Une approche coopérative d'aide à la réparation de requêtes floues

Grégory Smits, Olivier Pivert

Université de Rennes 1 / IRISA
Lannion - France
gregory.smits@irisa.fr,olivier.pivert@irisa.fr

RÉSUMÉ. *Les systèmes coopératifs visent notamment à éviter de retourner à l'utilisateur « il n'y a pas de réponse » lorsqu'une requête échoue. De tels systèmes cherchent plutôt à fournir les raisons de l'échec, sous la forme par exemple de sous-requêtes minimales à résultat vide (Minimal Failing Subqueries, MFS), et à construire des requêtes alternatives à résultat non vide (maXimal Succeeding Subqueries, XSS), aussi proches que possible de la requête initiale. Dans le contexte spécifique de l'interrogation floue de bases de données, nous proposons une méthode efficace pour déterminer les MFS et les XSS graduelles d'une requête, méthode qui s'appuie sur un résumé, calculé dynamiquement, d'une partie de la base.*

ABSTRACT. *Telling the user that there is no result for his/her query is very poorly informative and corresponds to the kind of situation cooperative systems try to avoid. Cooperative systems should rather explain the reason(s) of the failure, materialized by Minimal Failing Subqueries (MFS), and build alternative succeeding queries, called maXimal Succeeding Subqueries (XSS), that are as close as possible to the original query. In the particular context of fuzzy querying, we propose an efficient unified approach to the computation of gradual MFSs and XSSs that relies on a fuzzy-cardinality-based summary of the relevant part of the database.*

MOTS-CLÉS : *requêtes floues, réponses coopératives, cardinalités floues, résumés de base de données, réparation de requête.*

KEYWORDS: *fuzzy querying, cooperative answering, fuzzy cardinalities, database summarization, query repair.*

DOI:10.3166/ISI.21.3.11-30

1. Introduction

L'un des objectifs majeurs des systèmes coopératifs est d'éviter de retourner « il n'y a pas de réponse » quand une requête échoue. Un tel système cherchera plutôt, en une telle occurrence, à expliquer pourquoi l'échec est survenu et à suggérer des requêtes alternatives, au résultat non vide, aussi proches que possible, sémantiquement parlant, de la requête originale. Alors que la plupart des travaux de la littérature s'intéressent au problème des réponses vides dans un cadre de requêtes booléennes, nous nous penchons ici sur le cas de requêtes floues, exprimant des préférences utilisateur. Nous considérons la situation où une requête floue « échoue » — nous verrons plus loin que ce terme peut prendre différents sens — et proposons une approche de type « expliquer et réparer » reposant sur un résumé d'une partie de la base de données. Le résumé en question fournit des informations relatives à la distribution des données sur les divers prédicats flous impliqués dans la requête initiale.

Par rapport aux requêtes booléennes, les requêtes floues réduisent le risque d'obtenir des ensembles vides de réponses dans la mesure où l'usage d'une échelle de discrimination plus fine — [0, 1] au lieu de {0, 1} — augmente les chances d'un élément d'être considéré un tant soit peu satisfaisant. Néanmoins, il peut se produire qu'aucun élément de la base de données considérée ne satisfasse la requête même faiblement. Dans un contexte d'interrogation floue, en plus du problème des réponses vides au sens strict, une autre situation problématique est celle où, bien que le résultat ne soit pas vide, il ne contient que des éléments dont le degré de satisfaction relatif aux préférences exprimées par l'utilisateur est faible. Nous montrons dans cet article qu'une approche générique exploitant la notion de cardinalité floue peut être employée pour i) fournir des explications dans les deux types de situation (résultat vide ou insuffisamment satisfaisant), et ii) suggérer des requêtes alternatives pertinentes dont on sait que le résultat sera non vide. Comme nous le verrons, les sous-requêtes minimales à résultat vide (Minimal Failing Subqueries, MFS (McSherry, 2005)) constituent des explications utiles relativement aux conflits dans une requête, qui peuvent être utilisées pour fonder une stratégie de relaxation semi-automatique ciblée. Cette stratégie consiste à identifier les sous-requêtes maximales à résultat non vide (maXimal Succeeding Subqueries, XSS en abrégé) de la requête initiale et à laisser l'utilisateur choisir quelle relaxation il est prêt à accepter.

Le reste de l'article est structuré de la façon suivante. La section 2 fournit quelques rappels utiles sur les requêtes floues et les résumés de base de données fondés sur des cardinalités floues. La section 3 montre comment de tels résumés peuvent être utilisés pour calculer efficacement les MFS et les XSS d'une requête à résultat vide, afin d'expliquer et de réparer cette dernière. Les résultats d'une première expérimentation sont présentés et analysés en section 4. La section 5 est consacrée au positionnement de l'approche par rapport à des travaux connexes. Enfin, la section 6 rappelle les contributions principales de l'article et esquisse quelques perspectives.

Principe de l'approche

La figure 1 illustre le processus coopératif mis en place pour aider un utilisateur à corriger une requête retournant un résultat vide ou faiblement satisfaisant. La première étape consiste à résumer la partie utile de la BD et c'est à partir de ce résumé que les raisons minimales de l'échec sont identifiées. Le système suggère ensuite à l'utilisateur des corrections de sa requête initiale avec la garantie d'obtenir un résultat non vide et/ou de meilleure qualité (en termes de satisfaction).

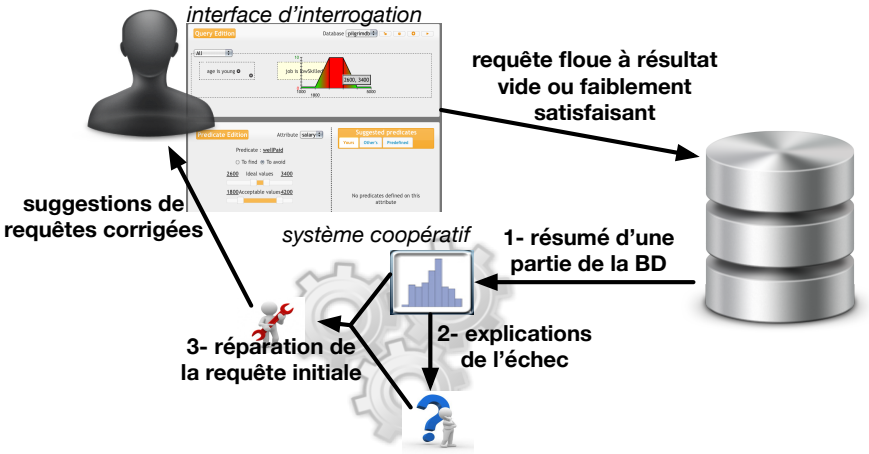

Figure 1. Aperçu de l'approche coopérative de réparation de requête

2. Préliminaires

2.1. Ensembles flous et requêtes floues

La théorie des ensembles flous a été définie par L.A. Zadeh (Zadeh, 1965) pour modéliser des classes d'objets aux frontières vagues. Pour de tels ensembles, la transition entre l'appartenance complète et la non-appartenance est graduelle plutôt que tranchée. Des exemples typiques de classes floues sont celles associées à des adjectifs du langage naturel tels que *jeune, bon marché, rapide*, etc. Formellement, un ensemble flou F défini sur un référentiel U est caractérisé par une fonction d'appartenance $\mu_F : U \rightarrow [0,1]$ où $\mu_F(u)$ désigne le degré d'appartenance de u à F. En particulier, $\mu_F(u) = 1$ reflète l'appartenance totale de u à F, tandis que $\mu_F(u) = 0$ exprime la non-appartenance absolue. Lorsque $0 < \mu_F(u) < 1$, on parle d'appartenance partielle.

Deux ensembles classiques présentent un intérêt particulier lorsque l'on définit un ensemble flou F:

– le noyau $C(F) = \{u \in U \mid \mu_F(u) = 1\}$, constitué des *prototypes* de F,
– le support $S(F) = \{u \in U \mid \mu_F(u) > 0\}$.

La notion de coupe de niveau, ou α-coupe englobe ces deux concepts. L'α-coupe (resp. α-coupe stricte) F_α (resp. $F_{\overline{\alpha}}$) d'un ensemble flou F correspond à l'ensemble des éléments du référentiel qui ont un degré d'appartenance à F au moins égal à (resp. strictement plus grand que) α :

$$F_\alpha = \{u \in U \mid \mu_F(u) \geq \alpha\} \text{ et } F_{\overline{\alpha}} = \{u \in U \mid \mu_F(u) > \alpha\}.$$

De façon directe, on a : $C(F) = F_1$ et $S(F) = F_{\overline{0}}$.

Soit F et G deux ensembles flous sur l'univers U. On dit que $F \subseteq G$ ssi $\mu_F(u) \leq \mu_G(u), \forall u \in U$. Le complément de F, noté F^c, est défini par $\mu_{F^c}(u) = 1 - \mu_F(u)$. De plus, $F \cap G$ (resp. $F \cup G$) est défini de la façon suivante : $\mu_{F \cap G} = min(\mu_F(u), \mu_G(u))$ (resp. $\mu_{F \cup G} = max(\mu_F(u), \mu_G(u))$).

Comme dans le cas booléen, les contreparties logiques des opérateurs ensemblistes $\cap$, $\cup$ et la complémentation sont respectivement la conjonction $\wedge$, la disjonction $\vee$ et la négation $\neg$. Voir (Dubois, Prade, 2000) pour de plus amples détails.

Dans un cadre de bases de données, les ensembles flous sont un outil commode et très expressif pour modéliser des critères vagues et des préférences utilisateur. La théorie des ensembles flous offre une large panoplie de connecteurs permettant d'agréger des préférences suivant différentes sémantiques, avec effet compensatoire entre les critères ou non. Les prédicats flous sont utilisés pour représenter des propriétés telles que *récent*, *faible*, *peu cher*, *élevé*, ..., qui correspondent à des notions familières et aisément compréhensibles par un utilisateur final. De plus, conformément à la nature imprécise des concepts qu'ils modélisent, les ensembles flous sous-jacents à ces propriétés introduisent de la gradualité dans l'évaluation de la satisfaction des préférences par les objets de la base. Cette notion de satisfaction graduelle permet d'ordonner les réponses qui satisfont un tant soit peu les critères spécifiés par l'utilisateur.

Les opérateurs de l'algèbre relationnelle peuvent être étendus de façon assez directe aux relations floues en considérant d'une part ces dernières comme des ensembles flous, et en introduisant des prédicats graduels dans les opérations appropriées d'autre part. Les définitions de ces opérateurs algébriques étendus sont données dans (Bosc *et al.*, 1999 ; Pivert, Bosc, 2012). En guise d'illustration, nous donnons ci-après la définition de la sélection floue, où r désigne une relation (floue ou classique) et φ une condition floue.

$$\mu_{\sigma_\varphi(r)}(t) = \min(\mu_r(t),\ \mu_\varphi(t)).$$

Une extension de SQL s'appuyant sur cette algèbre relationnelle étendue est présentée dans (Bosc, Pivert, 1995 ; Pivert, Bosc, 2012). Nous nous limitons ici à la description du bloc de base, car c'est tout ce dont nous avons besoin pour la suite. Par rapport à SQL, les différences concernent principalement deux aspects :

– le calibrage du résultat puisque ce dernier est composé d'éléments discriminés, ce qui peut être réalisé à l'aide d'un seuil quantitatif (nombre souhaité de réponses, noté k), d'un seuil qualitatif (degré minimal de satisfaction exigé, noté α), ou les deux,

– la nature des conditions autorisées, comme mentionné plus haut.

En conséquence, le bloc de base prend la forme suivante :

select [**distinct**] [$k \mid \alpha \mid k, \alpha$] *attributs* **from** *relations*
where *cond-floue*

où *cond-floue* peut faire intervenir à la fois des prédicats flous et des critères booléens.

Bien que ce ne soit pas le sujet principal de l'article, il est important de mentionner que des techniques d'évaluation appropriées ont été proposées pour les requêtes floues. La mise en œuvre d'un système d'interrogation floue peut être abordée selon trois types d'architecture (Urrutia *et al*., 2008) :

– *couplage faible* : les nouvelles fonctionnalités sont intégrées au travers d'une couche logicielle au-dessus du SGBD. L'avantage majeur de ce type d'architecture réside dans sa portabilité, puisque n'importe quel SGBD peut être utilisé comme moteur de requête sous-jacent.

– *couplage moyen* : les nouvelles fonctionnalités peuvent être intégrées par l'intermédiaire de procédures stockées, à l'aide d'un langage procédural approprié (tel que PL/SQL dans le cas d'Oracle). Une alternative consiste à recourir à des fonctions externes.

– *couplage fort* : les nouvelles fonctionnalités sont incorporées dans le noyau même du SGBD. Cette solution, qui est évidemment la plus efficace en termes d'évaluation de requête, implique de réécrire entièrement le moteur d'évaluation, y compris l'analyseur syntaxique et l'optimiseur du SGBD.

Le type de mise en œuvre présenté dans (Smits et al., 2013b) est à la jonction entre le couplage moyen et le couplage fort dans la mesure où i) les fonctions d'appartenance correspondant aux prédicats flous spécifiés par l'utilisateur sont définies comme des procédures stockées et ii) les extensions graduelles des opérateurs (norme triangulaire pour la conjonction, conorme triangulaire pour la disjonction, quantificateurs flous) sont implémentées en langage C et intégrées dans le moteur de requêtes du SGBD relationnel PostgreSQL.

Indépendamment du type de couplage choisi, certaines approches de la littérature proposent de passer par une étape dite de *dérivation* (Bosc, Pivert, 2000 ; Pivert, Bosc, 2012) pour générer une requête booléenne utilisée pour préfiltrer la relation (ou le produit cartésien des relations) concernée. L'idée est de restreindre le calcul des degrés de satisfaction aux seuls n-uplets qui satisfont à un niveau suffisant (α) la condition de sélection (ou de jointure). Dans ce type d'approche, l'évaluation de requêtes floues comporte trois étapes :

1. dérivation d'une requête booléenne à partir de la condition floue apparaissant dans la requête utilisateur,

2. évaluation de cette requête booléenne et production de la relation résultante,

3. calcul des degrés de satisfaction attachés aux n-uplets de cette relation (et élimination éventuelle des n-uplets non suffisamment satisfaisants), ce qui produit la

relation floue constituant le résultat final.

En termes de performances, l'intérêt d'utiliser une architecture de type « couplage moyen » (plutôt que faible) réside dans le fait que la relation floue résultante est calculée durant la phase de sélection des n-uplets (aucun programme externe n'a besoin d'être appelé pour effectuer l'étape 3).

2.2. Cardinalités floues

Nous introduisons maintenant une notion qui va jouer un rôle central dans l'approche coopérative décrite plus loin, celle de cardinalité floue. Dans le contexte de l'interrogation flexible de bases de données, les cardinalités floues (Dubois, Prade, 1985) constituent un formalisme commode pour représenter le nombre de n-uplets d'une relation qui satisfont un prédicat flou donné à des degrés divers. On suppose dans la suite que les degrés d'appartenance appartiennent à une échelle finie $\mathcal{S}$: $1 = \sigma_1 > \sigma_2 > ... > \sigma_f > 0$.

Une cardinalité floue (Dubois, Prade, 1985) peut être représentée par une distribution de possibilité ou, sans perte d'information, à l'aide d'une représentation plus compacte :

$$F_{P_1} = 1/c_1 + \sigma_2/c_2 + ... + \sigma_f/c_f,$$

où $c_i,\ i = 1..f$ est le nombre de n-uplets dans la relation concernée qui sont P_1 à un degré au moins égal à σ_i. En ce qui concerne le calcul de la cardinalité relative à une conjonction de q prédicats, notée $F_{P_1 \wedge P_2 \wedge ... \wedge P_q}$, le principe est de considérer le minimum des degrés de satisfaction obtenus par les n-uplets t pour chacun des prédicats concernés : $\min(\mu_{P_1}(t), \mu_{P_2}(t), ..., \mu_{P_q}(t))$.

2.3. MFS graduelles

Un ensemble vide de réponses associé à une requête floue $Q = P_1 \wedge P_2 \wedge \ \ldots \ \wedge P_n$ est nécessairement dû à un support vide (vis-à-vis de l'état courant de la base de données) pour au moins l'une des *sous-requêtes* de Q. La notion d'ensemble de réponses faiblement satisfaisantes est une généralisation correspondant à la présence d'une α-coupe vide de Q où α est un seuil qualitatif spécifié par l'utilisateur (et non pas seulement 0^+ comme dans le cas précédent). Dans la suite, nous utilisons la notion d'α-coupe vide pour désigner les deux cas.

Un cas extrême correspond à un noyau (1-coupe) vide pour Q uniquement. L'autre extrême correspond à la situation ou un ou plusieurs prédicats P_i ont un support (0^+-coupe) vide. Entre ces deux extrêmes, il peut exister des sous-requêtes composées de plus d'un et de moins de n prédicats qui ont une α-coupe vide. Dans la suite, on supposera qu'α prend ses valeurs dans l'échelle $\mathcal{S}$ également utilisée pour la représentation des cardinalités floues.

Définition 1. Soit Q une requête de la forme $Q = P_1 \wedge P_2 \wedge \ldots \wedge P_n$, soit S et S' deux sous-ensembles de prédicats tels que $S' \subset S \subseteq \{P_1,\ P_2,\ \ldots\ ,\ P_n\}$. Une conjonction

d'éléments de S (resp. S') est une *sous-requête* (resp. *sous-requête stricte*) de Q.

Ordonné par une relation d'inclusion, l'ensemble de toutes les sous-requêtes d'une requête conjonctive , e.g. $Q = P_1 \wedge P_2 \wedge P_3 \wedge P_4$, forme un treillis (voir figure 2), avec Q comme élément le plus grand et $\emptyset$ comme élément le plus petit.

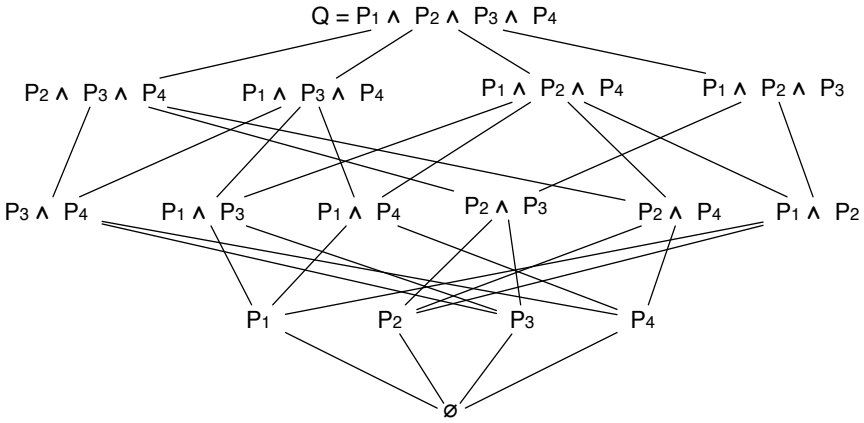

Figure 2. Treillis des sous-requêtes de $Q = P_1 \wedge P_2 \wedge P_3 \wedge P_4$

Pour expliquer l'échec (ou le semi-échec) de la requête initiale et/ou relaxer cette dernière en identifiant les sous-requêtes dont l'α-coupe est vide, on doit bien sûr imposer que de telles sous-requêtes soient minimales : une sous-requête Q' de Q constitue une explication minimale si l'α-coupe considérée est vide et s'il n'existe pas de sous-requête stricte de Q' dont l'α-coupe soit vide. Ceci correspond à une généralisation du concept de « Minimal Failing Subquery » (MFS) (Godfrey, 1997).

On note Σ_Q^α l'ensemble des réponses appartenant à l'α-coupe d'une requête Q adressée à une base de données D:

$$\Sigma_Q^\alpha = \{t \in D \mid \mu_Q(t) \geq \alpha\}.$$

Définition 2. Une *sous-requête conflictuelle minimale* d'une requête $Q = P_1 \wedge P_2 \wedge \ldots \wedge P_n$ pour un α donné est toute sous-requête Q' de Q telle que i) $\Sigma_{Q'}^\alpha = \emptyset$ et ii) pour toute sous-requête stricte Q'' de Q', on a $\Sigma_{Q''}^\alpha \neq \emptyset$.

Propriété 1. Du fait de la monotonie de la satisfaisabilité des α-coupes par inclusion, on a $\Sigma_Q^{\alpha_i} \subseteq \Sigma_Q^{\alpha_j}$ si $\alpha_i \geq \alpha_j$. En conséquence, une requête Q qui échoue pour un α_j donné échoue aussi pour les seuils plus élevés $\alpha_i > \alpha_j$. On n'a cependant pas ce phénomène de monotonie pour les sous-requêtes conflictuelles *minimales*. En effet, une sous-requête Q' peut être une MFS de Q pour un α_j donné sans toutefois être

minimale pour un seuil $\alpha_i > \alpha_j$ puisqu'une sous-requête stricte de Q' peut échouer au niveau α_i et pas au niveau α_j.

L'ensemble des MFS associées à une requête initiale Q pour un degré α sera noté MFS_α^Q.

2.4. XSS graduelles

Pour une requête floue Q qui échoue à un degré α, les MFS détectées à ce niveau constituent les explications de l'échec. Une façon de réparer Q pour obtenir un ensemble non vide de réponses à ce degré α est de relaxer Q en supprimant un ou plusieurs prédicats apparaissant dans les MFS. Ces prédicats sont choisis de telle façon que i) la sous-requête obtenue ne contienne aucune des MFS détectées, et ii) la sous-requête soit aussi proche que possible de la requête initiale, autrement dit on doit supprimer un nombre *minimal* de prédicats. À l'évidence, si la sous-requête Q' de Q ne contient aucune MFS de Q pour un seuil α donné, on a la garantie que Q' retourne un ensemble non vide de réponses. La réparation d'une requête initiale Q ayant échoué à un degré α consiste à identifier les XSS de Q au niveau α. Plusieurs XSS distinctes peuvent exister à un niveau α donné, et l'on peut envisager d'ordonner les relaxations candidates selon deux critères : leur proximité avec la requête initiale Q et le nombre de réponses retournées qui doit être aussi proche que possible de la cible k spécifiée (éventuellement) par l'utilisateur dans sa requête (cf. section 2.1).

Définition 3. Soit une requête Q qui échoue à un degré α. Une sous-requête maximale à résultat non vide (XSS en abrégé) au degré $\alpha' \leq \alpha$ est une requête $Q' \subseteq Q$ telle que $|\Sigma_{Q'}^{\alpha'}| > 0$, et telle qu'il n'existe aucune requête Q'', $Q' \subset Q''$ qui n'échoue pas au degré α'.

Propriété 2. Si une sous-requête Q' d'une requête Q est une XSS à un niveau α, alors Q' n'échouera pas non plus à un niveau quelconque $\alpha' < \alpha$. Cependant, la propriété de maximalité d'une telle XSS n'est pas garantie pour des degrés de satisfaction inférieurs. En effet, une sous-requête Q' peut être une XSS de Q au degré α sans être maximale à des degrés $\alpha' < \alpha$ puisqu'il est possible qu'une requête $Q'', Q' \in Q''$ produise un résultat non vide pour α' mais pas pour α.

L'ensemble des XSS associées à une requête initiale Q au degré α sera noté dans la suite XSS_α^Q. Évidemment, une XSS de XSS_α^Q ne peut contenir aucun prédicat (ou conjonction de prédicats) élément de MFS_α^Q.

3. Réparation de requête floue

Confronté à l'échec d'une requête conjonctive $Q = P_1 \wedge P_2 \wedge ... \wedge P_n$, le système coopératif doit d'une part fournir des explications quant à la vacuité du résultat, en utilisant les MFS classées par niveau, d'autre part proposer des relaxations sous la

forme de XSS ordonnées. L'identification de ces MFS et XSS repose sur un résumé de la partie utile de la base de données, faisant intervenir des cardinalités floues. La construction de ce résumé constitue la première étape de l'approche unifiée que nous proposons.

3.1. Construction du résumé

Soit une requête utilisateur initiale $Q = P_1 \wedge P_2 \wedge ... \wedge P_n$ ne produisant aucune réponse. La première étape de notre approche, inspirée de (Bosc, Dubois *et al*., 2002), vise à construire un résumé, sous la forme d'un ensemble de cardinalités floues associées aux prédicats présents dans Q. Ainsi, le résumé ne porte pas sur la base de données complète mais sur le résultat de la requête disjonctive $P_1 \vee P_2 \vee ... \vee P_n$. Ce résumé permet de connaître directement le nombre de n-uplets de la base qui satisfont chacune des sous-requêtes conjonctives de Q pour les différents niveaux de satisfaction de l'échelle prédéfinie $\mathcal{S}$ (voir section 2.2).

Un tel résumé, relatif à un ensemble de prédicats $\{P_1, P_2, \ldots, P_n\}$, peut comporter jusqu'à $2^n - 2$ cardinalités floues, une pour chaque sous-requête stricte non vide de Q. Il n'est bien sûr pas nécessaire de stocker la cardinalité floue associée à une conjonction de prédicats qui n'est satisfaite par aucun n-uplet de la base, même au plus faible degré possible $\sigma^f = 0^+$.

Pour calculer ce résumé, un seul parcours de la relation concernée est nécessaire. Pour chaque n-uplet t de la relation, on calcule un vecteur de degrés de satisfaction relatifs aux différents prédicats présents dans Q :

$$t = \langle \mu_{P_1}(t), \mu_{P_2}(t), ..., \mu_{P_n}(t) \rangle.$$

Ensuite, un parcours du treillis contenant les sous-requêtes de Q est effectué, du bas vers le haut en largeur d'abord (voir Figure 2), et pour chaque sous-requête stricte Q' de Q, si t satisfait quelque peu cette sous-requête ($\mu_{Q'}(t) > 0$), on met à jour la cardinalité floue associée à Q'. Plus précisément, on incrémente la cardinalité pour chaque degré σ_i, $i = 1..f$, $\sigma_i \leq \mu_{Q'}(t)$. Rappelons que pour calculer le degré de satisfaction $\mu_{Q'}(t)$ relatif à une sous-requête non atomique Q', on prend en compte le minimum des degrés de satisfaction individuels de t relatifs aux différents prédicats impliqués. À la fin du processus, une cardinalité floue est associée à chaque nœud du treillis.

Grâce à la propriété 1, on sait que si $Q \subset Q'$, alors $\mu_Q(t) = 0 \Rightarrow \mu_{Q'}(t) = 0$. Cette propriété est utilisée pour couper des branches du treillis durant la mise à jour des cardinalités floues pour un n-uplet t donné.

L'algorithme dont les grandes lignes sont décrites plus haut repose sur un parcours séquentiel des données à résumer. Cependant, les tuples pouvant être traités indépendamment et les cardinalités floues mises à jour de manière incrémentale, des architectures de traitement parallèle de type map-reduce (Dean, Ghemawat, 2008) sont utili-

sables pour construire efficacement ce résumé en répartissant les données concernées sur différentes machines. L'étape map consiste à appliquer l'algorithme de calcul des cardinalités floues sur des sous-ensembles de données réparties sur plusieurs unités de calcul et de stockage. La seconde étape, i.e. $reduce$, consiste à agréger les différentes cardinalités floues remontées en sommant le nombre de tuples trouvés dans les sous-ensembles de données pour chaque sous-requête de la requête initiale et chaque degré de satisfaction considéré (par rapport à l'échelle $\mathcal{S}$).

Exemple 1. Soit une requête floue $Q = A \wedge B \wedge C \wedge D \wedge E$ qui échoue même pour le plus petit degré $\alpha = 0^+$, où $\{A, B, C, D, E\}$ sont des prédicats flous. Le tableau 1 indique les degrés de satisfaction obtenus par cinq n-uplets vis-à-vis des prédicats présents dans Q, et le tableau 2 représente un extrait du résumé flou obtenu. $\diamond$

Tableau 1. Satisfaction de cinq n-uplets relativement aux prédicats de Q

n-uplet	μ_A	μ_B	μ_C	μ_D	μ_E
t_1	0.2	0.5	0	0	0
t_2	0.5	0.1	0	0.5	0
t_3	0	0.2	1	1	0
t_4	0.2	0.5	0.4	0	0
t_5	0	0.8	0.7	1	0

Tableau 2. Extrait du résumé fondé sur des cardinalités floues

$Q' \in \mathcal{P}(Q)$	$\lvert Q'_1 \rvert$	$\lvert Q'_{.8} \rvert$	$\lvert Q'_{.6} \rvert$	$\lvert Q'_{.4} \rvert$	$\lvert Q'_{.2} \rvert$	$\lvert Q'_{.0^+} \rvert$
A	0	0	0	0	3	3
...	...	...	...	...	...	...
D	2	2	2	3	3	3
$A \wedge B$	0	0	0	0	2	3
...	...	...	...	...	...	...
$C \wedge D$	1	1	2	2	2	2
$A \wedge B \wedge C$	0	0	0	0	1	1
...	...	...	...	...	...	...
$B \wedge C \wedge D$	0	0	1	1	2	2

3.2. Explication de l'échec

En présence d'un ensemble de réponses vide ou insatisfaisant pour une requête Q, le processus de détection de conflits décrit ci-après génère les MFS associées aux différents degrés de $\mathcal{S}$.

L'algorithme utilisé pour identifier ces MFS examine toute sous-requête de la requête initiale Q pour les différents degrés de $\mathcal{S}$. Même si un seuil qualitatif α est spécifié par l'utilisateur dans la requête Q initiale, les MFS sont détectées pour tous les degrés de $\mathcal{S}$. Il est en effet important de pouvoir détecter, par exemple, qu'il est impossible d'espérer des réponses quand une sous-requête atomique de Q échoue au degré

le moins restrictif σ_f. L'algorithme détaillé permettant d'identifier les MFS peut être trouvé dans (Smits et al., 2013). À la manière d'Apriori (Agrawal, Srikant, 1994), il commence par examiner les prédicats atomiques en considérant le degré le plus faible $\sigma_f = 0^+$. Pour déterminer si une requête atomique Q' échoue, il suffit de se reporter à la table des cardinalités floues. Si aucun n-uplet ne satisfait Q' à un degré supérieur ou égal à α_f, alors Q', en tant que requête atomique, est par définition une MFS de Q et est aussi une MFS pour un quelconque $\alpha_j > \alpha_f$. Ensuite, les conjonctions de deux prédicats non MFS sont générées, et pour chacune d'entre elles, on vérifie la cardinalité floue correspondante pour déterminer s'il s'agit d'une MFS. Si une sous-requête conjonctive Q'' est une MFS à un degré α_i, on essaie de la « propager » à des degrés de satisfaction plus élevés. Comme la propriété de minimalité d'une MFS n'est pas monotone vis-à-vis des alpha-coupes (voir propriété 1), on doit vérifier pour chaque $\alpha_j > \alpha_i$ si une sous-requête stricte de Q'' a été précédemment détectée comme étant une MFS au degré α_j. Si ce n'est pas le cas, Q'' est stockée dans la liste des MFS de Q au niveau α_j. L'algorithme revient alors à la boucle principale et les conjonctions de trois prédicats sont générées pour chaque degré de satisfaction, en prenant soin de ne considérer que des conjonctions ne contenant aucune MFS déjà identifiée. Ce processus récursif se poursuit jusqu'à ce qu'aucune conjonction candidate ne puisse plus être générée.

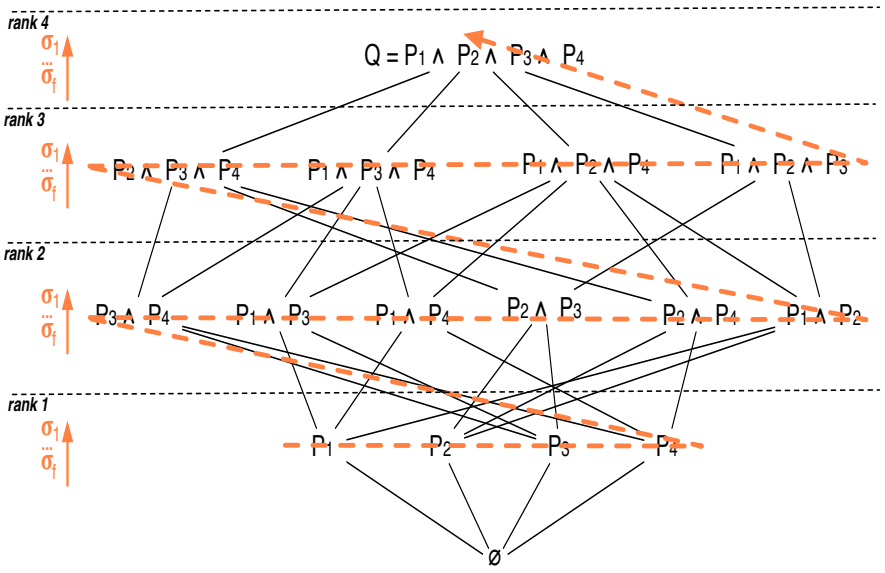

Figure 3. Parcours bas-haut en largeur du treillis pour identifier les MFS

Comme illustré dans la figure 3, la détection par niveau des MFS repose sur un parcours de bas en haut, en largeur d'abord, du treillis. Pour chaque sous-requête, on

teste la vacuité de l'α-coupe par ordre décroissant de α, $\sigma_1 = 1 \leq \alpha \leq \sigma_f = 0^+$. La propriété 1 est utilisée pour propager les sous-requêtes à réponse vide vers des degrés de satisfaction plus élevés. Le cas le pire, en termes de complexité, correspond au cas où seule Q est une MFS au niveau le plus élevé de $\mathcal{S}$, c'est-à-dire $\sigma_1 = 1$.

Exemple 2. En utilisant le résumé décrit dans le tableau 2, on peut calculer les MFS expliquant l'échec de la requête Q introduite dans l'exemple 1. Ces dernières sont présentées dans le tableau 3. $\diamond$

Tableau 3. MFS de Q classées par niveau

σ	MFS_σ^Q
$\sigma_1 = 1$	$\{A,\ B,\ E\}$
$\sigma_2 = .8$	$\{A,\ B \wedge C,\ E\}$
$\sigma_3 = .6$	$\{A,\ E\}$
$\sigma_4 = .4$	$\{A \wedge B,\ A \wedge C,\ E\}$
$\sigma_5 = .2$	$\{E,\ A \wedge B \wedge D,\ A \wedge C \wedge D\}$
$\sigma_{f=6} = 0^+$	$\{E\}$

3.3. Étape de réparation

Réparer une requête floue produisant un résultat vide (ou insatisfaisant) consiste à suggérer des XSS pour les différents degrés présents dans l'échelle $\mathcal{S}$. L'identification des XSS à un certain niveau α dépend des MFS trouvées pour ce degré α (MFS_α^Q).

Soit une requête floue Q échouant à un niveau $\alpha \in \mathcal{S}$ et MFS_α^Q les MFS détectées à ce niveau. S_α^Q désigne l'ensemble rassemblant les prédicats de Q qui n'apparaissent dans aucun élément de MFS_α^Q, et F_α^Q l'ensemble des prédicats de Q qui apparaissent dans au moins une MFS de MFS_α^Q.

Comme illustré dans la figure 4, l'algorithme utilisé pour identifier les XSS repose sur un parcours du treillis du haut vers le bas, en largeur d'abord. On commence par examiner les sous-requêtes les plus contraignantes (celles comportant le plus de prédicats) en considérant le degré de satisfaction le plus élevé σ_1. Contrairement à l'espace exploré lors de la détection des MFS, l'ensemble des sous-requêtes de Q qui sont considérées comme des XSS candidates à un niveau α peut être sérieusement élagué en utilisant les sous-requêtes de MFS_α^Q. L'ensemble des XSS candidates à explorer correspond aux sous-requêtes de Q qui peuvent être formées au moyen des prédicats $S_\alpha^Q \cup f$, où $f \in \mathcal{P}(F_\alpha^Q)$, ($\mathcal{P}(F_\alpha^Q)$ est l'ensemble des parties de F_α^Q) et f ne contient aucune MFS de MFS_α^Q. De cette façon, tous les nœuds du treillis tels que la sous-requête associée contient une MFS peuvent être ignorés, et leur cardinalité n'a pas à être testée (on sait à l'avance qu'elle vaudra zéro).

La propriété 2 est utilisée pour propager une XSS détectée à un niveau α vers les niveaux α', $\sigma_f \leq \alpha' < \alpha$, ce qui implique de tester la maximalité des sous-requêtes à résultat non vide pour les seuils de satisfaction moins élevés. Dès qu'une sous-requête

$Q' \subseteq Q$ est identifiée comme étant une XSS de Q au niveau α, on peut élaguer, en se fondant sur la définition d'une XSS, la partie du treillis composée des sous-requêtes strictes de Q'. Le cas le pire en termes de complexité correspond au cas où les seules XSS sont des sous-requêtes atomiques au niveau le plus bas σ_f.

De plus, grâce au résumé calculé précédemment, on peut afficher, pour chaque XSS, la cardinalité de son ensemble de réponses.

Exemple 3. Considérons une nouvelle fois la requête floue Q introduite dans l'exemple 1. La table 4 présente les réparations possibles de Q. $\diamond$

Tableau 4. XSS de Q ordonnées par niveau

σ	MFS_σ^Q	XSS_σ^Q
$\sigma_1 = 1$	$\{A, B, E\}$	$\{C \wedge D\}$
$\sigma_2 = .8$	$\{A, B \wedge C, E\}$	$\{B \wedge D, C \wedge D\}$
$\sigma_3 = .6$	$\{A, E\}$	$\{B \wedge C \wedge D\}$
$\sigma_4 = .4$	$\{A \wedge B, A \wedge C, E\}$	$\{B \wedge C \wedge D\}$
$\sigma_5 = .2$	$\{A \wedge B \wedge D, A \wedge C \wedge D, E\}$	$\{B \wedge C \wedge D\}$
$\sigma_{f=6} = 0^+$	$\{E\}$	$\{A \wedge B \wedge C \wedge D\}$

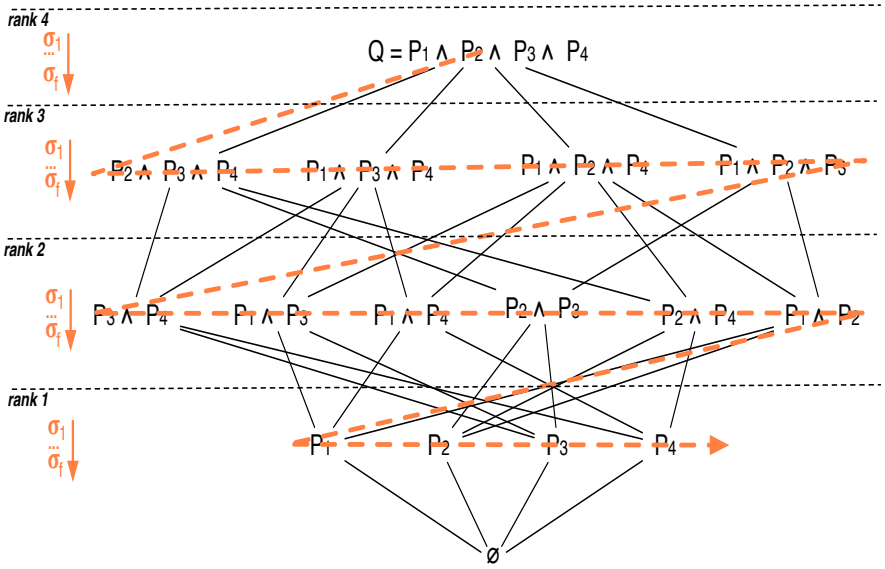

Figure 4. Parcours haut-bas en largeur du treillis pour identifier les XSS

3.4. Génération d'explications intelligibles

Les explications et les réparations possibles sont soumises à l'utilisateur de façon à ce qu'il choisisse la relaxation qui lui convient le mieux. La gradualité des MFS et des XSS améliore significativement l'aspect informatif de ce mécanisme coopératif, dans la mesure où l'utilisateur peut ajuster précisément ses attentes. De plus, grâce aux cardinalités floues constituant le résumé, on peut indiquer à l'utilisateur le nombre de réponses retournées par chaque XSS de la liste. Cette information peut fortement influencer le choix de l'utilisateur quant aux prédicats qu'il est prêt à sacrifier. L'exemple 4 illustre la structure des explications et des suggestions affichées par le système dans le cas d'un échec.

Exemple 4. Considérons à nouveau la requête de l'exemple 1, ses MFS présentées dans l'exemple 2 et ses XSS décrites dans l'exemple 3. Les explications suivantes, intelligibles par un humain, sont générées :

Aucun élément ne satisfait simultanément A, B, C, D et E pour les raisons suivantes ...

- *aucun élément ne satisfait E,*
- *aucun élément ne satisfait A et D, ainsi que B ou C à un degré ≥ 0.2,*
- *aucun élément ne satisfait A ainsi que B ou C à un degré ≥ 0.4,*
- *aucun élément ne satisfait A à un degré ≥ 0.6;*
- *aucun élément ne satisfait B et C à un degré ≥ 0.8,*
- *aucun élément ne satisfait pleinement B.*

... mais la base de données contient :

- *k_1 éléments qui satisfont pleinement C et D,*
- *k_2 éléments qui satisfont D, ainsi que B ou C, à un degré ≥ 0.8,*
- *k_3 éléments qui satisfont B, C et D à un degré ≥ 0.6,*
- *k_4 éléments qui satisfont A, B, C et D à un degré > 0.* $\diamond$

4. Expérimentation

Dans cette section, nous présentons un certain nombre de résultats expérimentaux obtenus à l'aide d'un prototype implémentant l'approche décrite ci-dessus. Les prédicats et requêtes conjonctives flous ont été définis au moyen de l'interface ReqFlex décrite dans (Smits et al., 2013a). L'objectif de cette expérimentation est d'évaluer l'efficacité de l'approche sur des données réelles, et nous avons utilisé pour ce faire une relation contenant des petites annonces sur 92 178 voitures d'occasion. Le schéma de la relation est {*prix*, *marque*, *kilométrage*, *année*, *longueur*, *hauteur*, *nbsièges*, *accélération*, *consommation*, *émissionco2*}. Les résultats présentés ci-après ont été obtenus en faisant fonctionner le prototype sur un ordinateur doté d'un Intel Core 2 Duo 2.53GHz avec 4Go 1067 MHz de ram DDR3.

Dans un premier temps, nous avons cherché à évaluer le temps nécessaire au calcul du résumé, en utilisant des jeux de données de différentes tailles. Comme on pouvait s'y attendre, et comme l'illustre la figure 5, le temps consacré à la construction du résumé varie linéairement en fonction de la taille des données pour un nombre fixé de prédicats (ici six).

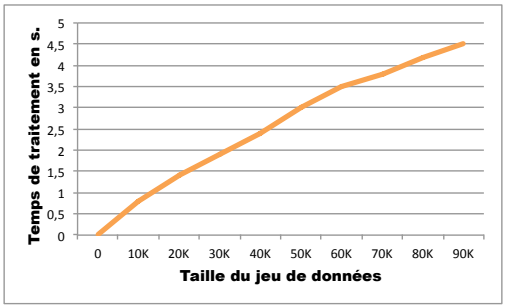

Figure 5. Calcul du résumé / taille de la base

La figure 6 montre que la taille du résumé converge très rapidement, phénomène qui s'explique par le fait que, quel que soit le nombre de n-uplets, les combinaisons possibles et sensées de propriétés pouvant servir à les décrire sont en nombre fini (il est de toute façon borné par 2^n où n est le nombre de prédicats flous considérés) et facilement énumérable (en pratique, on trouve nettement moins de 2^n cardinalités non nulles puisque de nombreuses propriétés ne sont jamais satisfaites conjointement). Ce dernier résultat a été obtenu en considérant dix prédicats pour la phase de résumé. Ainsi, même pour des bases de données volumineuses, les résumés fondés sur des cardinalités floues peuvent tenir facilement en mémoire. Rappelons également que ce n'est pas la relation concernée *dans son ensemble*, qui est résumée, mais le résultat d'une requête sur cette relation (où la conjonction de prédicats exprimée par l'utilisateur est transformée en une disjonction).

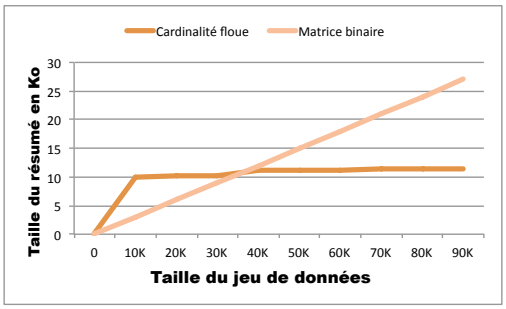

Figure 6. Taille du résumé / taille de la base

Une fois le résumé calculé et stocké en mémoire centrale, nous avons appliqué l'approche coopérative décrite plus haut pour identifier les MFS et XSS de diverses requêtes conduisant à un échec, et étudié le temps nécessaire à la réalisation de cette tâche. La figure 7 montre que pour différentes tailles de requêtes (de un à dix prédicats), le temps de détection des MFS et XSS est négligeable par rapport au temps de calcul du résumé.

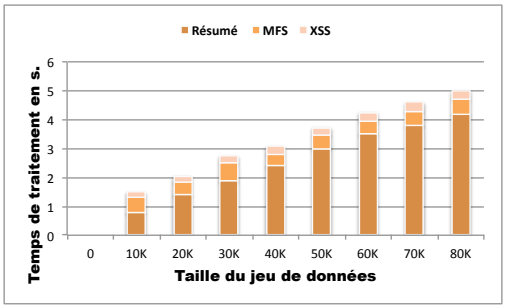

Figure 7. Temps de calcul des diverses étapes (résumé, explication, réparation)

Finalement, pour justifier l'avantage de notre approche fondée sur un résumé flou (dénotée *Cardinalité floue* sur la figure 8), nous avons comparé le temps qu'elle mettait à identifier les MFS d'une requête à résultat vide avec celui que prend une approche naïve (dénotée *Naïve* sur la figure 8) où toutes les sous-requêtes candidates sont évaluées. La figure 8, qui utilise une échelle logarithmique, montre le gain considérable obtenu.

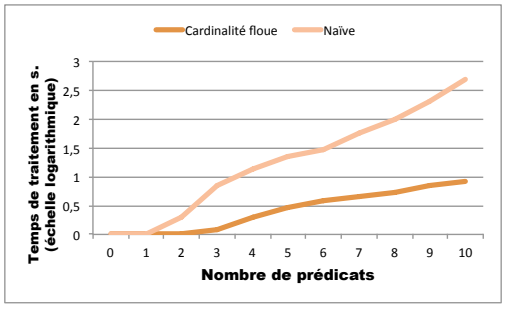

Figure 8. Approche cardinalité floue vs. approche naïve (échelle logarithmique)

Ces expérimentations montrent que l'approche proposée est peu sensible à la taille de la base de données et demeure réaliste même pour des données volumineuses, pour autant que la taille de la requête soit raisonnable. On peut donc en conclure que l'approche est tout à fait adaptée à de nombreux contextes applicatifs, par exemple le e-commerce où les sites web proposent généralement des interface permettant de spécifier au plus une demi-douzaine de conditions élémentaires.

5. Travaux connexes

La logique floue apporte des mécanismes d'interrogation flexible pouvant s'avérer utiles pour pallier l'absence de réponses à une requête. En effet, en plus d'étendre les opérateurs binaires de la logique booléenne, i.e. la conjonction et la disjonction, la dimension graduelle de la notion de satisfaction a permis le développement d'opérateurs d'agrégation offrant des sémantiques diverses voire paramétrables, comme e.g. les moyennes pondérées, l'opérateur OWA (Yager, 1993) ou les propositions quantifiées (Bosc et al., 2002). Une proposition quantifiée détermine dans quelle mesure un tuple satisfait une proportion ou un certain nombre de prédicats (Zadeh, 1983). Cette proportion ou quantité est exprimée à l'aide d'un quantificateur associant une fonction d'appartenance à un terme linguistique. Des exemples de quantificateurs relatifs représentant une proportion sont *la plupart*, *une majorité*, *plus de la moitié*, etc., alors qu'un quantificateur dit absolu représente une quantité, souvent imprécise, comme *environ six*, *quelques uns*, etc. En l'absence de résultats à une requête conjonctive $Q = P_1 \wedge P_2 \wedge \ \ldots \ \wedge P_n$, l'usage d'un quantificateur tel que *la plupart* permet de retourner l'ensemble des tuples satisfaisant tout ou une partie des prédicats $\{P_1, P_2, \ \ldots \ , P_n\}$ (Andreasen, Pivert, 1995).

Ainsi, passer d'une requête conjonctive ne retournant pas de résultat à une proposition quantifiée permet non seulement de garantir un résultat non vide, à condition qu'au moins un des prédicats de la requête initiale soit satisfait par au moins un tuple, mais également d'identifier les tuples les plus intéressants en les classant par ordre décroissant du degré $\mu_{la\ plupart(P_1, P_2, \ \ldots \ , P_n)}$, c'est-à-dire ceux satisfaisant un maximum de prédicats de la requête initiale. Cependant, cette approche a deux inconvénients majeurs. Tout d'abord elle tend à passer d'un ensemble vide de résultats à un ensemble pléthorique de résultats, mais surtout elle ne permet pas d'expliquer les raisons de l'échec de la requête initiale.

Une seconde approche envisageable pour pallier le caractère non-informatif d'un ensemble vide de réponse est de présenter à l'utilisateur des informations sur la distribution des données. Les techniques de résumés linguistiques flous (Kacprzyk, Yager, 2001) fournissent des explications linguistiques de la forme *Q des tuples sont P* où Q est un quantificateur linguistique, e.g. *la plupart*, *quelques uns*, *la majorité*, etc., et P est une conjonction de prédicats. Ainsi, ces résumés linguistiques fournissent à l'utilisateur une information intéressante sur le nombre de tuples qui satisfont les différentes sous-requêtes de la requête initiale. Générés à partir des tuples retournés par la version disjonctive ($P_1 \vee P_2 \vee ... \vee P_n$) de la requête initiale ($P_1 \wedge P_2 \wedge ... \wedge P_n$), les résumés linguistiques peuvent aider l'utilisateur à identifier une requête moins restrictive retournant des résultats, mais sans pour autant expliquer les raisons de l'échec initial.

Fournir des explications et proposer des réparations quand une requête se solde par un échec sont des questions qui ont donné lieu à de nombreux travaux dans le domaine des systèmes coopératifs. Dans (Godfrey, 1997), Godfrey étudie la complexité

calculatoire relative à la découverte d'une ou de toutes les MFS et XSS associées à une requête conjonctive. Bien qu'il soit aisé de trouver une MFS quelconque, Godfrey montre que le calcul de *toutes* les raisons minimales d'un échec est un problème NP-difficile qui conduit à exécuter un nombre exponentiel de sous-requêtes (en l'absence de tout résumé de la base de données, bien entendu). Dans le contexte des systèmes de recommandation, McSherry introduit dans (McSherry, 2004) une heuristique fondée sur la notion de « couverture » pour déterminer l'ordre dans lequel supprimer les prédicats d'une requête initiale à résultat vide. L'idée est d'identifier les prédicats les plus susceptibles d'être à l'origine de l'échec, et la couverture d'un prédicat est définie comme le nombre de MFS dans lesquel il est impliqué. Dans (Bidault *et al*., 2000), Bidault *et al.* cherchent également à identifier les explications minimales de l'échec d'une requête, et à réparer cette dernière, mais ils se placent dans le contexte spécifique d'un système de *médiation* et considèrent le cas où l'échec est dû à une violation de *contraintes* dans la requête.

Alors que les bases de données deviennent de plus en plus volumineuses, il n'est pas concevable d'exécuter un nombre exponentiel de requêtes pour établir les causes d'une réponse vide. Ce constat, qui motive notre approche, a d'abord été fait par Jannach dans (Jannach, 2008) où il propose une technique pour identifier les MFS et XSS d'une requête (booléenne) en un seul parcours de la relation concernée. Durant ce parcours, une matrice binaire est construite, qui contient autant de lignes qu'il y a de n-uplets dans la relation, et autant de colonnes qu'il y a de prédicats booléens dans la requête. Un bit 1 est affecté à une cellule si le n-uplet correspondant satisfait le prédicat concerné, 0 sinon. Cette matrice binaire est ensuite utilisée pour détecter les MFS et suggérer des réparations (XSS). Bien que la stratégie proposée par Jannach n'induise qu'un seul parcours de la collection de données, et obtient donc des temps de traitement très légèrement inférieurs à notre approche du fait de la prise en compte de conditions booléennes uniquement, elle se heurte à un problème majeur qui est que la taille de la matrice binaire dépend linéairement de la taille de la relation sous-jacente. Comme le montre la figure 6, la taille de la matrice binaire dépasse largement celle d'un résumé par cardinalités floues dès lors que la base de données est un tant soit peu volumineuse.

6. Conclusion

Un consensus existe au sein de la communauté travaillant sur les approches coopératives pour dire que des explications claires doivent être fournies en cas d'échec d'une requête. Ces explications doivent porter sur les raisons minimales de l'échec et doivent, dans l'idéal, être accompagnées de réparations correspondant aux sous-requêtes maximales permettant d'obtenir un résultat non vide. Dans cet article, nous avons proposé une approche efficace fondée sur un résumé de la partie utile de la base de données, sous la forme d'un ensemble de cardinalités floues. L'avantage principal de cette approche tient dans le fait qu'un seul parcours de la relation concernée est nécessaire pour identifier les causes de l'échec et suggérer des requêtes alternatives, aussi proches que possibles de la requête initiale, et dont on a la garantie qu'elles

n'échoueront pas. En un sens, l'approche proposée est comparable à celle proposée dans (Jannach, 2008). Cependant, nous utilisons une structure plus compacte pour résumer les données (ce qui nous permet de passer à l'échelle beaucoup plus facilement) et l'approche que nous proposons permet de traiter des requêtes *floues* tandis que celle décrite dans (Jannach, 2008) ne s'applique qu'à des requêtes *booléennes*.

En termes de perspectives, mentionnons l'intérêt qu'il y aurait à étudier des heuristiques permettant d'améliorer la détection des XSS, en exploitant par exemple des mesures de corrélation entre attributs ou un dépôt de requêtes soumises précédemment au système. Ces heuristiques pourraient influencer le choix des prédicats à écarter en priorité. Nous entendons par ailleurs procéder à une évaluation qualitative de l'approche par le biais d'une étude impliquant des utilisateurs, afin d'estimer l'intelligibilité des explications fournies.

Bibliographie

Agrawal R., Srikant R. (1994). Fast algorithms for mining association rules in large databases. In J. B. Bocca, M. Jarke, C. Zaniolo (Eds.), *Vldb*, p. 487-499. Morgan Kaufmann.

Andreasen T., Pivert O. (1995). Improving answers to failing fuzzy relational queries. In *Proceedings ot the sixth international fuzzy systems association world congress*, vol. 2, p. 345–348.

Bidault A., Froidevaux C., Safar B. (2000). Repairing queries in a mediator approach. In *Ecai*, p. 406–410.

Bosc P., Buckles B., Petry F., Pivert O. (1999). Fuzzy databases. In J. Bezdek, D. Dubois, H. Prade (Eds.), *Fuzzy sets in approximate reasoning and information systems, the handbook of fuzzy sets series*, p. 403-468. Dordrecht, The Netherlands, Kluwer Academic Publishers.

Bosc P., Dubois D., Pivert O., Prade H., Calmès M. de. (2002). Fuzzy summarization of data using fuzzy cardinalities. In *Proc. of the 9th international conference on information processing and management of uncertainty in knowledge-based systems (ipmu'02)*, p. 1553-1559. Annecy, France.

Bosc P., Liétard L., Pivert O. (2002). Evaluation of flexible queries: The quantified statement case. In *Technologies for constructing intelligent systems 1*, p. 337–350. Springer.

Bosc P., Pivert O. (1995). SQLf : a relational database language for fuzzy querying. *IEEE Transactions on Fuzzy Systems*, vol. 3, n° 1, p. 1-17.

Bosc P., Pivert O. (2000). SQLf query functionality on top of a regular relational database management system. In O. Pons, M. Vila, J. Kacprzyk (Eds.), *Knowledge management in fuzzy databases*, p. 171-190. Heidelberg, Germany, Physica-Verlag.

Dean J., Ghemawat S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, vol. 51, n° 1, p. 107–113.

Dubois D., Prade H. (1985). Fuzzy cardinalities and the modeling of imprecise quantification. *Fuzzy sets and systems*, vol. 16, p. 199-230.

Dubois D., Prade H. (2000). *Fundamentals of fuzzy sets* (vol. 7). Netherlands, Kluwer Academic Pub.

Godfrey P. (1997). Minimization in cooperative response to failing database queries. *Int. J. Cooperative Inf. Syst.*, vol. 6, n° 2, p. 95-149.

Jannach D. (2008). Finding preferred query relaxations in content-based recommenders. In *Intelligent techniques and tools for novel system architectures*, p. 81–97. Springer.

Kacprzyk J., Yager R. R. (2001). Linguistic summaries of data using fuzzy logic. *International Journal of General System*, vol. 30, n° 2, p. 133–154.

McSherry D. (2004). Incremental relaxation of unsuccessful queries. In *Advances in case-based reasoning*, p. 331–345. Springer.

McSherry D. (2005). Retrieval failure and recovery in recommender systems. *Artif. Intell. Rev.*, vol. 24, n° 3-4.

Pivert O., Bosc P. (2012). *Fuzzy preference queries to relational databases*. London, UK, Imperial College Press.

Smits G., Pivert O., Girault T. (2013a). Reqflex: fuzzy queries for everyone. *Proceedings of the VLDB Endowment*, vol. 6, n° 12, p. 1206–1209.

Smits G., Pivert O., Girault T. (2013b). Towards reconciling expressivity, efficiency and user-friendliness in database flexible querying. In *Proc. of the IEEE international conference on fuzzy systems (FUZZ-IEEE 2013).*

Smits G., Pivert O., Hadjali A. (2013). Fuzzy cardinalities as a basis to cooperative answering. In O. Pivert, S. Zadrożny (Eds.), *Flexible approaches in data, information and knowledge management*, vol. 497, p. 261-289. Springer.

Urrutia A., Tineo L., Gonzalez C. (2008). FSQL and SQLf: Towards a standard in fuzzy databases. In J. Galindo (Ed.), *In handbook of research on fuzzy information processing in databases*, p. 270-298. Hershey, PA, USA, Information Science Reference.

Yager R. R. (1993). Families of owa operators. *Fuzzy sets and systems*, vol. 59, n° 2, p. 125–148.

Zadeh L. A. (1965). Fuzzy sets. *Information and control*, vol. 8, n° 3, p. 338–353.

Zadeh L. A. (1983). A computational approach to fuzzy quantifiers in natural languages. *Computers & Mathematics with applications*, vol. 9, n° 1, p. 149–184.