集合差演算を含む問合せに対する why-not provenance

朴　柱英†　吉川　正俊††

† 京都大学 情報学研究科 社会情報学専攻　〒 606-8501 京都市左京区吉田本町
†† 京都大学 情報学研究科 社会情報学専攻　〒 606-8501 京都市左京区吉田本町
E-mail: †pjy953@db.soc.i.kyoto-u.ac.jp, ††yoshikawa@i.kyoto-u.ac.jp

あらまし　why-not provenance は問合せの結果に対する利用者の「なぜ、このレコードは結果に含まれていないのか？」という疑問に答えるためのものである。結果と利用者の知識が一致しない時は、(1) 利用者の知識が間違っている、または、(2) 問合せ結果が実世界のデータを反映していない、のいずれかが理由である。また、(2) は (2a) 利用者が与えた問合せが間違っている、(2b) データベースに対象とするデータが存在しない、の二通りに分けることができる。今回、我々は問合せとレコードの条件（これを Why-not question と呼ぶ）が与えられたときに，そのレコードが問合せ結果に含まれない原因になる部分を探すことを目的にする。その結果、上述の (1) と (2) の状況に対応することができる。状況 (1) と (2) はアルゴリズムの前で与えられる外部条件により決まる。従来の研究では問合せとして SPJUA(Selection, Projection, Join, Union, Aggregation) を対象とするものはあるが，問合せに複数個以上の集合差を含むものはない。我々は SPJU に加え集合差を含めた問合せを対象とする why-not provenance アルゴリズムを提案する。このアルゴリズムで一番重要な考え方は why-not question と問合せのどこが衝突しているかを探すことである。衝突しているところを探すために、元の問合せに why-not question の条件を追加した問合せを求め、元の問合せの一部を削除する方法で原因を探す。これは元のデータベースに why-not question が対象にするデータが含まれている限り、結果は必ず得られる。また、100MB 以下のデータベースに対して実時間で答を求めることができることを示す。問合せの修正やインスタンス修正は今後の課題である。

キーワード　データベース理論、問合せ処理、問合せ言語、データの provenance、why-not provenance

1. 序　　論

Provenance の研究は Why(結果がなぜ出たのか？), How(結果がどうして出たのか？), Where(結果がどこから出たのか？) などに始まり最近は Why-not(結果がなぜ出ていないのか？) に関する研究も行われている。結果に対して provenance を知ることでシステム管理者はデバッギングやシステムメインテナンスが楽になり、利用者はシステムへの理解度を高めると同時に、自分の間違った知識を修正できるようになる。これらいずれの場合でも、利用者が結果を詳しく理解できるようになることを意味する。

我々の論文は provenance の中で Why-not に関するものを扱っている。Why-not に関する研究は [12] を始め、最近活発に行われている。Why-not Provenance に関する研究は大きく二つの分野に分けることができる。一つ目の分野は Why-not Provenance を求めることを重点にする研究分野で、もう一つの分野 [13] は Why-not Provenance が利用者にどのような影響を与えるかに関する研究分野である。本論文は前者に含まれる。前者の研究分野で行われている研究は多様な SQL 文に対して Why-not Provenance を求めること、Why-not Provenance の計算を高速化すること、及びそれらの応用を含む。

対象とする問合せの表現力を増強する研究は [12] の SPJ を最初とし、[10] の SPJUA まで研究され、最近は計算速度高速化の研究も行われている。[9] で集合差演算を含む問合せを対象としているが、ただ一つの集合差演算を含む問合せに限られる。それゆえ、まだ複数個以上の集合差演算 (Set Difference) を含む問合せを対象とした研究はない。そこで、我々は今回複数個以上の集合差演算を含む問合せに対応するための Why-not Provenance アルゴリズムを提案する。また、従来のアルゴリズムが持っている問題点としては問合せを処理する時、与えられた問合せの処理順序を木構造で表現することである。その結果、Why-not Provenance を計算するアルゴリズムは作ら

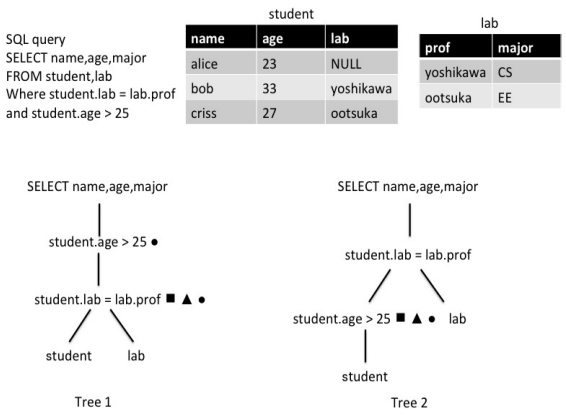

図 1　データと SQL 問合せの例、および SQL 問合せの木構造、■ は [5]、▲ は [3]、● は [9] のアルゴリズムで計算される原因である

れる木の形に影響を受けてしまう。その点を指摘し、解決するためのアルゴリズムを提案した論文が [4] であるが、そこでもまだ集合差を含む問合せは対象としてしていない。木の形に依存する従来の研究の問題点は図 1 に例示される。図 1 の Tree1 と Tree2 は与えられた SQL Query の異なる二つの処理法を表す木構造の一部である。しかし、従来のアルゴリズムによると、“なぜ、名前が alice である人は結果に含まれていない ”の Why-not Question に対して Tree1 と Tree2 で計算される Why-not Answer は異なる。Tree2 の場合は、[3], [5], [9] が “student.age>25” を原因として指摘している。それに比べて、Tree1 の場合は [3], [5] は “student.lab = lab.prof” を原因として指摘し、[9] は両方を原因として指摘している。それは木構造と [5] で定義された Successor(結果から逆にトレースして得られる関係）に依存する既存アルゴリズムの問題である。しかし、我々が提案するアルゴリズムを用いると、木構造に依存せず両方を原因として得ることができる。

以降、2 章で関連研究についてまとめ、3 章で理論的な背景を説明し、4 章ではアルゴリズムを提案する。また 5 章では 4 章で提案したアルゴリズムの実験結果とその考察を扱う。最後に 6 章では結論とこれからの課題について述べる。

2. 関 連 研 究

Why-not Provenance の最初の研究は Huang ら [12] によるとされている。また、“Why-not” という単語は [5] で始めて使われた。Why-not Provenance の研究の動機として言われるのはシステムと問合せ結果に対する利用者の疑問である。疑問は利用者が持っている知識と問合せ結果が一致しないため発生する。疑問の原因として言われるのは二つ存在する。一つは問合せの中身で、もう一つは問合せの対象になるデータベースである。そのため、Why-not Provenance に関する研究も各々のケースに分けて行われている。

まず、Why-not Provenance の研究は技術的に三つに分かれる。一番目は、Query-based explanations である。Query-based explanations は利用者の疑問の原因になる問合せの一部を探すことを目的にする。問合せやデータベースが間違っているのではなく、ただ問合せの一部が原因になって、利用者が期待する答が結果に現れないということを意味する。この、Query-based explanations はただ探すだけで、修正のための方法は提案していない。[5], [4], [3] が Query-based explanations の研究であり、[5] は SPJU、[3] は SPJUA、[4] は数学の多項式を基盤にして why-not provenance を計算している。その結果木構造に依存しない計算結果を与えている。しかし、いずれの研究でも集合差は対象にしていない。二番目は、Instance-based explanations である。Instance-based explanations は問合せの原因が Instance、すなわちデータベースにあると仮定する。実世界ではいろいろなデータがインタネット上に存在する。そのデータを集めてデータベースに保存しておく時もある。その場合、データが事実でない場合もあり得る。従って、利用者が正しい問合せを入力しても利用者が期待している結果が得られない可能せもある。そこで、その考え方に基づいて問題になる Instance を探し修正したり、新しい Instance を挿入したりする。[12], [11], [14] が Instance-based explanations の研究で、[12] は SPJ, [11] は SPJUA, [14] は conjunctive queries を対象にしている。Query-based explanations と同様に、集合差に関する研究はまだ行われていない。最後は Modification-based explanations である。Modification-based explanations は問合せが間違ったという仮定で問題になる問合せの部分を探し、それを修正することで、Why-not Question を答に含めさせるためのアルゴリズムを提案する。例えば、実世界ではシステムのエラーにより利用者の入力がそのまま問合せとして使われず他の値が問合せに使われる場合もあり、利用者が自分の入力として期待している答が結果に現れない可能性もある。そこで、問題になる問合せを見つけて修正をする。[16], [8], [17], [7] が Modification-based explanations の研究で、[16] は SPJA, [8] は Top-k 問合せ、[17] は Reverse Skyline 問合せ [7] は Reverse Top-k 問合せを対象にしている。Query-based と Instance-based は互いに独立なアプローチであるが、これらを統合するための研究もある。その例としては [9] があり、hybrid explanation と呼ばれる方法を提案している。

また、Why-not Provenance の技術的な研究ではないが、Why-not を知ることにより利用者のシステムへの理解度を高めることができるという研究も [13] で行なわれている。[13] の結果によると、利用者は Why と How と Where より Why-not を知ることがシステムへの理解度が高くなるという実験結果を示している。また、Why-not の応用研究もいくつか行われている。[2] では画像検索で Why-not の概念を使い、[6] では地理検索で Why-not の概念を使っている。また、SQL のみの説明に限定せず ontology を使った説明に関する研究 [15] も行なわれている。

3. 基本的事項

本論文では datalog を使って理論的な説明を行う。再帰問合せを使わない限り SQL 問合せと datalog 問合せは相互に変換することができ、同じ結果を出す。第 3 章では我々らのアルゴリズムの説明のために必要な基本的な datalog の概念や Why-not の概念について書く。

3.1 Datalog

datalog のルールは head と body で構成されている。また、body は一つ以上の原子 (atom) で構成される。head は body を構成するすべての原子が真実 (true) であった時、答として出力すべきデータを指定する。その時、答がないというのは body を構成する原子の間で互いに衝突があることを意味する。衝突というのは、互いに矛盾な状態を意味する (e.g. A>50 と A <30)。body を構成するのは EDB(Extensional database) や IDB(Intensional database) や条件文である。EDB は SQL 文の FROM に該当するもので、データベースを意味し、datalog ではいつも事実として認められるものである。IDB は幾つかの EDB や IDB や条件文で構成される relation である。条件文は SQL の WHERE に現れるものである (e.g. A>100, A=B)。理論的には body にある IDB が head にも現れることもあるが、

今回は再帰の問合せは扱っていないのでその可能性は排除する。また、本研究では body の中は IDB のみを含むか、EDB や条件文のみを含むかのいずれかとする。前者は集合和や集合差がある場合で、その IDB をそれぞれの原子として扱うことができる。後者は集合和や集合差がない問合せである。先のような制約を使うと datalog で表現できるすべての問合せを扱えないが、SPJUD に該当する SQL 問合せで扱う問合せはすべて扱える。次に、datalog の形式的な定義を [1] に基づいて与える。

Definition 1. Syntax of datalog
本論文で使う datalog のルールは以下のようなものである。
$R_1(u_1)$:- $R_2(u_2)$, ... , $R_n(u_n)$, C_1, C_2 , ... C_m.
$n \geqq 1$ に対して、R_1,R_2,...,R_n は関係名で、u_1,...u_n は適切な大きさの組を意味する。また、u_1 に含まれる変数は必ず $u_2, \ldots, u_n$ の中に含まれている。$m \geqq 0$ に対して、 C_1,C_2,...,C_m は条件文を意味する。条件文は u_2 から u_n のいずれかに現れる二つの変数を x,y とすると、x と y、x と定数で作られる比較文である。
R_1 は datlaog ルールの head で、$R_2, \ldots, R_n, C_1, \ldots, C_m$ は datalog のルールの body である。

datalog 問合せは、ルールの有限集合で表される。datalog の問合せが集合和と集合差を含まない場合は、問合せを一つのルールで表現できる。そのため、一つのルールにより構成される問合せは “conjunctive query” である。また、集合和と集合差がある場合は、最終結果を表す関係（本論文では result) を IDB のみで表現しそれぞれの IDB を集合和と集合差が含まれていない問合せと見なして問合せを作成する。

Definition 2. EDB, IDB, 条件文
Extensional relation は body の中にしか存在しない関係で、Intensional relation は head に存在する関係を意味する。EDB(Extensional database) はすべての Extensional relation の集合である。IDB(Intensional database) はすべての Intensional relation の集合である。条件文は、EDB でも IDB でもない、属性と属性あるいは属性と定数間の比較文である。

Example 1. datalog ルール
$result(A, B)$:- $a(A, B, C), b(D, E, F), A = D, C$>500, $F =$ “$park''$.
$result(A, B)$ は head であり、$a(A, B, C)$ と $b(D, E, F)$ は各々テーブル a の属性を順番に (A,B,C) とする、テーブル b の属性を順番に (D,E,F) とするということを意味し、A=D や C>500,F=“park ”は条件文である。ここで $a(A, B, C)$ と $b(D, E, F)$ は（データベースにある）テーブルなので EDB である。

Example 2. datalog 問合せ
$result(A, B)$:- $result2(A, B), \neg result3(A, B)$.
$result2(A, B)$:- $a(A, B, C), C =$ “CS ”.
$result3(A, B)$:- $a(A, B, C), C =$ “EE ”.
$result2(A, B)$ と $result3(A, B)$ は IDB であり、¬ は否定を意味する (SQL の集合差に対応する)

Definition 3. 集合和と集合差を含まない datalog 問合せの答
元の集合和と集合差を含まない datalog 問合せ q を

$$result(x_1, \ldots, x_n) \text{ :- } a_1, \ldots, a_m$$

とする。この時、すべての EDB や条件文である $a_l \in A(= \bigcup_{i=1}^{m} a_i)$ により $result$ は計算される。$t \in result(x_1, \ldots, x_n)$ に対して t のすべての集合 T を答、あるいは結果とする。

Definition 4. datalog の中での衝突
元の集合和と集合差を含まない datalog 問合せ q を

$$result(x_1, \ldots, x_n) \text{ :- } a_1, \ldots, a_m$$

とする。$result(x_1, \ldots, x_n)$:- $a_1, \ldots, a_m, a_x$ が解を持たないような新しい a_x があるとする。空集合ではない $C \subset A(= \bigcup_{i=1}^{m} a_i)$ に対して、$A_C = A \backslash C$ を定義する。$result(x_1, \ldots, x_n)$:- A_C, a_m が解を持つ時、C と a_m は互いに衝突している。

3.2 Why-not Question

Why-not Question は利用者の疑問である。利用者がデータベースへの問合せ結果を見て、自分の知識と間違っている結果があるとデータベースあるいは結果そのものに疑問を持つ。その疑問が Why-not Question である。Why-not Question は [属性 op 値] の集合から成る。属性は datalog 問合せの結果に表れている属性で、op は=,>,>=,<,<=のいずれかの比較演算子、値は演算子の対象になる変数や定数を意味する。例えば、[A>100] というのは「属性 A の値が 100 より大きいものがなぜないか」という Why-not Question を意味する。
この場合、属性は結果に含まれる属性のいずれかである。ここで重要な点は同じ EDB でも違う属性名を持つことができるということである。例えば、$result(A, B)$ には $a(A, B, C)$ があるが、$result2(C, D)$ には $a(C, D, E)$ が存在するかも知れない。しかし、本論文では同じ EDB には、同じ名前の属性名を使う。もし、ルールの中で同じ EDB が 2 回以上使われる場合は、“$a(A, B, C), a(D, E, F)$” のようにすべての属性名に違う名前を付け、同じ物理データを参照にする二つの EDB と見なす。
また、利用者の疑問である Why-not Question を既存の問合せに追加できるように問合せ形に変更したのが Why-not Predicate である、[A>100] という Why-not Question は datalog で使うために A>100 という Why-not Predicate になり、そのまま既存の問合せルールの body に追加することができる。この場合、Why-not Predicate が追加された新しい問合せはどんな場合でも結果は出て来ない。その理由は Why-not Predicate が既存の問合せと衝突しているからである。例えば、結果には “A>100” を満足するレコードがないのに、問合せに “A>100” を追加すると、元の結果から “A>100” で絞り込むことを意味するので結果は空集合になる。

Definition 5. Why-not Question
Why-not Question は利用者の疑問であり、Why-not Question を満足する答は結果に含まれていない。Why-not Question は [属性 op 値] の集合である。op に該当するものは =,!=,>,>=,<,<=である。値に該当するものは “属性”、“定数” である。

Definition 6. Why-not Predicate
Why-not Predicate は Why-not Question であるものを、datalog に条件として追加するための条件文の形である。
“ 属性 op 値, ... , 属性 op 値 ” である。

3.3 Why-not Answer

Why-not Answer は問合せの中で Why-not Question と衝突しているところである。すなわち、Why-not Answer を修正あるいは削除することで我々は Why-not Question に該当する答を結果から得ることができる。Why-not Answer を求めることが本論文の目標である。利用者あるいは管理者は Why-not Answer を見ることで問合せの問題か利用者の知識の問題かがわかるようになる。

Definition 7. 集合和と集合差を含まない問合せの Why-not Answer
元の集合和と集合差を含まない datalog 問合せ q を

$$result(x_1, \ldots, x_n) \text{ :- } a_1, \ldots, a_m$$

とする．x は属性を表し、a は EDB や条件文を表す。ここで一般性を失うことなく，このルールの body に現れる原子のうち $a_1, \ldots, a_l$ $(l \leqq m)$ を条件文とする．また，Why-not predicate w を

$$w_1, \ldots, w_k$$

とする．このとき，q に対する w の Why-not Answer は，$\{a_1, \ldots, a_l\}$ の部分集合で w と衝突する部分である．

Proposition 1. *(Why-not Answer の存在条件)*
Why-not Question を満足するレコードが問合せの答には含まれていなく問合せの対象になるデータベースには含まれている時は、必ず *Why-not Answer* が存在する。

Proof 1. Why-not Answer の存在
1. [属性 op 値] の集合として与えられた Why-not Question が与えられたとする。
2. Why-not Question から Why-not predicate を求めることができる。
3. 利用者が持っている知識が真実である限り、Why-not Question はデータベースに含まれている。
4. 元の集合和と集合差を含まない問合せは $result(x_1, \cdots, x_n)$:- $a_1, \cdots, a_n$ であり、x は属性を表し、a は EDB や条件文を表す。一般には a に IDB も含まれているが、集合和と集合差がない限りは IDB は存在しない。
5. 条件文がない EDB のみで構成される新しい問合せ $result2(x_1, \cdots, x_m)$:- $a_1, \cdots, a_l$ を定義する。$\forall i, a_i \in$ EDB
6. 3が成立する限り、$result2$ は Why-not Question に対して解を持つ。
7. この時、$result2$ に条件文を追加した $result3$ を定義する。$result3$ は $result2$ に既存の問合せである $result$ にある条件文を追加したものであり、Why-not Question に対して解を持つ問合せである。
8. $result3$ の中で一番、多くの条件文が追加されたものらを $result\_max$ と定義する。$result\_max$ は Why-not Question に対して解を持つ。
9. $result$ と $result\_max$ の差分を計算する。その差分になる条件文は Why-not Question が答にならなかった原因である。
10. この条件文の集合は、既存の Why-not Answer の定義と一致する。

証明 1 により、データベースに Why-not Question が含まれている限り Why-not Answer を求めることができるということが証明された。この証明を使うためにデータベースはいつも事実である（利用者の知識、あるいは問合せが間違っている）と仮定をする。

名前	専門	性別	年齢
James	EE	M	25
Alice	EE	W	35
Bob	CS	M	33

表 1 Table A

名前	専門	性別	年齢
James	EE	M	25

表 2 Table B

Example 3. Why-not Question と Why-not Answer
SQL Query
select * from a where 専門=‘EE ’ AND 年齢<34 AND 性別=‘M ’.
Datalog Query
$result(A, B, C, D)$:- $a(A, B, C, D), B = \text{‘}EE\text{’}, C = \text{‘}M\text{’}, D$<34.
Table A に上記の問合せを適用すると Table B のような結果が出る。この結果から利用者は次のような疑問を持つ。
“ どうして Bob は結果に含まれていないのか？”
これが利用者の Why-not Question である。
証明 1 の 5 によると、$result2$ は $result2(A, B, C, D)$:- $a(A, B, C, D)$ である。この $result2(A, B, C, D)$ によると、利用者が疑問としている Bob は $result2(A, B, C, D)$ に表れている。$result3$ の候補としては $result3(A, B, C, D)$:- $a(A, B, C, D), C = \text{‘}M\text{’}$、$result3(A, B, C, D)$:- $a(A, B, C, D), D$<34.、 $result3(A, B, C, D)$:- $a(A, B, C, D), C = \text{‘}M\text{’}, D$<34. がある。この三つの候補の中で一番条件が多く追加されているものは $result3(A, B, C, D)$:- $a(A, B, C, D), C = \text{‘}M\text{’}, D$<34 である。$result3$ と $result$ の差分は B=‘EE ’である。したがって、我々は B=‘EE ’が Bob を結果の中に含まらないようにした原因であることを分かった。すなわち、B=‘EE ’が Why-not Answer である。実際に我々は B=‘EE ’を削除するか、B=‘CS ’にすることで Bob を結果に出すことができる。

今までの定義は Negation 含まれていない問合せに対する定義であった。Negation が含まれていない問合せは単純に利用者の Why-not Question と衝突しているところを見つけることで Why-not Answer を求められた。しかし、我々の論文では Negation も扱っている。Negation がある場合の Why-not Answer は既存の定義とは異なるところがある。また、集合和の場合と同様に集合差が含まれている問合せは最終結果を表す関係 (result) が IDB のみで構成されることにする。これから、

Negation と Negation が含まれている問合せに対する Why-not Answer を定義する。

Definition 8. datalog の中の Negation
Negation は relation の前に not あるいは ¬ をつけて表現する。それがつけられている時の解釈は [1] の通りに以下のようである。
datalog の Stratified Semantics を使い、$\neg R(x)$ は "x はすべての可能なドメイン集合 (Active domain) に含まれ、x$\notin R$ を満足すること" を意味する。

Definition 9. 集合和と集合差を含む datalog 問合せの答
元の集合和と集合差を含む datalog 問合せ q を

$$result(x_1, \ldots, x_n) \text{ :- } I_1, \ldots, I_m$$

とする。また、I_k は

$$I_k(x_{k_1}, \ldots, x_{k_n}) \text{ :- } a_{k_1}, \ldots, a_{k_m}$$

とする。この時、すべての EDB や条件文である $a_{k_l} \in A(= \bigcup_{i=1}^{m} a_{k_i})$ により I_k は計算されて、それらにより result が計算される。$t \in result(x_1, \ldots, x_n)$ に対して t のすべての集合 T を答、あるいは結果とする。

Definition 10. Why-not Answer
元の集合和と集合差を含む datalog 問合せ q を

$$result(x_1, \ldots, x_n) \text{ :- } I_1, \ldots, I_m$$

とする。また、I_k は

$$I_k(x_{k_1}, \ldots, x_{k_n}) \text{ :- } a_{k_1}, \ldots, a_{k_m}$$

とする。a は EDB や条件文を表し、x は属性を表し、I は IDB を表す。その中で、¬ が付けられていない $P_1, \ldots, P_o \in I(= \bigcup_{i=1}^{m} I_i)$ $(o \leqq m)$ を肯定 IDB とする。また、¬ が付けられている $N_1, \ldots, N_k \in I(= \bigcup_{i=1}^{m} I_i)$ $(k<m)$ を否定 IDB とする。このとき、利用者の疑問である Why-not Question に対する答えが肯定 IDB の結果に含まれていないときは、定義 7 のように Why-not Answer の定義に従う。また、Why-not Question に対する答えが否定 IDB の結果に含まれているときは、否定 IDB の中にあるすべての条件文の組み合わせが Why-not Answer になる可能性を持つ。

3.4 集合和と集合差

集合和と集合差が問合せに含まれている場合でも証明 1 を応用することで証明できる。ここでは、集合和と集合差があった場合に、どう証明するかとそれが事実であることを示す。まず、集合和の場合は証明 1 を繰り返すことで証明できる。

Proof 2. 集合和の場合
1. 集合和が含まれている問合せを $result$ とする。$result(x_1, \cdots, x_n)$:- $I_1(x_1, \cdots, x_n), \cdots, I_n(x_1, \cdots, x_n).$, $\forall m, I_m(x_1, \cdots, x_n)$: $-a_1, \cdots, a_l$
2. 各々の $\forall m, I_m(x_1, \cdots, x_n)$ は、証明 1 に表れている問合せと同じ形である。したがって、それぞれの問合せにに対して証明 1 を適用することで証明できる。

集合和の場合は集合和で繋がっているいずれかの部分問合せも対等である。しかし、集合差の場合は違う。なぜかというと、集合差が含まれている問合せは部分問合せが互いに違う役割をするからである。例えば、A except B の時、A と B は Why-not Provenance を同じ方法で求めることができないし、それが従来のアルゴリズムの問題であった。今回、我々のアルゴリズムで使う方法は B が持つ特徴に起因する。A except B の問合せにより、利用者が期待する答が結果に含まれていない時は、A に答が含まれていないか、B に答が含まれているか、いずれかも成立するかである。A に答が含まれていない時は、今までの方法とおなじ方法で Why-not Answer を計算することができる。しかし、B に答が含まれているのは違う方法を使わないといけない。その理由を証明 3 で示す。

Proof 3. 集合差の場合
1. 集合差が含まれている問合せを result とする。$result(x_1, \cdots, x_n)$:- $I_1\ (x_1, \cdots, x_n), \neg\ I_2(x_1, \cdots, x_n)$. I が三つ以上の場合でも 2 と 3 を繰り返すことで求められる。
2. I_1 に答が含まれていない場合は証明 1 で証明できる。
3. I_2 に答が含まれている場合は、I_1 のように原因を特定することができない。なぜかというと、I_2 の中にあるすべての条件文が Why-not Question と衝突しないからである。これは、I_2 の中にあるすべての条件文が原因になる可能性があることを意味する。
4. したがって、I_2 で答が含まれている場合は、I_2 の中にあるすべての条件文が Why-not Provenance の原因になる。

そのゆえ、集合和と集合差のいずれかにも問題がある場合は、証明 1 と証明 3 で Why-not Answer を探すことができる。

4. アルゴリズム

我々のアルゴリズムはデータベースの上での計算を繰り返す。アルゴリズム 1 では Why-not Question が条件文の形である Why-not Predicate で問合せに追加され、元の問合せの条件の一部が削除されたものが解を持っているかどうか確認する。集合和と集合差がない問合せはアルゴリズム 1 で計算できる。combination 関数は A の集合から i 個の combination を計算する関数である。もし、1 個の場合で解が存在したら、2 個の場合は計算せずに終わる。
アルゴリズム 2 は問合せがあった時、それを部分問合せに分解するためのアルゴリズムである。自然結合で繋がっている条件文と EDB(SQL 文の FROM のところ) で構成される小さい問合せを作ってそれを部分問合せとしてする。このアルゴリズムを使うことで、直積集合 (Cartesian product) を早く計算できるようになる。大量のデータを使う時は、自然結合がないとデータベースでの計算が遅くなる。それを防ぐためには自然結合で繋がっていないものを独立的に計算する必要性がある。
集合和と集合差が含まれている場合はアルゴリズム 3 を使う。アルゴリズム 3 ではまず原因になる IDB を探す。not が付いていない部分問合せは Why-not Question を追加した問合せが答を出さない時、原因の可能性が有る。逆に、not が付いている問合せは Why-not Question を追加した問合せが答を出す時、原因の可能性が有る。アルゴリズム 3 により、原因の可能性が

あると判断された IDB はアルゴリズム 1 を使って細かい条件文を探す。not が付いていない問合せはアルゴリズム 1 により特定できるが、not が付いている場合は、アルゴリズム 1 による特定ができない。したがって、証明 3 により not がついている問合せが原因になった時は、その問合せのすべての条件文を Why-not Answer とする。

result_true は Why-not Answer の可能性がない条件文である。これを使う理由は、集合和と集合差がある場合、すべての部分問合せが満足する Why-not Answer を保証するためである。したがって、最後の結果は Why-not Answer から result_true を除いた条件文の集合になる。

Algorithm 1

①は、アルゴリズム 3 の 9 行目と 16 行目に該当する。②は, アルゴリズム 3 の 12 行目に該当する。
input : why-not Question W_q, Query Q
output : Why-not Answer W_a
A = Q の条件文の集合, F = Q の EDB
for $i = 1$ to len(A) **do**
    ConArr = combination(A,i)
    for $j = 1$ to len(ConArr) **do**
        A_1 = A - ConArr[j], result = [] ,result_true = []
        $W_p = W_q$ の Why-not Predicate. $A_1 = A_1 + W_p$
        SubQueries = アルゴリズム 2(A_1,F)
        for $j = 1$ to len(SubQueries) **do**
            if SubQueries[i] **then**
                condition = True
            else
                condition = False
                break
            end if
        end for
        if condition **then**
            ① result =result + subset
            ② result_true = result_true + ConArr[j]
        else
            ① result_true = result_true + ConArr[j]
        end if
    end for
    ② break
    if result **then**
        break
    end if
end for

4.1 Why と Why-not

この節では Why と Why-not 質問が同時にあった時、対応できることを示す。Why は答に対して “どうして答が出たか ” について理由を説明する。Why-not は疑問に対して “どうして疑問に該当する答が出てこないか ”について理由を説明する。すなわち、互いに逆のことを説明している。datalog を使うと negation を意味する ‘not’ を使って集合差を表現することができる。アルゴリズム 1 では Why-not Predicate を ‘not’ をつけずそのまま使っている。しかし、Why の質問はそのまま使わず、‘not’ を付けて Why-not Predicate を表現する。例えば、Why and Why-not predicate “A=‘Bob ’, not A = ‘Alice ’”が意味するのは Alice は結果に含まれているが Bob は結果に含まれていない理由を出力するために使われる。もし、原因になる部分が ‘not’ が付いていない問合せであれば Why-not predicate と同様に計算できる。。しかし、‘not’ が付いている部分問合せが原因であれば、すべての条件文が原因ではなくアルゴリズム 3 の②で原因になる条件文を探す。

Algorithm 2

input : 条件文 A, EDB F
output : subQueries Q_s
if F が全て natural join で繋がっている **then**
    Q = A と F により構成される SQL 文
    $Q_s = Q_s$ + Q
    return Q_s
else if F の一部が natural join で繋がっている **then**
    for i=1 to len(A) **do**
        if A[i] が natural join で繋がっている F の一部に含まれる **then**
            A = A + A[i]
        end if
        Q = A と F の一部により構成される SQL 文
        $Q_s = Q_s$ + Q
        A = []
        for i=1 to len(F) **do**
            if F[i] が natural join で繋がっていない **then**
                for j=1 to len(A) **do**
                    if A[j] が F[i] の演算である **then**
                        A = A + A[j]
                    end if
                end for
                Q = F[i] と A により構成される SQL 文
                $Q_s = Q_s$ + Q
            end if
        end for
    end for
    return Q_s
else if 全ての EDB が natural join で繋がっていない **then**
    for i=1 to len(F) **do**
        A = []
        for j=1 to len(A) **do**
            if A[j] が F[i] の演算 **then**
                A = A + A[j]
            end if
        end for
        Q = F[i] と A で構成される SQL 文
        $Q_s = Q_s$ + Q
    end for
    return Q_s
end if

Algorithm 3

1: W_q が Why and Why-not Question の場合は①, why-not Question の場合は②
2: input : why-not Question W_q, Query Q
3: output : 原因である IDB 集合, Why-not Answer W_a
4: I = Q の IDB の集合
5: **for** i=1 to len(I) **do**
6: $W_p = W_q$ の Why-not Predicate, result = [],result_true = []
7: **if** I[i] に'not' が付いている **then**
8: **if** I[i] + W_p が結果を出す **then**
9: ① アルゴリズム 1(I[i], not W_q)
10: ② $W_a = W_a$ + I[i] のすべての条件文
11: **else**
12: アルゴリズム 1(l[i], W_q)
13: **end if**
14: **else if** I[i] に'not' が付いていない **then**
15: **if** I[i] + W_p が結果を出さない **then**
16: アルゴリズム 1(I[i], W_q)
17: **else**
18: result_true = result_true + I[i] のすべての条件文
19: **end if**
20: **end if**
21: **end for**
22: W_a =W_a - result_true

5. 実　　験

本章では 4 章で表れているアルゴリズムの実験評価を行う。実験で使うデータとして、"http://pgfoundry.org/projects/dbsamples/" に公開されている "World" と "dellstore2" を使う。それぞれの容量は 81KB と 2.16MB であり、現実の状況を反映している。また、仮想のデータ (TPC-H) を容量ごとに用意し、1MB から 10GB までのデータへの実験を行った。実験で使われる計算機ーの仕様は MAC OS X10.10.5, 8GB RAM, 1.7 GHz Intel Core i7 でデータベースは PostgreSQL を使う。
実験の手順は "1. 問合せを作る.2. 結果を見て Why-not Question をする。3. アルゴリズムに入力し Why-not Answer を求める" である。まず、表 4 に示す問合せに対してシナリオを設定し実験を行う。それぞれのシナリオは Why-not Question と問合せが実行されるデータベースが与えられる。実験で使うシナリオは表 5 のようである。

それぞれのシナリオに対する実験結果を図 2 と図 3 に与える。図 2 は World1、World2、World3、dellstore1 の実行時間である。World1、World2、World3 は同じデータベースに対して異なる問合せと異なる Why-not Question であるが、実行時間はあまり変わらないことが分かる。dellstore1 は World データに比べて容量が大きいデータベースを対象にしている。したがって、実行時間も容量に比例し増加することを確認できる。仮想 1 と仮想 2 は [4] のアルゴリズムとの時間比較のためのシナリオである。同じ実験環境ではないので、正確な比較は不可能であるが、仮想 2 の質問に対しては 10GB で 10 倍以上早く計算している。また、[4] は仮想 1 (simple query) が仮想 2(complex query) より早く計算されるが、我々のアルゴリズムは逆である。その理由は我々のアルゴリズムはデータベースを使って計算しているからである。データベースはデータベースの中にある最適機により自然結合などの演算を処理する前に処理されるデータを絞り込むことが出来るなら、先に絞り込んでから演算を行う。したがって、相対的に条件文が多い complex query の

表 3　SQL 問合せ

問合せ	表現
Q1	select * from city as c,country as co,countrylanguage as cl where c.countrycode = co.code and cl.countrycode = co.code and c.population >10000000
Q2	select * from city as c,country as co where c.countrycode = co.code and c.population>9000000 **except** select * from city as c,country as co where c.countrycode = co.code and c.population<90000
Q3	select * from city as c,country as co where c.countrycode = co.code and c.population>9000000 **except** select * from city as c,country as co where c.countrycode = co.code and c.population<90000 and c.country='China'
Q4	select * from orders as o,customers as c,orderlines as ol,products as p where o.customerid = c.customerid and ol.orderid = o.orderid and p.prod_id = ol.prod_id and c.income >= 100000 and c.country = 'China' and ol.quantity>4 **union** select * from orders as o,customers as c,orderlines as ol,products as p where o.customerid = c.customerid and ol.orderid = o.orderid and p.prod_id = ol.prod_id and o.totalamount >432 and p<10
Q5	select * from customer as c,orders as o,lineitem as l where o.orderdate<'1998-07-21' and l.shipdate>'1998-07-21' and c.custkey=o.custkey and o.orderkey=l.orderkey

表 4　シナリオ

シナリオ	問合せ	Why-not Question
World1	Q1	c.name='China'
World2	Q2	c.name='China'
World3	Q3	co.name='China' and co.name!='South Korea'(Why and Why-not)
dellstore1	Q4	c.country='China'
仮想 1	Q5	o.orderdate<'1996-01-01' and l.extendedprice >50000
仮想 2	Q5	l.extendedprice>100000 and o.orderdate = l.commitdate and c.nationkey=4

表 5　Why-not Answer

シナリオ	Why-not Answer
World1	'c.countrycode = co.code', 'c.population>10000000'
World2	'c.population<90000'
World3	('c.countrycode = co.code', 'co.name = 'China"), ('c.population<90000', 'co.name = 'China')
dellstore1	'ol.quantity>4' , 'o.totalamount >432', 'p.price<10'
仮想 1	'l.shipdate>'1998-07-21', 'o.orderkey=l.orderkey'
仮想 2	'o.orderkey=l.orderkey'

方が simple query の方より早く処理される。また、仮想データの場合は実験ではインデックスを使っていないがインデックスを使うことで、より早く Why-not Answer を計算できると期待している。最後に、今回の実験では集合和と集合差の両方を使う問合せは対象にしていないが、その場合でもアルゴリズム 1 をもう一回使うだけで Why-not Answer を求めることが出来る。

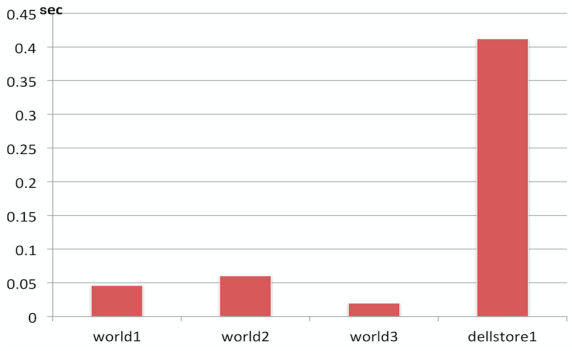

図 2　実データのアルゴリズムの処理時間

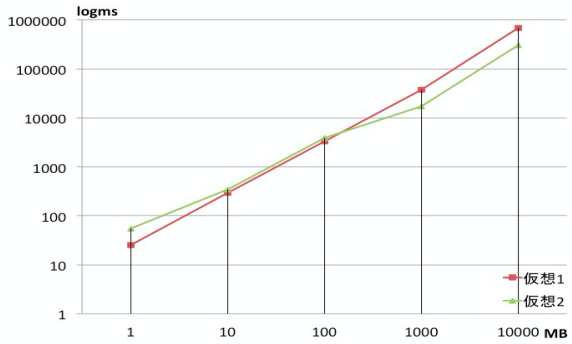

図 3　仮想データのアルゴリズムの処理時間

6. 結　　論

本論文では集合差を含む問合せに対応する Why-not アルゴリズムを提案した。Why-not 技術は利用者の疑問を入力として、疑問を解決するための解決案を提案するものである。Why-not 技術を利用することで、利用者はデータベースへの信頼度を高められる。また、管理者はディバギングやシステムメインテナンスが楽になる。本論文で提案されたアルゴリズムはデータベースのでの計算を繰り返すことで Why-not Answer を計算する。その結果、疑問が詳しいければ詳しいほど実行時間が短くなることも分かった。しかし、疑問の原因になる条件文を見つけるだけではなく、疑問を解決するための問合せやデータベースの修正は今後の課題である。

文　　献

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Sourav S. Bhowmick, Aixin Sun, and Ba Quan Truong. Why not, wine?: Towards answering why-not questions in social image search. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13, pages 917–926, New York, NY, USA, 2013. ACM.

[3] Nicole Bidoit, Melanie Herschel, and Katerina Tzompanaki. Query-based why-not provenance with nedexplain. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014.*, pages 145–156, 2014.

[4] Nicole Bidoit, Melanie Herschel, and Katerina Tzompanaki. Efficiently and Effectively Answering Why-Not Questions based on Provenance Polynomials. Research Report RR-8697, OAK team, Inria Saclay ; INRIA, March 2015.

[5] Adriane Chapman and H. V. Jagadish. Why not? In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 523–534, New York, NY, USA, 2009. ACM.

[6] Lei Chen, Xin Lin, Haibo Hu, Christian S Jensen, and Jianliang Xu. Answering why-not questions on spatial keyword top-k queries. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 279–290. IEEE, 2015.

[7] Yunjun Gao, Qing Liu, Gang Chen, Baihua Zheng, and Linlin Zhou. Answering why-not questions on reverse top-k queries. *Proc. VLDB Endow.*, 8(7):738–749, February 2015.

[8] Zhian He and Eric Lo. Answering why-not questions on top-k queries. *Knowledge and Data Engineering, IEEE Transactions on*, 26(6):1300–1315, 2014.

[9] Melanie Herschel. Wondering why data are missing from query results?: ask conseil why-not. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, CIKM '13, pages 2213–2218, New York, NY, USA, 2013. ACM.

[10] Melanie Herschel and Mauricio A. Hernández. Explaining missing answers to spjua queries. *Proc. VLDB Endow.*, 3(1-2):185–196, September 2010.

[11] Melanie Herschel, Mauricio A. Hernández, and Wang-Chiew Tan. Artemis: A system for analyzing missing answers. *Proc. VLDB Endow.*, 2(2):1550–1553, August 2009.

[12] Jiansheng Huang, Ting Chen, AnHai Doan, and Jeffrey F. Naughton. On the provenance of non-answers to queries over extracted data. *Proc. VLDB Endow.*, 1(1):736–747, August 2008.

[13] Brian Y. Lim, Anind K. Dey, and Daniel Avrahami. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 2119–2128, New York, NY, USA, 2009. ACM.

[14] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. The complexity of causality and responsibility for query answers and non-answers. *Proc. VLDB Endow.*, 4(1):34–45, October 2010.

[15] Balder ten Cate, Cristina Civili, Evgeny Sherkhonov, and Wang-Chiew Tan. High-level why-not explanations using ontologies. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems*, PODS '15, pages 31–43, New York, NY, USA, 2015. ACM.

[16] Quoc Trung Tran and Chee-Yong Chan. How to conquer why-not questions. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 15–26, New York, NY, USA, 2010. ACM.

[17] Rui Zhou, Chengfei Liu, and Md. Saiful Islam. On answering why-not questions in reverse skyline queries. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ICDE '13, pages 973–984, Washington, DC, USA, 2013. IEEE Computer Society.