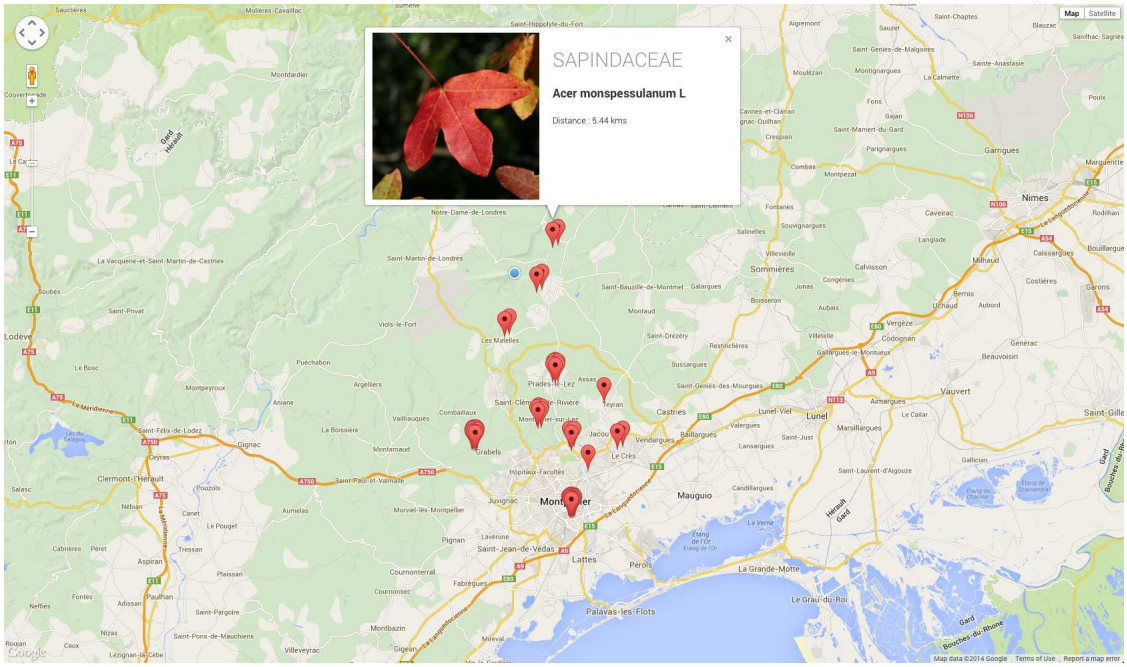


Figure C.1: Géo-recommandation avec PlantRT autour de Montpellier.

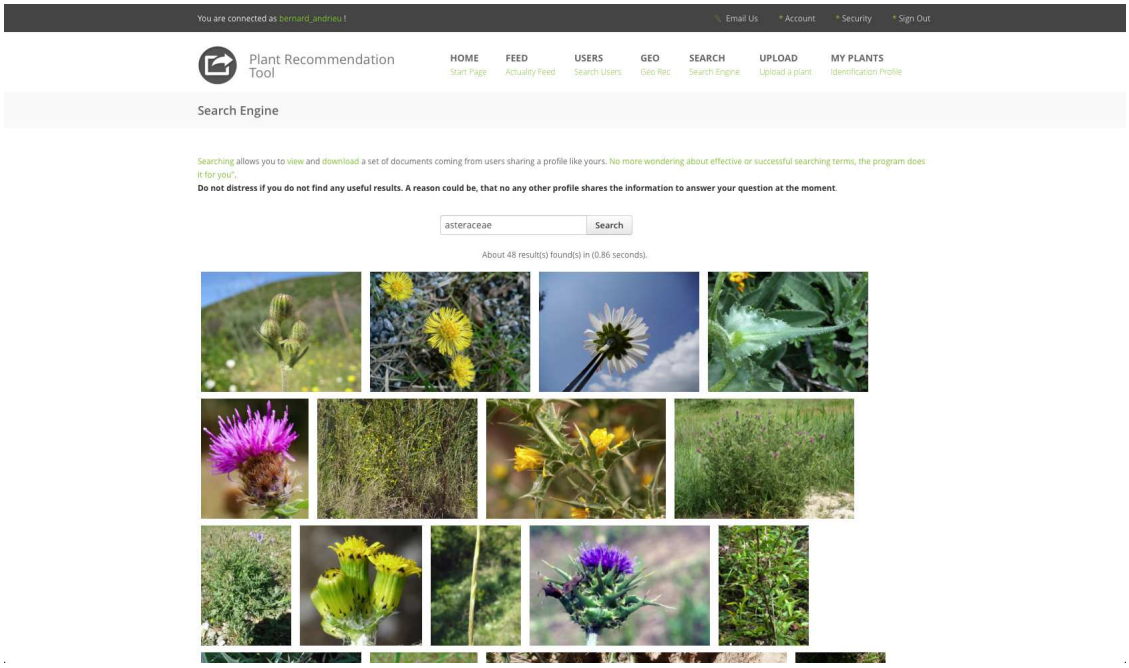


Figure C.2: Recherche de plantes de la famille « Asteraceae » avec PlantRT.

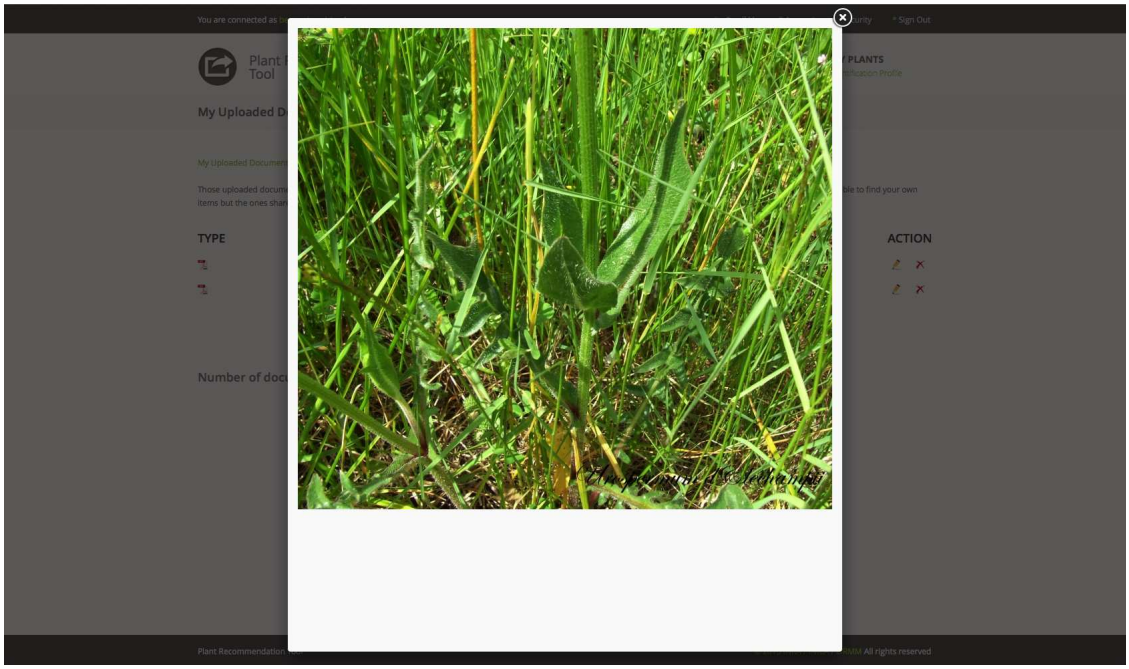


Figure C.3: Affichage d'une observation de plante partagée par l'utilisateur courant avec PlantRT.

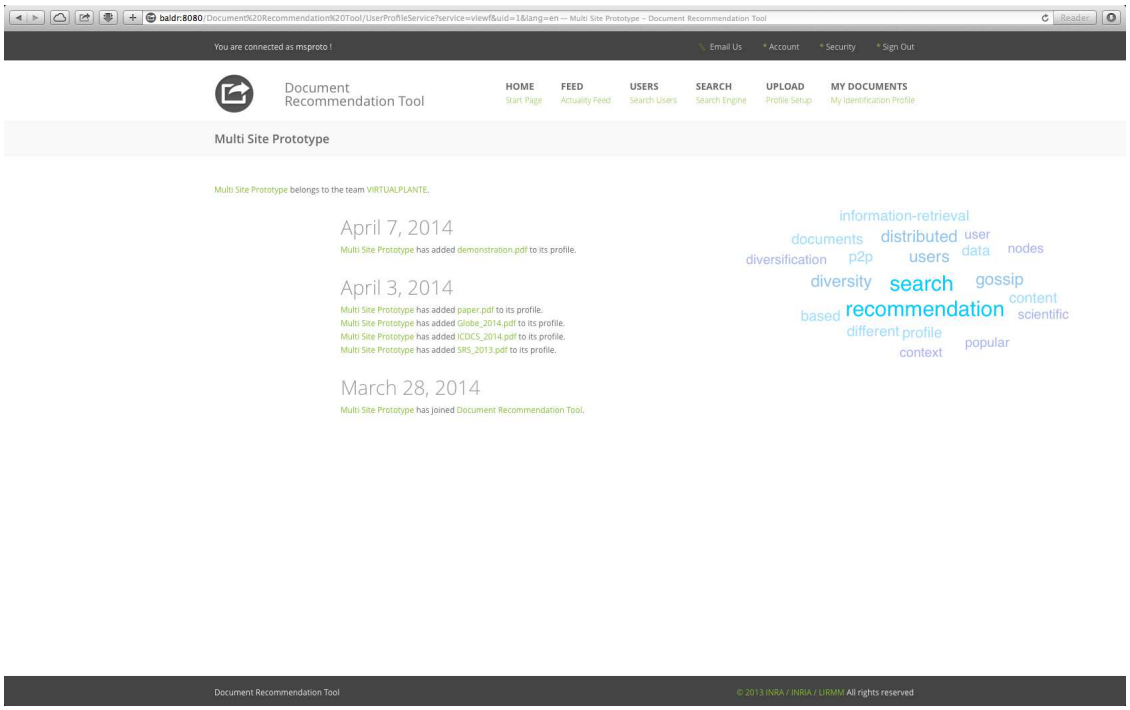


Figure C.4: Affichage des documents partagés et du profil de l'utilisateur courant sous la forme d'un nuage de mots avec DocRT.

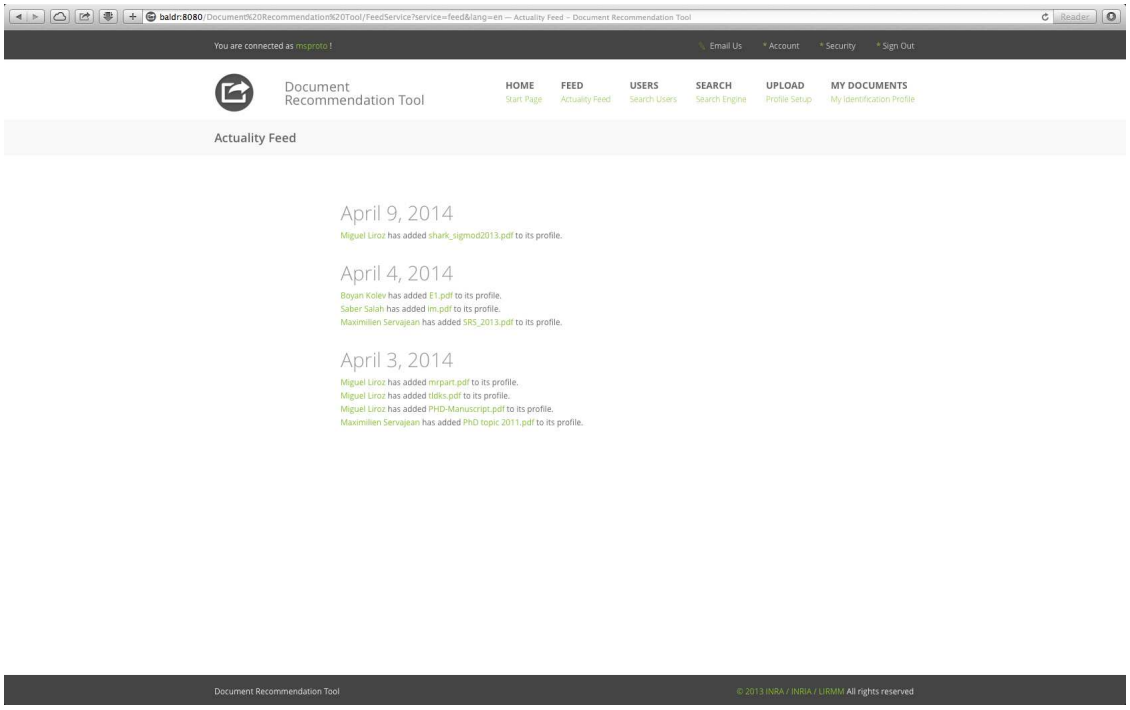


Figure C.5: Affichage des documents partagés par d'autres utilisateurs « suivis » avec DocRT.

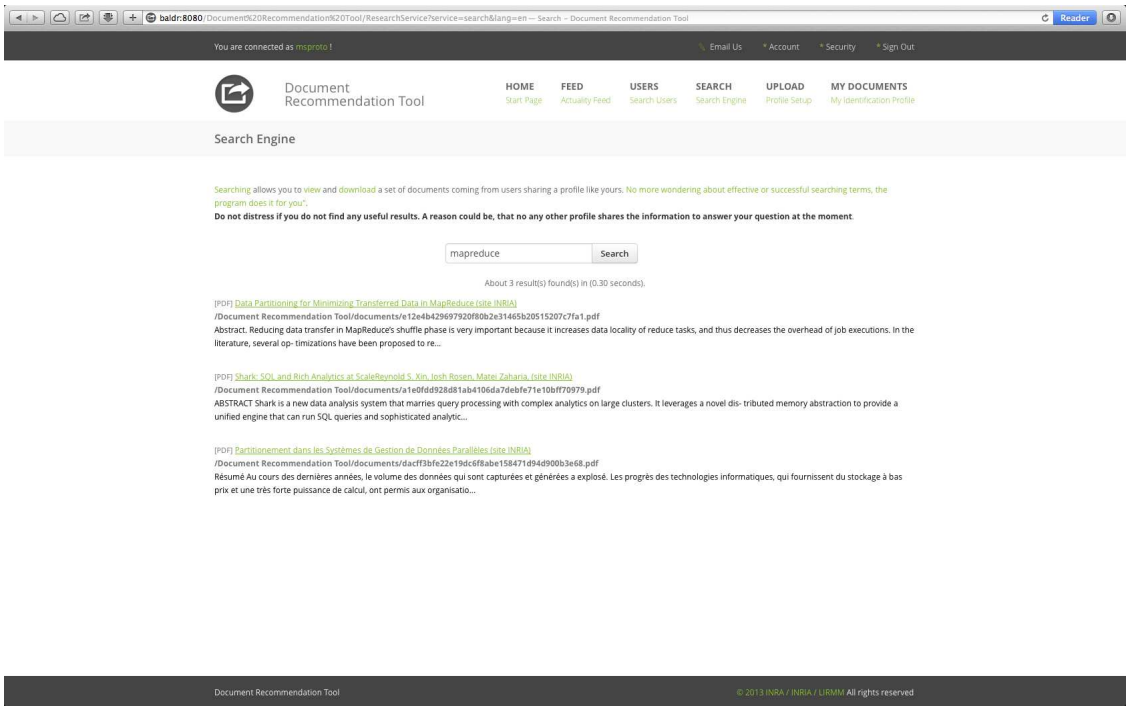


Figure C.6: Affichage des résultats d'une recherche avec la requête « mapreduce » avec DocRT.

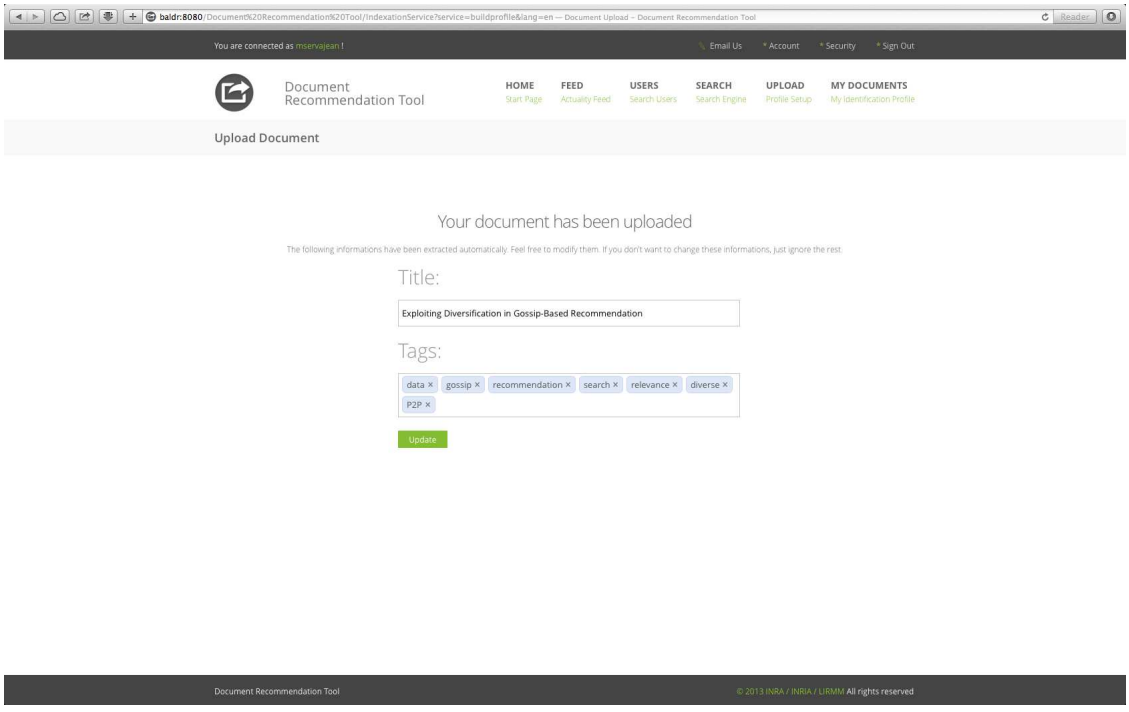


Figure C.7: Ajout d'un nouveau document, extraction du titre et d'étiquettes avec DocRT.

Index

B

D

R

S