DEIM2021 B21-1

複合的データ解析を伴う分析処理に対するトレーサビリティの研究

山田　真也†　　北川　博之††　　天笠　俊之††

† 筑波大学院 理工情報生命学術院 システム情報工学研究群　〒 305-8577 茨城県つくば市天王台１丁目１－１
†† 筑波大学 計算科学研究センター　〒 305–8577 茨城県つくば市天王台１丁目１－１
E-mail: †yamada@kde.cs.tsukuba.ac.jp, ††{kitagawa,amagasa}@cs.tsukuba.ac.jp

あらまし　データ分析は意思決定のような重要度の高い応用先で利用されており，分析結果への信頼性を高めるためには分析に対するトレーサビリティが重要である．Data Lineage は分析結果データがどの入力データを元に作られたのかを示す情報のことであり，分析のトレーサビリティを保証するために重要な技術になっている．近年の分析処理は，分析対象が画像や動画，テキストといったコンテンツデータにまで広がったことや，処理にデータ抽出や機械学習，AI を用いた複雑な解析が含まれるようになってきている．そのような複合的データ解析を含む分析処理のトレーサビリティとしては，もとになったデータを提示する Lineage だけではなく，複合的データ解析でどのような判断根拠に基づいて分析結果を生成したかまで追跡する必要があるが，そのような検討を十分に行っている研究はこれまでにない．そこで本稿ではリレーショナルモデルのもとで User Defined Function（UDF）を利用して複合的データ解析をモデル化し，複合的データ解析における判断根拠を考慮した Lineage（Augmented Lineage）の定義と導出方法を提案する．さらに実験で提案する方法のパフォーマンスを評価する．

キーワード　トレーサビリティ，Data Lineage，複合的データ解析，User Defined Function，AI 処理

1 はじめに

Data Lineage は分析処理の結果データがどの入力データから導出されたのかを示す情報のことであり，分析処理に対するトレーサビリティを保証するために重要な技術になっている．Data Lineage を利用することで分析処理を実行した分析者は，出力された分析結果データのもとになったデータを知ることができ，それによって分析結果がどうして導出されたのかを理解することができるようになる．

近年ではビッグデータの普及に伴い，複合的データ解析を伴う分析処理が重要になっている．ここでいう複合的データ解析とは，画像や動画，テキストのようなコンテンツデータを対象とする処理や，データ抽出や機械学習，AI を用いた複雑な処理を組み合わせた処理のことを指し，それらの解析を利用することで分析対象の拡大や，より高度な知識をデータから発見することが可能になる．そのため，近年社会で行われる分析処理の多くには複合的データ解析が伴うようになってきており，分析結果をより重要度の高い意思決定に使用することが広く行われている．

Data Lineage については以前から，分析処理を関係代数のような単純なオペレータを用いてモデル化し，モデル化した処理に対して Lineage を導出するという研究が多く行われている．ただし近年行われているような複合的データ解析は，関係代数だけではモデル化できないことが普通である．そのようなことから，より複雑な分析処理に対する Lineage の導出方法を示した研究も存在する．

しかしながら複合的データ解析を伴う分析処理の特性を考えると，単に Lineage を提示するだけではそのような分析処理に対するトレーサビリティとしては不十分である．そのことを説明するために，以下のような例を考える．

例 1. 金融機関が大量のローンの申請に対して機械学習の学習モデルを利用して，それぞれの申請を審査する分析を行う．分析をリレーショナルモデルに基づいてモデル化したものを図 *1* に示す．このとき，分析結果の ‘土浦次郎’ のローン申し込みだけがどうして不可と判断されたのかを知るためには，そのデータの *Lineage* を利用することが考えられる．この例で分析結果の ‘土浦次郎’ のデータの *Lineage* は，図 *1* で示した入力データのうち $\{C_{42}, D_{42}, L_4\}$ である．しかしながら，*Lineage* を使って ‘土浦次郎’ の申請結果が $\{C_{42}, D_{42}, L_4\}$ のデータが入力になっていることがわかっても，どうしてその入力データから学習モデルが不可と判定したのかがわからないため，“土浦次郎のローン申し込みだけがどうして不可と判断されたのか” は分からないままである．

例 1 に示したように，複合的データ解析を伴う分析処理に対しては，複合的データ解析は解析内部のロジックが非常に複雑であるため，単に入力データを提示するだけではどうして分析結果が導出されたのかが理解できないという問題が起こりうる．

その問題に対して本研究では，複合的データ解析を伴う分析処理に対してはもとになった入力データ（Lineage）だけでなく，複合的データ解析単体の判断根拠もあわせて提示するというアプローチで，分析処理に対するトレーサビリティを保証するフレームワークを提案する．つまり本研究では，例 1 の分析の ‘土浦次郎’ の分析結果データのトレーサビリティとして，そのデータの Lineage だけでなく “学習モデルは，土浦次郎には借入金が多くあり，さらにローンの申請金額も大きいから不可の判定を行った” というような複合的データ解析における判断

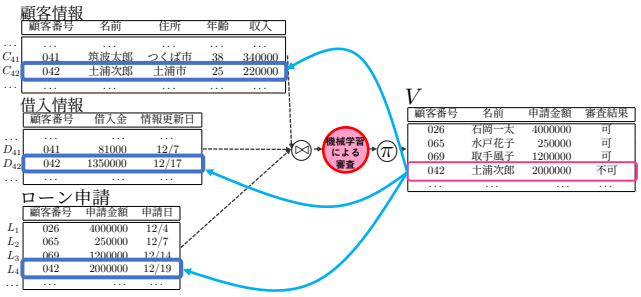

図 1: それぞれのローン申請に対して学習モデルを利用して審査を行う分析処理．テーブル 顧客情報 には申込者の基本データ，テーブル 借入情報 には申込者の借入金に関する情報，テーブル ローン申請 には申請されたローンの内容が記録されている．この図では 3 つのテーブルを入力データとして最終的にテーブル V が分析結果として導出されることを意味している．

根拠の情報も一緒に提示することを行う．そうすることで分析者は，複合的データ解析を伴う分析処理の分析結果データがどうして導出されたのかを理解することができるようになる．

本研究は，複合的データ解析をモデル化して判断根拠まで含めたトレーサビリティに関する初めての研究である．本稿における分析処理のモデル化はリレーショナルモデルに基づいて行う．複合的データ解析は，User Defined Function（UDF）を実行する新たなオペレータを定義することで複合的データ解析を伴う分析処理のモデル化を行う．本稿では，複合的データ解析を伴う分析処理に対する判断根拠を含めたトレーサビリティを提示するための Augmented Lineage とその導出方法，さらに効率的な導出方法について報告を行う．

2 関 連 研 究

本研究は，複合的データ解析を伴う分析処理に対するより適切なトレーサビリティを実現することを目的としている．その核となるのは分析結果のもとになった入力データを提示する Data Lineage である．Data Lineage に関してはデータベースの分野で多くの研究が行われており [2], [3], [7]，本節では特に，関係代数に基づいて分析処理をモデル化し Data Lineage を導出する研究について述べる．

単純なオペレータで表現可能な分析処理に対する Lineage: 従来の分析処理は，入力データに対してある条件を満たすデータを取り出したり，入力データ同士を結合したり，入力データをグループ化しそれぞれのグループに対して集約関数を適用して平均や総和を求めたりするなど，比較的単純な処理の組み合わせであることがほとんどだった．それらの処理は関係代数のような比較的単純なオペレータを組み合わせることで記述することができるため，分析処理を関係代数でモデル化し，モデル化された分析処理に対して Lineage を求める研究が広く行われてきている．[4] はリレーショナルモデルに基づいており，分析処理の出力タプルの属性の値と一致する入力タプルを探すことで Lineage を導出する研究である．また [1], [5], [6] は，はじめに全ての入力データに対してユニークな識別子を割り当てておいて，分析処理を実行する時にその識別子もオペレータの出力と一緒に適切に出力することで Lineage を提示する方法を示している．さらに別のアプローチとして [10] は，分析処理の実行時にそれぞれのオペレータにおける入力データと出力データの対応関係を記録しておき，Lineage が必要になった時に各オペレータで記録されている入出力データの対応関係を使うことで分析処理の Lineage を提示する方法を示している．しかしながらこれらの研究では，近年しばしば行われる複合的データ解析を伴う分析処理のモデル化が困難である．

より複雑な分析処理に対する Lineage: [11] は，より複雑な分析処理に対する Lineage の計算を可能にしている．この研究における Lineage の計算方法の基本的な方針は，前述した [10] と同様に，分析処理の実行時にそれぞれのオペレータにおける入出力データの対応関係を記録しておき，後からその対応関係を使って Lineage を計算するというものである．この研究では，ユーザが定義した関数である User Defined Function（UDF）を枠組みの中に取り入れることで複雑な分析処理のモデル化を可能にし，UDF の設計者がオペレータの入力データと出力データの対応関係を事前に定義しておくことによって，より複雑な分析処理に対する Lineage の計算方法を示している．

しかし，上記のいずれの研究についても複合的データ解析を伴う分析処理に対する判断根拠まで含めたトレーサビリティについて議論は行われていない．

3 提案する枠組み

本節で，本研究が提案する枠組みについて説明を行う．本研究は，セットリレーショナルモデルに基づいて分析処理をモデル化する．また，各入力データには Tuple Identifier（TID）が割り当てられている．

以降の説明で用いる表記を説明する．テーブル $R(A_1, \ldots, A_k)$ は，タプルの集合 $\{t_1, \ldots, t_n\}$ を持ち，$t_i.\boldsymbol{A}$ は，タプル t_i が持つ属性のうち属性集合 $\boldsymbol{A}$ ($\subseteq \{A_1, \ldots, A_k\}$) に含まれる属性だけを残し他の属性を削除することを意味する．$\langle t_1, \ldots, t_n \rangle$ はタプル $t_1, \ldots, t_n$ を連結したタプルを表す．データベース D はベーステーブルから構成される．ベーステーブルの集合 $\{R_1, \ldots, R_m\}$[1] に対するクエリを Q，そのクエリの出力を V と表記する．すなわち，$V = Q(R_1, \ldots, R_m)$ である．

3.1 Reason

本研究のリレーショナルモデルを用いた分析処理のモデル化では，複合的データ解析における判断根拠のことを Reason と呼ぶ．Reason は，分析処理に含まれる複合的データ解析でどうしてその処理結果を出力したのかを説明する情報のことである．また本研究で Reason は，複合的データ解析処理の設計者が適切に決定したものである．

例えば，例 1 のような顧客の情報を利用してその顧客のローン申請を通すか通さないかの分析を行うことを考える．そのとき学習モデルに入力されたデータのうち，学習モデルが特にど

1：同じリレーションを複数回参照するときは，それぞれ別のテーブルを参照していると考える e.g., $R \bowtie R = R_1 \bowtie R_2$.

のデータに着目してそれぞれの審査結果を決定したのかというような，学習モデルの判断根拠にあたる情報[2]を得ることができればこの分析での複合的データ解析における判断根拠が提示されたことになるため，そのような情報が Reason として取り扱われることになる．

3.2 分析処理のモデル化に使用するオペレータ

ここで複合的データ解析を伴う分析処理のモデル化に使用するオペレータを示す．分析処理のモデル化には関係代数の 6 つのオペレータ（Projection，Selection，Join，Aggregate，Union，Difference）と，複合的データ解析をモデル化するために User Defined Function（UDF）を実行する Function オペレータのあわせて 7 つのオペレータを用いる．ここで Function オペレータとは以下のように定義したものを指す．

定義 1 (Function オペレータ)**.** *Function* オペレータは，*UDF* に入力タプルの属性 $\boldsymbol{E}$ を入力したときに出力された値を新たな属性として付与して出力するオペレータである．*Function* オペレータで実行する *UDF* には以下に示すような *2* つのモードがある．

Normal モード: $udf\text{-}n\colon Domain(\boldsymbol{E}) \rightarrow Value$

Reasoning モード: $udf\text{-}r\colon Domain(\boldsymbol{E}) \rightarrow Value \times Reason$

UDF を Reasoning モードで実行した場合，複合的データ解析の出力とその判断根拠を得ることができる．しかし一般に Reason を求めるためには複合的データ解析の処理とは別の演算を行う必要があるため，必要なとき以外は複合的データ解析の出力だけを計算してほしい．そこで UDF には複合的データ解析の出力のみを導出する Normal モードが必要になる．また，この 2 つのモードは複合的データ解析を実行したい分析者自身が設計，実装する必要がある．その上で Function オペレータ $\phi_{udf(\boldsymbol{E})}$ は，実行する UDF のモードに応じて以下のどちらかを返す．

Normal モード: $\phi_{udf\text{-}n(\boldsymbol{E})}(R) = \{\langle t, udf\text{-}n(t.\boldsymbol{E}).Value\rangle \mid t \in R\}$

Reasoning モード:

$$\phi_{udf\text{-}r(\boldsymbol{E})}(R) = \{\langle t, udf\text{-}r(t.\boldsymbol{E}).Value, udf\text{-}r(t.\boldsymbol{E}).Reason\rangle \mid t \in R\}$$

例 2. 本研究のデータモデルを用いると図 *1* の分析処理は図 *2* のようにモデル化することができる．この例でローン審査を行う処理は *Function* オペレータ ϕ でモデル化されている．またこのとき *Function* オペレータの出力は，表 *1* のようになる．

3.3 Augmented Lineage

複合的データ解析を伴う分析処理に対する Lineage は，例 1 で示したように，もとになった入力データを見つけるだけでは不十分である．そのため，複合的データ解析の判断根拠を適切に提示することができる Lineage を考える必要がある．本研究ではそのような Lineage を Augmented Lineage と呼ぶ．はじめ

2：例えば例 1 の ‘土浦次郎’ のデータでいえば借入金と申請金額がそのような判断根拠にあたる

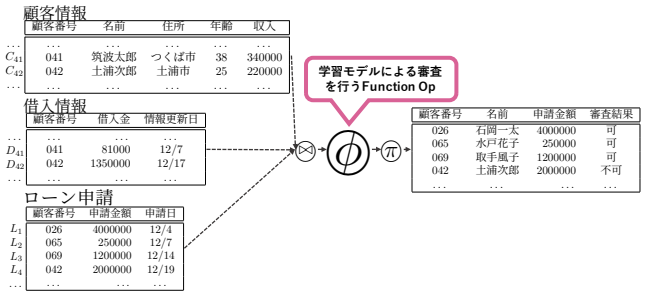

図 2: 図 1 の分析処理を Function オペレータを用いてモデル化したもの．学習モデルを用いてローン審査を行うのは Function オペレータで実行する UDF である．

表 1: Function オペレータの出力例．以下のテーブルは図 2 の Function オペレータを実行したときに出力されるものである．ここでは UDF は Reasoning モードで実行しているため，Function オペレータの出力にはローンの審査結果だけでなく，その審査結果の判断根拠として根拠にした特徴量も出力されていることに注意が必要である．

顧客番号	…	収入	借入金	申請金額	審査結果	根拠にした特徴量
…	…	…	…	…	…	…
042	…	220000	1350000	2000000	不可	借入金，申請金額
…	…	…	…	…	…	…

に Augmented Lineage を定義する準備として，分析結果のタプル t の Lineage を定義する．

定義 2 (Lineage)**.** ベーステーブル集合 $\{R_1, \ldots, R_m\}$ に対してクエリ Q を実行した結果を $V = Q(R_1, \cdots, R_m)$ とする．このとき，タプル $t \in V$ の *Lineage* $Lin(t)$ とは，以下の *3* つの条件を満たす各ベーステーブル $R_i (1 \leqq i \leqq m)$ の部分集合 R_i^* $(\subseteq R_i)$ に含まれるタプルの *TID* の集合である．

（*1*） $Q(R_1^*, \cdots, R_m^*) = \{t\}$
（*2*） $\forall t^* \in R_i^* : Q(R_1^*, \ldots, \{t\}, \ldots, R_m^*) \neq \emptyset$
（*3*） R_i^* は，*(1) (2)* を満たす R_i の最も大きな部分集合である．

また，タプル集合 $T \subseteq V$ の *Lineage* は

$$Lin(T) = \bigcup_{t \in T} Lin(t)$$

である．

また，Augmented Lineage として適切な判断根拠の提示を行うためには Function オペレータのどの出力タプルが分析結果のもとになったかの情報が必要である．Function オペレータが分析処理のどの部分に現れるかは分析処理に依存し，当然 Function オペレータが分析処理の途中に現れることもある．そこで，Augmented Lineage を定義する準備として，任意のオペレータの出力である中間結果 V' における分析結果のタプル t の Lineage Tuple $LT(t, V')$ を定義する．

定義 3 (Lineage Tuple)**.** ベーステーブル集合 $\{R_1, \ldots, R_m\}$ に対してクエリ Q を実行した結果を $V = Q(R_1, \cdots, R_m)$ とする．クエリ結果 V がクエリ Q', Q'' を用いて $V =$

$$[\{\ \begin{matrix} C_{42} \\ D_{42} \\ L_4 \end{matrix}\ \},\ \langle\ \langle\ 220000,\ 1350000,\ 2000000\ \rangle,\ \begin{matrix} \\ \text{不可, “借入金, 申請金額”} \end{matrix}\ \rangle\ ]$$

図 3: 例 1 の土浦次郎のデータの Augmented Lineage．はじめの要素が Base Table Lineage を表しており，次の要素が Reasoning Lineage を表している．

$Q'(Q''(R_{l_1}, \cdots, R_{l_k}), R_{l_{k+1}}, \cdots, R_{l_m})$ であるとき，中間結果 $V'(= Q''(R_{l_1}, \cdots, R_{l_k}))$ におけるタプル $t \in V$ の *Lineage Tuple* $LT(t, V')$ とは，以下の *3* つの条件を満たす V' の部分集合 $V'^*(\subseteq V')$ のことである．

（*1*） $Q(V'^*, R_{l_{k+1}}, \cdots, R_{l_m}) = \{t\}$
（*2*） $\forall t^* \in V'^* : Q(\{t^*\}, R_{l_{k+1}}, \cdots, R_{l_m}) \neq \emptyset$
（*3*） V'^* は，*(1) (2)* を満たす V' の最も大きな部分集合である．

また，中間結果 V' におけるタプル集合 $T \subseteq V$ の *Lineage Tuple* は

$$LT(T, V') = \bigcup_{t \in T} LT(t, V')$$

である．

次に定義 2，3 を利用して Augmented Lineage を定義する．

定義 4 (Augmented Lineage)**.** タプル集合 $T \subseteq Q(D)$ の Augmented Lineage $AL(T)$ とは以下のペア $(BTL(T), RL(T))$ を指す．ただし，クエリ Q に含まれる Function オペレータ $\phi_i(= \phi_{udf_i(\boldsymbol{E_i})})$ の出力テーブルを V_i' と表記する．

- Base Table Lineage（BTL）

$$BTL(T) = \{tid \mid tid \in Lin(T)\}$$

- Reasoning Lineage（RL）

$$RL(T) = \{\langle v.\boldsymbol{E}, v.Value, v.Reason\rangle \mid v \in LT(T, V_i')\}$$

例 3. この定義から，例 *1* の土浦次郎のデータの *Augmented Lineage* は，図 *3* に示したものになる．

3.4 Augmented Lineage の導出

本研究で提案する Augmented Lineage の導出方法は [4] を拡張したものである．本研究では Augmented Lineage を求めるために，Tracing Query を使用する．Tracing Query は，1）Lineage 導出の対象となるタプルと 2）分析処理の条件，そして 3）分析処理の入力になった全てのタプルの 3 つを入力すると，Lineage 導出の対象となるタプルの属性値と一致する入力タプルを探すことで Lineage を導出するクエリのことであり，関係代数を用いて記述することができる．Tracing Query の例を下に示す．

例 4. 金融機関が *25* 歳以下の顧客がどれくらいの借入金があるのかを調べる分析を考え，その分析処理をリレーショナルモデルでモデル化したものを図 *4* に示す．この例では分析結果の ‘土浦次郎’ のタプル $t(= \langle 042, \cdots, 1350000\rangle)$ の *Lineage* を求めることにする．*Tracing Query* に前述した *3* つの入力を与える

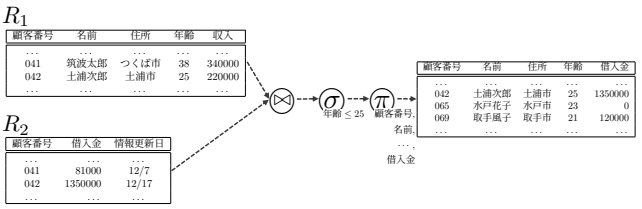

図 4: 25 歳以下の顧客がどれくらいの借入金があるのかを調べる分析処理．テーブル R_1 には顧客の基本データ，テーブル R_2 には顧客の借入金に関する情報が記録されている．

と，R_1 と R_2 における ‘土浦次郎’ のタプルの *Lineage* を求める *Tracing Query* は

$$V = \sigma_{\text{顧客番号}=t.\text{顧客番号} \wedge \cdots \wedge \text{借入金}=t.\text{借入金} \wedge \text{年齢} \leqq 25}(R_1 \bowtie R_2) \tag{1}$$

とすると，それぞれ

$$R_1^* = \pi_{\text{顧客番号, 名前,..., 収入}}(V)$$

$$R_2^* = \pi_{\text{顧客番号, 借入金, 情報更新日}}(V)$$

となる．ここで，R_1^* と R_2^* はそれぞれ R_1 と R_2 のテーブルにおける *Lineage* を表している．*Tracing Query* のポイントは式（*1*）の 顧客番号 $= t.$顧客番号 $\wedge \cdots \wedge$ 借入金 $= t.$借入金 という部分で，ここが *Lineage* 導出の対象となるタプル t の属性値と一致する入力タプルを検索し，もとになった入力データを探している部分である．上記の *Tracing Query* を実行することで $R_1^* = \{\langle 042, \cdots, 220000\rangle\}$ と $R_2^* = \{\langle 042, 1350000, 12/17\rangle\}$ を得ることができ，これらがそれぞれのテーブルにおける ‘土浦次郎’ のタプルの *Lineage* であることは明らかである．

本稿では Tracing Query の詳細については説明を省略するが，いずれにしても 3.2 節で述べた 7 つのオペレータで表現されるクエリであれば，上記のような関係代数で記述される Tracing Query を実行することで分析処理の Lineage を求めることができる．

しかしながら，分析結果のタプルには入力タプルの属性のほとんどが残っておらず，分析結果のタプルの情報からもとになった入力タプルを見つけるには情報が不足しすぎていることがある．そのため，任意のクエリに対して一回の Tracing Query で Lineage が求まるわけではない．このような場合には，分析処理の中間結果を用意して段階的に Tracing Query を実行することで Lineage を求める必要がある．また Function オペレータではベーステーブルに存在しない新たな値が導出されるため，もとになったベーステーブルのデータとの対応関係を辿るために少なくとも Function オペレータの出力は必ず Tracing Query を実行するための中間結果として必要である．

そのようなことを踏まえ，Augmented Lineage の導出手順は大きく分けて以下の 3 つの段階から構成される．

（1） Tracing Query の決定と，その実行に必要な中間結果の再計算．ただし再計算の際には，Function オペレータで実行する UDF は必ず Reasoning モードで実行し，Function オ

ペレータの出力にあたる中間結果には判断根拠の属性が含まれるようにする．

（2） Tracing Query のうち分析処理の最も出力側にあり，未実行のものを 1 つ実行．実行結果のテーブルに判断根拠が含まれていれば定義 4 で示した Reasoning Lineage を記録する．

（3） 未実行の Tracing Query がなければ終了．そうでなければ（2）に戻る．

上記の Augmented Lineage の導出手順では Tracing Query の実行時に分析処理を再実行して中間結果を計算するため，Augmented Lineage の導出が遅いという問題がある．そこで，中間結果に関する戦略として以下の 3 つを提案する．

- Rerun: 先ほどの説明の通り，Tracing Query の実行時に分析処理を再実行して中間結果を計算するやり方．Augmented Lineage の導出に時間がかかる．
- Full Materialization（Full）: 分析処理の実行時に Function オペレータで実行する UDF を Reasoning モードで実行し，全ての Tracing Query の実行に必要な中間結果を予めストアしておくやり方．Augmented Lineage の導出は高速に行えるが，中間結果をストアするためのストレージコストが大きい．
- Function Materialization（FM）: ストレージコストは抑えつつ，高速な Augmented Lineage 導出を目指すやり方．分析処理の実行時に Function オペレータで実行する UDF を Reasoning モードで実行し，Function オペレータの出力のみをストアしておくやり方．それ以外の中間結果は Tracing Query の実行時に再計算する．

Function Materialization は，Function オペレータの実行が他のリレーショナルオペレータの実行と比較すると非常にコストの大きい処理であることに着目し，Function オペレータの再実行さえ避けられれば Tracing Query に必要な中間結果を再計算する時間を大幅に削減できるのではないかという仮定に基づいている．

4 実　　験

本節では提案した中間結果の戦略ごとの Augmented Lineage の導出時間とストレージコストについて評価を行う．実験では，画像に対する人物認識を用いてイベントの登壇者数を調べるような分析処理を実行した．実験の画像は, [8], [9] のデータセットを使用している．実験で実行した分析処理は 2 つのテーブル（R_1, R_2）を入力にとっており，それぞれ 4.5×10^5 件と 6.075×10^6 件のデータを持っている．実装には PostgreSQL9.6 と Python3.7.8 を使用し，全ての実験は CPU として Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz，メモリは 128GB を搭載しているマシンで実行を行った．

Augmented Lineage の導出コスト: 分析処理に対する Augmented Lineage の導出にかかった時間を図 5 に，ストアした中間結果のデータ数を表 2 にまとめる．実験結果から，Function Materialization（FM）は Rerun より遥かに高速であり，Full Materialization（Full）と同等の実行時間を示した．加えて Function Materialization は，Full Materialization よりストアしておくデータ数を約 91%削減して Augmented Lineage を導出することができることが示された．このことから，Function Materialization は Rerun と Full Materialization のトレードオフを図る適切な戦略であることが示された．

表 2: 中間結果の戦略別の Augmented Lineage 導出のためにストアする必要のある中間結果のデータ数

	Full	FM
ストアするデータ数	4.95×10^5	4.5×10^4

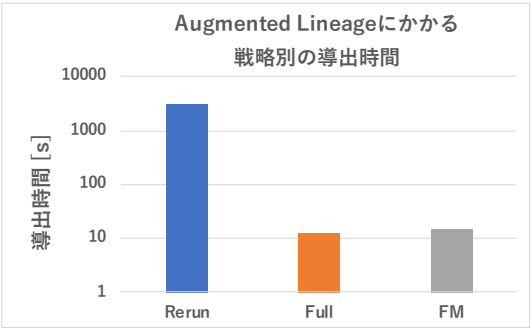

図 5: Augmented Lineage の導出時間

5 おわりに

本研究は，複合的データ解析を伴う分析処理に対するよりよいトレーサビリティを確保することを目的に，分析結果データのもとになったデータ（Lineage）と複合的データ解析における判断根拠をあわせて導出する Augmented Lineage のためのフレームワークの提案を行った．実験では Augmented Lineage の導出にかかるコストについて検証を行い，特に処理の中間結果の戦略に応じたコストの変化についての評価を行った．今後の予定としては，中間結果を削減するための枠組みの改良や他の手法との比較実験を検討している．

謝　　辞

本研究の一部は，日本学術振興会科学研究費・基盤研究（B）（#19H04114）ならびに新エネルギー・産業技術総合開発機構（NEDO）「人と共に進化する次世代人工知能に関する技術開発事業」による．

文　　献

[1] Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, p. 153–164, New York, NY, USA, 2011. Association for Computing Machinery.

[2] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: A survey. *ACM Comput. Surv.*, Vol. 37, No. 1, p. 1–28, March 2005.

[3] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and where: A characterization of data provenance. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory — ICDT 2001*, pp.

316–330, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[4] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, Vol. 25, No. 2, p. 179–227, June 2000.

[5] B. Glavic and G. Alonso. Perm: Processing provenance and data on the same data model through query rewriting. In *2009 IEEE 25th International Conference on Data Engineering*, pp. 174–185, 2009.

[6] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '07, p. 31–40, New York, NY, USA, 2007. Association for Computing Machinery.

[7] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar. A survey on provenance: What for? what form? what from? *The VLDB Journal*, Vol. 26, No. 6, pp. 881–906, 2017.

[8] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[9] Gary B. Huang Erik Learned-Miller. Labeled faces in the wild: Updates and new reporting procedures. Technical Report UM-CS-2014-003, University of Massachusetts, Amherst, May 2014.

[10] Fotis Psallidas and Eugene Wu. Smoke: Fine-grained lineage at interactive speed. *Proc. VLDB Endow.*, Vol. 11, No. 6, p. 719–732, February 2018.

[11] E. Wu, S. Madden, and M. Stonebraker. Subzero: A fine-grained lineage system for scientific databases. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pp. 865–876, 2013.