Déduction dans le fragment existentiel conjonctif de la logique du premier ordre : algorithme et expérimentations

Khalil Ben Mohamed Michel Leclère Marie-Laure Mugnier

LIRMM (CNRS - Université Montpellier II)
{benmohamed,leclere,mugnier}@lirmm.fr

Résumé

Nous considérons le problème de déduction dans le fragment existentiel conjonctif muni de la négation atomique en logique du 1^{er} ordre. Ce problème peut être reformulé en termes d'autres problèmes en bases de données et en intelligence artificielle : l'inclusion de requêtes, l'implication de clauses et la déduction dans un fragment des graphes conceptuels. Partant des résultats de Leclère et Mugnier (ICDT 07), nous raffinons leur schéma d'algorithme et testons expérimentalement différentes heuristiques. L'article décrit notre démarche et présente les résultats expérimentaux obtenus.

Mots Clef

Déduction, Négation, Graphes, Homomorphisme, Algorithme, Heuristiques, Expérimentations

Abstract

We consider the deduction problem in the existential conjunctive fragment of first order logic with atomic negation. This problem can also be expressed in terms of other problems in databases and artificial intelligence : query containment, clause implication and deduction in a fragment of conceptual graphs. Based on the results of Leclère and Mugnier (ICDT 07), we refine their algorithm scheme and test experimentally several heuristics. The article describes our approach and presents the experimental results obtained.

Keywords

Deduction, Negation, Graphs, Homomorphism, Algorithm, Heuristics, Experiments

1 Introduction

Nous considérons le problème de la *déduction dans le fragment existentiel conjonctif* de la logique du 1^{er} ordre (sans fonctions) : « Étant données deux conjonctions de littéraux fermées existentiellement f et g, f se déduit-elle de g (noté $g \models f$) ? » Il peut être immédiatement reformulé sous la forme de deux problèmes fondamentaux en informatique : le problème *d'inclusion de requêtes en bases de données* [1], fondamental pour l'optimisation des mécanismes d'évaluation de requêtes [2] ou la réécriture de requêtes avec vues [6], et le problème *d'implication de clauses* qui est à la base des techniques d'ILP (Inductive Logic Programming), domaine à la croisée de l'apprentissage automatique et de la programmation logique [10][7]. Dans le cadre du problème *d'inclusion de requêtes*, nous considérons des requêtes conjonctives avec négation. Étant données deux requêtes conjonctives avec négation q_1 et q_2, il s'agit de déterminer si q_1 est incluse dans q_2, c'est-à-dire si l'ensemble des réponses à q_1 est inclus dans l'ensemble des réponses à q_2 pour toute base de données (voir [11][12][9]). Le problème *d'implication de clauses* est le suivant : « Étant données deux clauses C_1 et C_2, C_1 implique-t-elle C_2, c'est-à-dire C_2 se déduit-elle de C_1 ? » Si nous considérons des clauses du premier ordre sans fonctions (comme dans [5]), nous obtenons par contraposition une instance du problème de *déduction dans le fragment existentiel conjonctif* de la logique du 1^{er} ordre sans fonctions. Enfin, si l'on ajoute un ordre partiel sur les prédicats, on obtient exactement le problème de *déduction dans les graphes conceptuels polarisés* [8]. Ces problèmes sont Π_2^P-$complets$[1] [3][4]. Dans cet article, nous étudions expérimentalement un algorithme de résolution de cette famille de problèmes. Cet algorithme est un affinement du schéma proposé dans [9] et met en œuvre différentes heuristiques.

Plan. La section 2 définit les notions de base. La section 3 présente notre démarche et nos choix expérimentaux. En section 4, nous précisons notre algorithme et comparons expérimentalement un certain nombre d'heuristiques. Nos perspectives sont esquissées en section 5.

2 Contexte

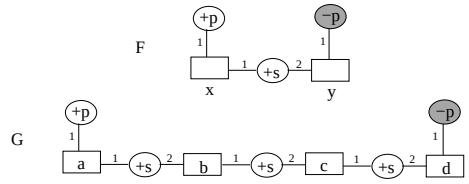
Nous nous plaçons dans le fragment conjonctif existentiel de la logique du 1^{er} ordre muni de la négation atomique, que nous noterons $FOL\{\exists, \wedge, \neg_a\}$, et nous ne considérons pas de symbole de fonction hormis des constantes. Une *formule* de $FOL\{\exists, \wedge, \neg_a\}$ est donc une conjonction de littéraux positifs ou négatifs fermée existentiellement, qu'on peut aussi voir comme un ensemble de littéraux.
Dans [9], ces formules sont vues comme des graphes étiquetés, ce qui permet de s'intéresser à leur structure et

[1] $\Pi_2^P = (co\text{-}NP)^{NP}$

manipuler des notions de graphes qui n'ont pas d'équivalent simple en logique (comme celle de graphe partiel vue plus loin). Plus précisément, une formule f est représentée par un graphe F biparti, non orienté et étiqueté, appelé *graphe polarisé*, composé de sommets termes et de sommets relations (Figure 1). Chaque terme de la formule devient un sommet terme, étiqueté soit par $\star$ si c'est une variable (omis sur les dessins), soit par la constante elle-même. Un littéral positif (resp. négatif) de prédicat r devient un sommet relation étiqueté $+r$ (resp. $-r$) et est relié aux sommets assignés à ses termes. Les étiquettes sur les arêtes correspondent à la position de chaque terme dans le littéral. Dans la suite, nous adoptons les conventions suivantes. On note $+r(c_1, \ldots, c_n)$ (resp. $-r(c_1, \ldots, c_n)$) un sommet relation étiqueté $+r$ (resp. $-r$) dont la liste de voisins est $c_1, \ldots, c_n$. Des sommets relations $+r(c_1, \ldots, c_n)$ et $-r(d_1, \ldots, d_n)$ ayant même symbole de relation mais des signes différents sont dits *opposés*. Des sommets relations opposés $+r(c_1, \ldots, c_n)$ et $-r(c_1, \ldots, c_n)$ ayant même voisinage sont dits *contradictoires*. r désigne une étiquette de relation de signe quelconque et $\overline{r}$ désigne l'étiquette opposée. Un graphe polarisé est dit *consistant* s'il ne possède pas deux sommets relations contradictoires (ce qui correspond exactement à la notion de satisfiabilité de la formule associée). Etant donnés les graphes polarisés G et F respectivement associés aux formules g et f, on note $G \models F$ *ssi* $g \models f$.

Une notion fondamentale dans cette étude est celle d'homomorphisme. Un *homomorphisme* h de F dans G est une application des sommets de F dans les sommets de G, qui respecte la bipartition (un sommet terme - resp. relation - a pour image un sommet terme - resp. relation), respecte les arêtes et leur numérotation (si rt est une arête de numéro i dans F alors $h(r)h(t)$ est une arête de numéro i dans G), conserve les étiquettes des sommets relations (un sommet relation et son image ont même étiquette) et peut « instancier » les étiquettes des sommets termes (si un sommet terme est étiqueté par une constante, son image a la même étiquette, sinon son image peut avoir une étiquette quelconque). Sur les formules f et g associées aux graphes, cette notion d'homomorphisme correspond à une substitution h de f dans g telle que $h(f) \subseteq g$. Si F et G représentent des formules positives, $G \models F$ *ssi* il existe un homomorphisme de F dans G. Par contre, dès que la négation intervient, un seul sens de cette propriété reste vrai : s'il existe un homomorphisme de F dans G alors $G \models F$. La réciproque ne l'est plus comme le montre l'exemple 1.

Exemple 1. *Considérons les formules f et g de la Figure 1. Il n'y a pas d'homomorphisme de F dans G alors que $g \models f$: en effet, si on complète g par rapport au prédicat p, on obtient la formule g' (équivalente à g) suivante : $g' = (g \wedge p(b) \wedge p(c)) \vee (g \wedge \neg p(b) \wedge p(c)) \vee (g \wedge p(b) \wedge \neg p(c)) \vee (g \wedge \neg p(b) \wedge \neg p(c))$. Chacune des 4 conjonctions de g' correspond à une façon de compléter g par rapport à p. Il existe un homomorphisme de F dans chacun des graphes qui leur sont associés. f se déduit donc de g'.*



$$f = \exists x \exists y \; (p(x) \wedge s(x, y) \wedge \neg p(y))$$
$$g = p(a) \wedge s(a, b) \wedge s(b, c) \wedge s(c, d) \wedge \neg p(d)$$

FIG. 1 – Les graphes polarisés associés à f et g.

Une façon de résoudre le problème de déduction consiste donc à générer tous les graphes polarisés « complets » que l'on peut obtenir à partir de G en utilisant les symboles de relation apparaissant dans G, puis à tester s'il existe un homomorphisme de F dans chacun de ces graphes.

Définition 1 (Graphe complet et complétion)**.** *Un graphe consistant G est dit* complet *par rapport à un ensemble de symboles de relation $\mathcal{P}$ si pour chaque symbole de relation p de $\mathcal{P}$ d'arité k et chaque k-tuple de sommets termes (non nécessairement distincts) $t_1, \ldots, t_k$ de G, G contient soit $+p(t_1, \ldots, t_k)$ soit $-p(t_1, \ldots, t_k)$. Une* complétion *G' de G est un graphe obtenu de G par ajouts successifs de nouveaux sommets relations (associés à des sommets termes de G) sans créer d'inconsistance. Chaque ajout représente une* étape de complétion*. Une complétion de G est dite* totale *si c'est un graphe complet par rapport à l'ensemble des symboles de relation considéré, sinon elle est dite* partielle*.*

Théorème 1. *[9] Soient deux graphes polarisés F et G (avec G consistant), $G \models F$* ssi *quelque soit G^c, une complétion totale de G par rapport à l'ensemble des symboles de relation apparaissant dans G, il existe un homomorphisme de F dans G^c.*

Une première propriété de [9] est que l'on peut restreindre l'ensemble des symboles de relation considérés à ceux apparaissant dans des sommets relations opposés, à la fois dans F et dans G. On appelle cet ensemble le *vocabulaire de complétion* de F et G, et on le note $\mathcal{V}$.

Une approche brutale (introduite dans [11]) consiste à calculer l'ensemble des complétions totales de G et à effectuer un test d'homomorphisme de F dans chacune. Néanmoins, la complexité d'une telle méthode est exorbitante : $\mathcal{O}(2^{(n_G)^k \times |\mathcal{V}|} \times hom(F, G^c))$, où n_G est le nombre de sommets termes dans G, k est l'arité maximum d'un sommet relation de G, $\mathcal{V}$ est le vocabulaire de complétion et $hom(F, G^c)$ est la complexité du test d'homomorphisme de F dans G^c. Ce dernier problème est *NP-Complet*. Sa résolution par un algorithme brutal est en $O(n_G^{n_F})$, où n_F est le nombre de sommets termes de F.

Deux types d'amélioration de cette méthode sont proposés dans [9]. Premièrement, une exploration de l'espace de recherche menant de G à ses complétions totales est effectuée. Cet espace de recherche est partiellement ordonné par la relation d'inclusion « sous-graphe de ». Il est exploré

sous la forme d'une arborescence binaire de racine G. Les fils d'un nœud sont obtenus en ajoutant au graphe associé à ce nœud (soit G') un sommet relation sous sa forme positive et sous sa forme négative (chacun des deux nouveaux graphes est donc obtenu par une étape de complétion à partir de G'). Au lieu de construire et tester toutes les complétions totales de G, on recherche un ensemble de complétions partielles couvrant ces complétions totales, c'est-à-dire la question de savoir s'il existe un homomorphisme de F dans chaque complétion totale de G devient : « Existe-t-il un ensemble de complétions partielles $\{G_1, \ldots, G_n\}$ tel que (1) il existe un homomorphisme de F dans chaque G_i pour $i = 1 \ldots n$; (2) chaque complétion totale G^c de G est couverte par un G_i, c'est-à-dire $G_i \subseteq G^c$? » Après chaque étape de complétion, un test d'homomorphisme de F dans la complétion courante est effectué ; si le résultat est positif, cette complétion est l'un des G_i recherchés, sinon l'exploration continue. La figure 2 donne un aperçu de cette méthode sur le cas très simple de l'exemple 1. On crée deux

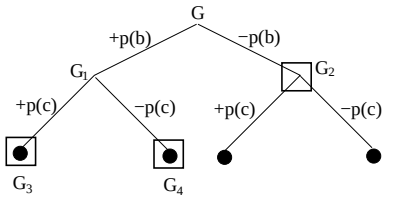


FIG. 2 – Exemple d'arbre de recherche de l'exemple 1. Chaque point noir représente un G^c et chaque carré un G_i.

nouveaux graphes G_1 et G_2, en ajoutant respectivement $+p(b)$ et $-p(b)$ à G. Il y a un homomorphisme de F dans G_2, on arrête donc de compléter G_2. Il n'y a pas d'homomorphisme de F dans G_1 : on crée deux nouveaux graphes G_3 et G_4, en ajoutant respectivement $+p(c)$ et $-p(c)$ à G_1. Il y a un homomorphisme de F dans G_3 et de F dans G_4. Finalement, l'ensemble prouvant que F se déduit de G est $\{G_2, G_3, G_4\}$ (alors qu'il existe 4 complétions totales de G par rapport à p). Un deuxième niveau d'amélioration consiste à identifier des « graphes partiels » de F (c'est-à-dire comportant tous les sommets termes de F mais pas nécessairement tous les sommets relations) pour lesquels il existe nécessairement un homomorphisme dans G quand $G \models F$. Un exemple de graphe partiel ayant cette propriété est celui des graphes partiels *purs* :

Définition 2. *Un graphe polarisé* pur *ne contient pas de sommets relations opposés (c'est-à-dire tout symbole de relation n'y apparaît que sous une forme, positive ou négative).*

Les deux objectifs principaux liés à de tels graphes partiels sont (1) de mettre en place un filtrage (s'il n'y a pas d'homomorphisme d'un graphe partiel de F dans G alors $G \not\models F$) ; (2) de les utiliser comme heuristique de choix du littéral à ajouter lors de la prochaine étape de complétion. De plus, on impose qu'un homomorphisme d'un graphe partiel de F dans G soit « compatible » avec un homomorphisme de F dans une complétion totale de G. D'où la définition suivante :

Définition 3 (Frontière, Homomorphisme compatible)**.** *Soient deux graphes polarisés F et G et F' un graphe partiel de F. Les sommets relations de $F - F'$ sont appelés* sommets relations frontières *(de F' par rapport à F). Un homomorphisme h de F' dans G est dit* compatible *par rapport à F si pour tout sommet relation frontière $p(t_1, \ldots, t_k)$, il n'existe pas de sommet relation opposé $\overline{p}(h(t_1), \ldots, h(t_k))$ dans G.* [2]

Théorème 2. *[9] Si $G \models F$ alors pour tout graphe partiel* pur *F' de F, il existe un homomorphisme compatible de F' dans G par rapport à F.*

On note F^+ (resp. F^-) le graphe partiel pur de F contenant tous les sommets relations positifs (resp. négatifs) de F. Après une étape de filtrage, notamment basée sur les graphes partiels purs (s'il n'y a pas d'homomorphisme compatible d'un graphe partiel pur de F dans G, on conclut immédiatement que $G \not\models F$), l'algorithme finalement proposé dans [9] est le suivant :

Algorithme 1 : **recCheck**(G)

Données : un graphe polarisé consistant G
Résultat : vrai si F se déduit de G, faux sinon
début
  si *il y a un homomorphisme de F dans G* **alors retourner** *vrai* ;
  si *G complet par rapport à $\mathcal{V}$* **alors retourner** *faux* ;
  Choisir $r \in \mathcal{V}$ et $t_1, \ldots, t_n$ dans G tel que $+r(t_1, \ldots, t_n) \notin G$ et $-r(t_1, \ldots, t_n) \notin G$;
  Soit G' obtenu de G en ajoutant $+r(t_1, \ldots, t_n)$;
  Soit G'' obtenu de G en ajoutant $-r(t_1, \ldots, t_n)$;
  retourner $recCheck(G')$ *ET* $recCheck(G'')$ *;*
fin

3 Démarche expérimentale

Afin de pouvoir comparer les différentes heuristiques de filtrage et de choix du littéral de complétion à chaque étape de l'algorithme `recCheck`, il est nécessaire de disposer de jeux de données et d'une méthodologie d'expérimentation. Nous décrivons dans cette section la démarche expérimentale que nous avons menée.

3.1 Les jeux de données

Nous avons tout d'abord essayé de récupérer un jeu de test standard pour le problème de déduction. Force est de constater que jusqu'à présent nous n'avons identifié aucun

[2] Pour qu'un homomorphisme compatible de F' dans G s'étende forcément à un homomorphisme de F dans une complétion totale de G, il faudrait en plus assurer la condition suivante : pour chaque paire de sommets relations opposés respectivement sur $(c_1, \ldots, c_k)$ et $(d_1, \ldots, d_k)$ de $F - F'$, $(h(c_1), \ldots, h(c_k)) \neq (h(d_1), \ldots, h(d_k))$. Toutefois, cette condition étant forcément satisfaite si F' est pur maximal pour l'inclusion, nous l'omettons dans cet article.

jeu de données réelles adapté à notre problème. Nous nous sommes alors orientés vers la génération automatique de jeux de tests et avons finalement décidé de construire un générateur aléatoire de graphes polarisés. Les paramètres de génération choisis sont les suivants :

– le nombre de sommets termes du graphe généré (c'est-à-dire le nombre de termes de la formule logique associée). Nous ne considérons pas dans notre étude les constantes[3] et ne générons donc que des variables ;
– le nombre de symboles de relation différents dans ce graphe (c'est-à-dire le nombre de prédicats) ;
– l'arité des symboles de relation (c'est-à-dire l'arité des prédicats) : dans notre étude, nous considérons uniquement des symboles de relation d'arité 2 ;
– la densité du graphe : cette notion dans notre cas représente le rapport entre le nombre de sommets relations du graphe généré et le nombre de sommets relations d'une complétion totale de ce graphe par rapport à $\mathcal{V}$;
– le pourcentage de négation : c'est-à-dire le nombre de sommets relations négatifs sur le nombre total de sommets relations du graphe généré.

Par ailleurs, nous imposons que les graphes générés soient connexes. Dans le cas contraire, cela reviendrait à traiter dans la même instance de test plusieurs problèmes de déduction. Le générateur construit un graphe complet (par rapport au nombre de symboles de relation) de n sommets termes. Les sommets relations, considérés dans un ordre aléatoire, sont alors supprimés (sous condition de maintien de la connexité) les uns après les autres, jusqu'à obtention de la densité recherchée. Les signes des sommets relations sont alors distribués aléatoirement en fonction du pourcentage de négation.

3.2 Méthodologie d'expérimentation

Le principe général des expérimentations sur données aléatoires consiste à générer un jeu de test en fixant certains paramètres et en en faisant évoluer un (ou plusieurs) autre. Pour chacune des valeurs du paramètre variable on génère un ensemble d'instances (100 instances pour nos expérimentations) et on lance la résolution de ces instances. On effectue alors des mesures sur les résultats obtenus que l'on présente sur des courbes prenant en abscisse le paramètre variable et en ordonnée la mesure calculée. Pour générer une instance du problème de déduction, nous générons deux graphes aléatoires, respectivement le source, F, et le cible, G, et cherchons à savoir si F se déduit de G. Une notion importante est celle de la mesure de la difficulté des instances du problème à résoudre. En effet, si l'on souhaite comparer différentes techniques de résolution, il semble préférable de le faire sur des instances « difficiles » du problème plus à même de discriminer les heuristiques. Nous avons donc procédé par essais successifs avec l'algorithme `recCheck` pour essayer de caractériser l'influence des paramètres sur la difficulté du problème. La difficulté a été estimée en calculant le temps d'exécution de la résolution et la taille de l'arbre de recherche exploré. Nous retenons la médiane de l'ensemble des résultats obtenus au lieu de la moyenne, ceci en vue d'évacuer les quelques instances très difficiles (produisant un timeout) non représentatives de l'ensemble des instances étudiées (fait révélé par l'expérience).

Le nombre de sommets termes a été fixé à 10. Moins de sommets termes et le problème était trop vite résolu, plus et il devenait trop long à résoudre. Nous avons également fixé un timeout à 1 minute. Nous avons alors étudié l'influence des densités des graphes source et cible sur la difficulté avec 1 symbole de relation et 50% de négation. On peut observer que contrairement à ce qu'on aurait pu penser, la taille de l'arbre exploré n'est pas corrélée au temps de calcul (cf. figure 3). Ceci peut s'expliquer par la difficulté variable du test d'homomorphisme d'une instance à l'autre, notamment en présence d'un seul symbole de relation (le nombre d'images possibles de chaque sommet du graphe source est très important). On observe des pics de difficulté lorsque la densité du cible est environ 3 fois plus importante que celle du source. Par ailleurs la difficulté est observée maximale pour une densité du source de 15% (17% si l'on considère le temps).

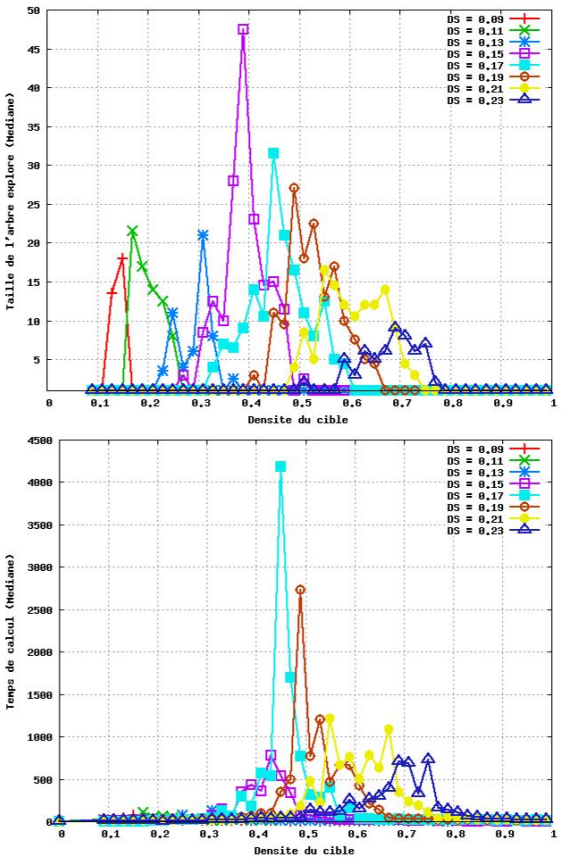


FIG. 3 – Influence des densités sur le temps de calcul et la taille de l'arbre exploré : $nbSymb = 1$, $neg = 50\%$.

[3] L'existence de constantes a tendance à rendre le problème plus facile dans la mesure où cela limite les possibilités d'homomorphismes mais ce paramètre n'influe pas sur les heuristiques comparées.

Concernant les symboles de relation, on s'attend à ce que l'ajout de nouveaux symboles de relation dans les graphes amène une réduction du temps de calcul et de l'espace de recherche exploré. Bien que le nombre de complétions totales possibles augmente fortement ($((2^{n^2})^{nbSymb}$ complétions pour $nbSymb$ symboles de relation), on s'attend à ce que la déduction ou l'absence de déduction soit détectée plus tôt dans l'arbre de recherche, le graphe source devenant beaucoup plus spécifique. Ces intuitions sont prouvées expérimentalement. On observe par ailleurs que les coûts en temps et en taille de l'arbre exploré sont désormais directement corrélés laissant ainsi penser que l'influence du temps de calcul de l'homomorphisme se fait moins sentir. La figure 4 qui reprend l'expérimentation précédente en considérant 3 symboles de relation (au lieu d'un seul) illustre ce phénomène. On notera que les densités du source entraînant des difficultés maximales sont à présent 3 fois moins importantes puisque les complétions totales sont 3 fois plus grandes. Par ailleurs, les pics de difficulté semblent se produire lorsque les densités du cible sont environ $3 * nbSymb$ fois plus grandes que celles du source.

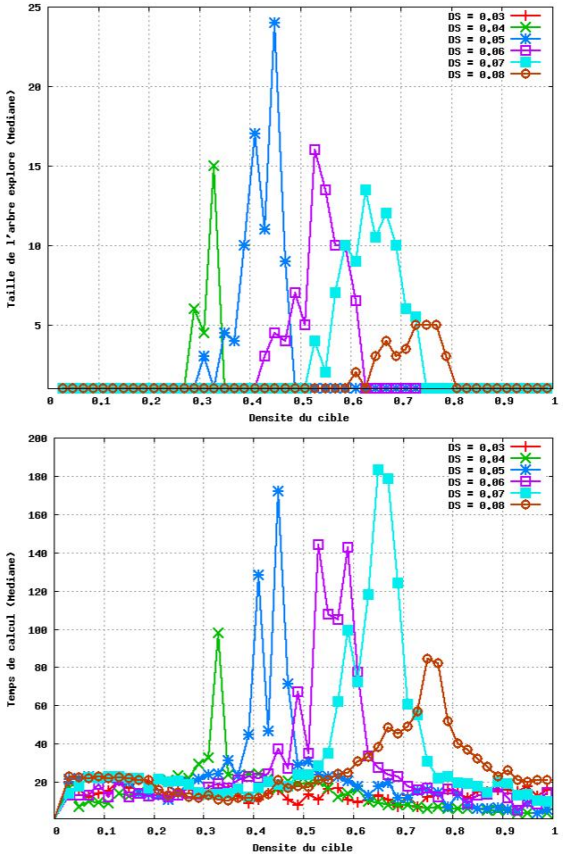


FIG. 4 – Influence des densités sur le temps de calcul et la taille de l'arbre exploré : $nbSymb = 3$, $neg = 50\%$.

Concernant le pourcentage de négation, on s'attend intuitivement à ce que le problème soit plus dur quand il y a autant de sommets positifs que de sommets négatifs pour chaque symbole de relation. Ceci est confirmé expérimentalement (cf. figure 5).

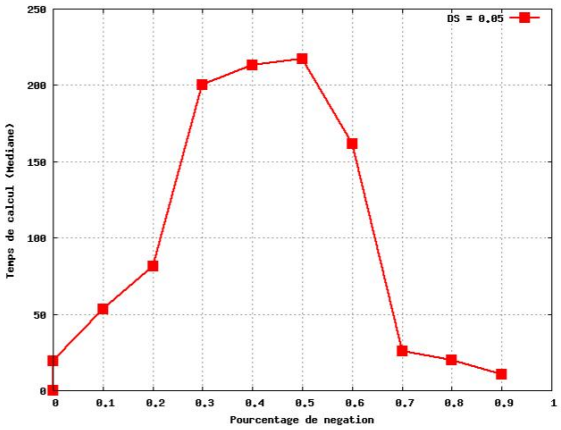


FIG. 5 – Influence du pourcentage de négation en fixant : $DS = 5\%$, $DC = 45\%$, $nbSymb = 3$.

Dans la suite de ce papier toutes les expérimentations seront réalisées en fixant le nombre de sommets termes à 10, la densité du source à 5%, le nombre de symboles de relation différents à 3 et le pourcentage de négation à 50%. Les mesures seront exprimées selon différentes densités du cible.

4 Affinement de l'algorithme *recCheck* et expérimentations

Comme mentionné dans la section 2, l'espace de recherche menant de G à ses complétions est partiellement ordonné par la relation d'inclusion. L'algorithme *recCheck* l'explore par un parcours en profondeur, qui définit une arborescence binaire de racine G (cf. figure 2). Le parcours en profondeur de l'espace de recherche a été privilégié par rapport à d'autres formes de parcours plus ou moins en largeur d'une part pour des raisons d'espace mémoire, d'autre part pour tenter d'aboutir plus rapidement à un échec.
Dans cette partie, nous proposons et testons expérimentalement trois affinements de l'algorithme *recCheck* visant à élaguer l'espace de recherche. Les deux premiers concernent la façon d'explorer l'espace de recherche alors que le troisième porte sur la notion de filtrage. L'algorithme obtenu est appelé *recCheckPlus* : cf. Algorithme 2 ; il est initialement appelé avec $(G, \emptyset)$.

4.1 Choix du prochain sommet relation à ajouter (1)

La résolution du problème s'effectuant par un parcours en profondeur de l'espace de recherche, le choix du prochain sommet relation à ajouter à G est primordial, comme l'illustre la figure 6 : avec le choix de l'atome de complétion $s(e, b)$, trois nœuds suffisent à prouver que $G \models F$. En effet il y a un homomorphisme de F dans G' ($h' = \{w \mapsto e, x \mapsto b, y \mapsto e, z \mapsto b\}$) et un homomorphisme de F dans G'' ($h'' = \{w \mapsto f, x \mapsto e, y \mapsto b, z \mapsto c\}$). Dans le

Algorithme 2 : recCheckPlus(G,h)

Données : un graphe polarisé consistant G, un homomorphisme compatible h de F' dans le père de G (vide si G est la racine)
Accès : un graphe polarisé F, un graphe partiel pur F' de F, le vocabulaire de complétion $\mathcal{V}$
Résultat : vrai si F se déduit de G, faux sinon
début
  si *il y a un homomorphisme de F dans G* **alors retourner** *vrai* ;
  si *G complet par rapport à $\mathcal{V}$* **alors retourner** *faux* ;
(3) **si filtrageDynamique(G) =** *échec* **alors retourner** *faux* ;
(1) $l, h \leftarrow$ **choixLitteralCompletion(G,h)** ;
  si $l =$ *échec* **alors retourner** *faux* ;
  Soit G' obtenu de G en ajoutant $\bar{l}$;
  Soit G'' obtenu de G en ajoutant l ;
(2) **retourner recCheckPlus(**$G', \emptyset$**)** *ETALORS* **recCheckPlus(**G'', h**)** *;*
fin

pire des cas (construction de l'ensemble des complétions totales de G), plus de 2 milliards ($2^{(6^2-5)} = 2^{31}$) de complétions totales auraient été construites, soit le double de nœuds parcourus ($2^{31} + (2^{31} - 1)$).

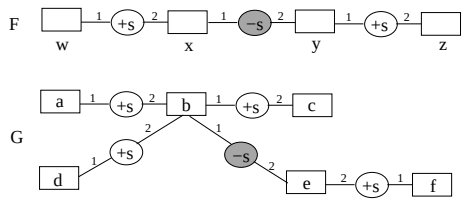


FIG. 6 – Un bon choix d'atome de complétion : $s(e, b)$

Le choix du prochain sommet à ajouter consiste plus précisément à choisir un symbole de relation $p \in \mathcal{V}$ d'arité n, ainsi qu'une liste de sommets termes (non nécessairement distincts) $t_1, \ldots, t_n$ de G tel que $+p(t_1, \ldots, t_n) \notin G$ et $-p(t_1, \ldots, t_n) \notin G$. Les fils de G sont obtenus en lui ajoutant respectivement $+p(t_1, \ldots, t_n)$ et $-p(t_1, \ldots, t_n)$. Une façon brutale de procéder consiste à faire un choix aléatoire de p et $t_1, \ldots, t_n$. Une autre méthode est de guider le choix par un homomorphisme compatible. Cette approche est notamment motivée par le fait que s'il ne manque que quelques sommets relations à G afin qu'un homomorphisme compatible de F' dans G devienne un homomorphisme de F dans une complétion G' de G, ces sommets relations manquants devraient être ajoutés en priorité afin d'éliminer des parties de l'espace de recherche (toutes les complétions incluant G'). Le sous-algorithme *choixLittéralCompletion* (Algorithme 3) s'appuie sur ce guidage. Cet algorithme prend en entrée un graphe polarisé G et retourne un littéral qui servira à compléter G. Il a en accès F et F', un graphe partiel pur de F maximal pour l'inclusion, que l'on appelle par la suite *graphe partiel de guidage*. Il recherche un homomorphisme compatible de F' dans G (par la fonction *recHomComp(F,F',G)*), soit h, que l'on appelle par la suite *homomorphisme compatible de guidage*. S'il n'y en a pas, il retourne un échec. Sinon, il choisit un sommet relation $p(t_1, \ldots, t_n) \in F - F'$ tel que $+p(h(t_1), \ldots, h(t_n)) \notin G$ et $-p(h(t_1), \ldots, h(t_n)) \notin G$ et retourne le littéral de complétion $p(h(t_1), \ldots, h(t_n))$ (le signe de ce littéral sera utilisé par une autre heuristique, voir Sect. 4.2). Comme on pouvait s'y attendre, cette méthode s'avère bien meilleure que le choix aléatoire.

Algorithme 3 : choixLitteralCompletion(G, h)

Données : G un graphe polarisé consistant, h un homomorphisme compatible de F' dans le père de G ou $\emptyset$
Accès : F, F' un graphe partiel pur de F maximal pour l'inclusion
Résultat : un littéral de complétion de G s'il existe et l'homomorphisme compatible de guidage de G, échec sinon
début
  si $h = \emptyset$ **alors**
    $h \leftarrow$ **recHomComp(F,F',G)** ;
    si $h =$ *échec* **alors retourner** *échec* ;
  Choisir un sommet relation frontière $r(t_1, \ldots, t_n) \in F - F'$ tel que $r(h(t_1), \ldots, h(t_n)) \notin G$ et $\overline{r}(h(t_1), \ldots, h(t_n)) \notin G$;
  retourner $r(h(t_1), \ldots, h(t_n))$, h *;*
fin

4.2 Choix du fils à parcourir en priorité (2)

Au fil des expérimentations, nous avons constaté que pour un nœud donné et après choix du littéral de complétion, le choix du fils à parcourir en priorité est important. Ceci est illustré par les courbes de la figure 7. Le graphe partiel de guidage est F^+ (les sommets frontières sont donc tous négatifs). La première courbe ("Positif-filsPositif") est obtenue en donnant la priorité au fils obtenu par ajout d'un sommet relation positif, et la seconde ("Positif-filsNegatif") au fils obtenu par ajout d'un sommet relation négatif. Il se dégage de ces courbes que la priorité donnée à la branche positive de l'arbre de recherche, pour le graphe partiel de guidage F^+, est beaucoup plus intéressante, que ce soit au niveau du temps de calcul ou de la taille de l'arborescence explorée.

Pour un homomorphisme compatible h de F' (un graphe partiel de F) dans G, on peut classer les sommets relations frontières de F' en deux catégories : ceux qui étendent h et les autres. On dit que $r(c_1, \ldots, c_k) \in F - F'$ étend h si $r(h(c_1), \ldots, h(c_k)) \in G$. L'algorithme *choixLittéralCompletion* retourne toujours un sommet relation frontière $r(c_1, \ldots, c_k)$ appartenant à la seconde catégorie. Étant donné ce sommet $r(c_1, \ldots, c_k)$, on construit deux graphes : l'un en ajoutant à G le sommet $r(c_1, \ldots, c_k)$, qu'on ap-

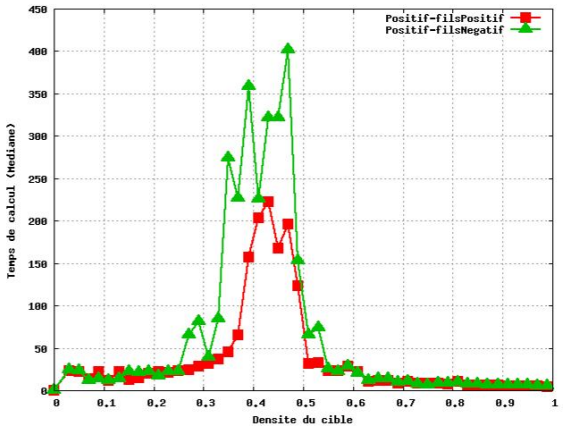


FIG. 7 – Comparaison des choix de nœud fils prioritaire.

pelle une *rel-extension de h*, l'autre en ajoutant à G le sommet $\overline{r}(c_1, \ldots, c_k)$, qu'on appelle une *rel-contradiction de h* (cf. Exemple 2). Plus généralement, il apparaît intéressant de choisir en priorité, à chaque pas dans l'arbre de recherche, le nœud contenant la *rel-contradiction* de l'homomorphisme compatible de guidage de son père. Intuitivement, cela revient à rechercher un échec plus rapidement, en supprimant pour le nœud fils un des homomorphismes compatibles de son père.

Exemple 2. *Considérons les graphes F et G de la figure 1 et le graphe partiel de guidage $F^+ = \{+p(x), +s(x, y)\}$. $h = \{x \mapsto a, y \mapsto b\}$ est un homomorphisme compatible de F^+ dans G. Soient G_1 et G_2 obtenus de G en ajoutant respectivement $+p(b)$ (une* rel-contradiction*) et $-p(b)$ (une* rel-extension*). Nous parcourons donc G_1 avant G_2.*

Mentionnons une amélioration complémentaire. Le sous-algorithme *recHomComp(F,F',G)* retourne le même homomorphisme compatible s'il est lancé avec les mêmes paramètres. Il n'est donc pas nécessaire de recalculer l'homomorphisme compatible de guidage sur chaque nœud de l'arbre de recherche. Soit G', un fils de G, et h l'homomorphisme de guidage de G. Dans le cas où le dernier sommet relation ajouté à G' est une *rel-extension de* h, h devient l'homomorphisme compatible de guidage de G'. (cf. Exemple 2 : l'homomorphisme compatible de guidage de G devient celui de G_2).

4.3 Filtrage dynamique (3)

Afin de détecter un échec plus rapidement, nous ajoutons un filtrage dynamique, qui sera effectué à chaque nœud de l'arbre de recherche. Ce filtrage correspond à un test d'homomorphisme compatible d'un, voire plusieurs, graphes partiels purs de F dans G. Dans le cas où il n'y a pas d'homomorphisme compatible d'un de ces graphes partiels purs dans G, on conclut que $G \not\models F$. Dans la suite, on appelle ces graphes des *graphes partiels de filtrage*. Notons que, pour un graphe partiel de filtrage F', il n'est nécessaire de relancer le test d'homomorphisme compatible de F' dans G', la complétion courante de G, que dans le cas où le dernier sommet relation ajouté est une *rel-contradiction* de l'homomorphisme compatible de F' dans le père de G'.

Algorithme 4 : filtrageDynamique(G)

Données : un graphe polarisé consistant G
Accès : un ensemble de couples $L = \{(F_1'', h_1''), \ldots, (F_n'', h_n'')\}$ où F_i'' est un graphe partiel de filtrage et h_i'' un homomorphisme compatible de F_i'' dans le père de G (vide si G est la racine)
Résultat : succès ou échec du filtrage
début
  pour chaque $(F'', h'') \in L$ **faire**
    si h'' *est contredit par le dernier sommet relation ajouté à* G **alors**
      $h'' \leftarrow$ **recHomComp(F,F",G)** ;
      si h'' = *échec* **alors retourner** *échec* ;
  retourner *succès* ;
fin

4.4 Expérimentations de *recCheckPlus*

Nous exposons ici les résultats expérimentaux obtenus pour l'algorithme *recCheckPlus*. Le graphique de la figure 8 illustre les résultats obtenus pour différents choix de graphe partiel de guidage (aucun graphe partiel de filtrage n'est utilisé) :

– Positif : F^+ comme graphe partiel de guidage ;
– Negatif : F^- comme graphe partiel de guidage ;
– Max : un graphe partiel pur de cardinalité maximale (notation F^{Max}) comme graphe partiel de guidage.

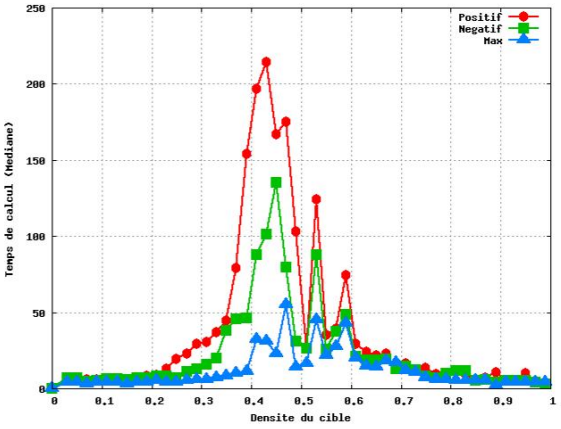


FIG. 8 – Comparaison des heuristiques de guidage.

Il se dégage de ces courbes la préférence d'un graphe partiel pur de cardinalité maximale comme graphe partiel de guidage. Nous partons de cette observation et ajoutons des graphes partiels de filtrage. Les graphiques de la figure 9 illustrent les résultats obtenus :

– Max : un F^{Max} comme graphe partiel de guidage et aucun graphe de filtrage ;

– Max-MaxBarre : un F^{Max} comme graphe partiel de guidage et $\overline{F^{Max}}$ (le graphe partiel sur les sommets relations de $F - F^{Max}$) comme graphe partiel de filtrage ;
– Max-Purs : un F^{Max} comme graphe partiel de guidage et l'ensemble des graphes partiel purs de F maximaux pour l'inclusion comme graphes partiels de filtrage.

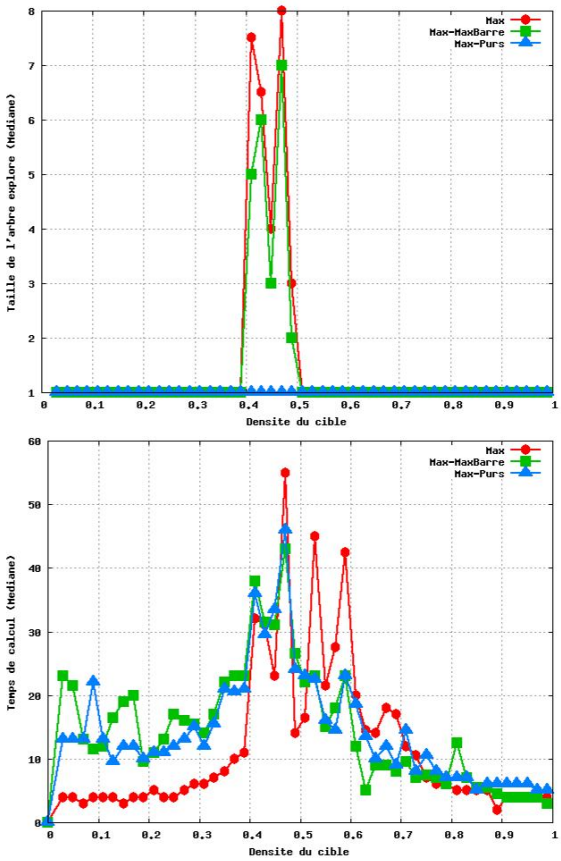


FIG. 9 – Comparaison des heuristiques de filtrage.

On constate que plus le filtrage dynamique est important, plus la taille de l'arbre de recherche est petite. En contrepartie le temps de calcul est plus important pour les densités du cible inférieures à 50%.
On en conclut que les meilleurs résultats concernant la taille de l'arbre de recherche exploré sont attendus avec un graphe partiel pur de cardinalité maximale pour guider la complétion et l'ensemble des graphes partiels purs maximaux pour l'inclusion pour filtrer dynamiquement. Néanmoins ces derniers peuvent entraîner un coût supplémentaire au niveau du temps de calcul pour certaines densités du cible.

5 Perspectives

Dans ce papier, nous avons étudié le problème de déduction dans le fragment existentiel conjonctif de la logique du premier ordre. Nous nous sommes basés sur le schéma d'algorithme proposé par [9] et l'avons affiné, proposant alors l'algorithme *recCheckPlus*. Nous avons comparé expérimentalement, à l'aide de graphes polarisés générés aléatoirement, différentes heuristiques allant du choix de l'exploration de l'arbre de recherche au choix du graphe partiel de guidage et des graphes partiels de filtrage.
Des heuristiques complémentaires ont été développées, toutefois nous ne les avons pas encore testées expérimentalement. Elles concernent le choix du littéral de complétion parmi tous les sommets relations frontières et l'affinement de la recherche d'homomorphisme compatible.
Nous envisageons de considérer des sommets relations d'arité quelconque. Nous voulons également pouvoir construire des graphes polarisés *safe* (c'est-à-dire tels que tout sommet terme est voisin d'au moins un sommet relation positif), notion incontournable pour le problème d'inclusion de requêtes conjonctives en bases de données, notamment en vue d'implémenter et de comparer l'algorithme proposé par [12] au nôtre.
Enfin, nous souhaitons affiner la notion de dureté d'une instance et rechercher l'existence d'une transition de phase pour le problème étudié.

Références

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases : The Logical Level*. Addison-Wesley, 1995.

[2] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *9th ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[3] M. Chein and M.-L. Mugnier. *A graph-based approach to knowledge representation and reasoning : computational foundations of conceptual graphs*. 2007.

[4] C. Farré, W. Nutt, E. Teniente, and T. Urpí. Containment of conjunctive queries over databases with null values. In *ICDT 2007*, volume 4353 of *LNCS*, pages 389–403. Springer, 2007.

[5] G. Gottlob. Subsumption and implication. *Inf. Process. Lett.*, 24(2) :109–111, 1987.

[6] A. Y. Halevy. Answering queries using views : A survey. *VLDB Journal*, 10(4) :270–294, 2001.

[7] N. Lavrac and S. Dzeroski. *Inductive Logic Programming : Techniques and Applications*. Ellis Horwood, 1994.

[8] M. Leclère and M.-L. Mugnier. Simple Conceptual Graphs with Atomic Negation and Difference. In *Proc. of ICCS'06*, volume 4068 of *LNAI*, pages 331–345. Springer, 2006.

[9] M. Leclère and M.-L. Mugnier. Some Algorithmic Improvments for the Containment Problem of Conjunctive Queries with Negation. In *Proc. of ICDT'07*, volume 4353 of *LNCS*, pages 401–418. Springer, 2007.

[10] S. Muggleton. *Inductive Logic Programming*. Academic Press, 1992.

[11] Jeffrey D. Ullman. Information Integration Using Logical Views. In *Proc. of ICDT'97*, volume 1186 of *LNCS*, pages 19–40. Springer, 1997.

[12] F. Wei and G. Lausen. Containment of Conjunctive Queries with Safe Negation. In *International Conference on Database Theory (ICDT)*, 2003.