埋め込み順序従属性の検証アルゴリズム

ラモスアレハンドロ† 天方 大地† 原 隆浩†

† 大阪大学 〒 560–0085 大阪府吹田市山田丘 1-5
E-mail: †{ramos.alejandro,amagata.daichi,hara}@ist.osaka-u.ac.jp

あらまし 順序従属性（order dependencies）は，クエリ最適化，データ統合，およびデータクリーニング等の重要な応用例がある．多くの研究が順序従属性を発見する効率的なアルゴリズムを提案しているが，実世界データに通常現れる NULL に対応している順序従属性に関する研究は行われていない．本論文では，NULL がある場合の順序従属性に対応する，埋め込み順序従属性を新たに定義し，埋め込み順序従属性を検証する効率的なアルゴリズムを提案する．実世界データを用いた実験を行い，提案アルゴリズムの有効性を示す．

キーワード 埋め込み順序従属性, 不完全データ，関係データベース

1 はじめに

1.1 動機

整合性制約は，データプロファイル，データクリーニング，クエリ最適化，およびスキーマデザイン等の関係データベースで用いられる機能における基本である [1–3,6,15,16,20,22]．整合性制約の中でも，関数従属性 [4] および順序従属性 [7,19] は最も重要な制約である．関数従属性とは，ある属性の値が決まると，ある別の属性の値も自ずと決まる制約である．つまり，関数従属性 $X \to Y$ は，属性の集合 X において各属性の値が決まると，Y における各属性の値が決まる．同様に，順序従属性は，ある属性の順序（sort order）が決まると，ある別の属性の順序も自ずと決まる制約である．例えば，順序従属性 $X \mapsto Y$ は，X における全ての値が昇（降）順でソートされている時，Y も同様に（昇順または降順で）ソートされる．関数従属性は順序従属性の特別な場合に相当する [19,20] ため，本論文では順序従属性について考える．

順序従属性は特にクエリ最適化，制約違反検出，およびデータリペアに用いられており，これまでに順序従属性の発見に関する多くの研究が行われている [5,10,12,17,23]．また，エラーを含むデータに対応するため，順序従属性の亜種に関してもいくつかの提案がなされている [9,11,13,14]．しかし，実世界データにおいて NULL（missing value）を含むデータは一般的にも関わらず，このようなデータに対応する順序従属性に関する研究は行われていない．そこで本研究では，NULL を含むデータに対応する順序従属性について考え，埋め込み関数従属性（embedded functional dependencies）[21,22] で提案されている概念を取り入れる．この概念は，埋め込み属性というアイデアを用いており，埋め込み属性において NULL がないデータだけを制約の対象にする（NULL を含むデータを無視する）．この概念を取り入れることにより，既存の順序従属性では発見できないが意味的に有効な順序従属性を発見できる可能性がある．また，この方法で発見した順序従属性も整合性・完全性制約の定義に利用可能である．

表 1: 社員リスト

ID	Rank	Years	Age	Salary
t_1	1	1	20	15000
t_2	2	1	⊥	20000
t_3	3	2	22	25000
t_4	4	3	25	30000
t_5	5	4	30	35000

表 1 において， 順序従属性 $Rank \mapsto Salary$（Salary increases as Rank increases）を満たすか確認したとき，これは有効であることが分かる．しかし，既存の順序従属性において，$Age \mapsto Salary$ は有効でなく，これは t_2 が NULL（⊥）を含むためである．ここで，NULL を含まないデータのみを考えた時（t_2 を除外した場合），$Age \mapsto Salary$ は成り立つ．同様に，$Years \mapsto Salary$ も既存の順序従属性は満たさないが，NULL を考慮すると成り立つ．この例から分かるように，埋め込みを利用することによって（既存の順序従属性の定義では見つからない）意味的に有効な順序従属性を発見できる．そのため，本研究では属性のリストのペアおよび埋め込み属性の集合が与えられた時に，この埋め込み順序従属性が成り立つかを計算し，成り立たない場合はその順序従属性が成り立つ埋め込み属性の集合を計算する問題に取り組む．

1.2 課題

既存のアルゴリズムを用いれば，ある属性のリストのペアによる順序従属性が成り立つかを計算することは容易である．しかし，この順序従属性が成り立たない場合に，これを成り立たせる埋め込み属性の集合を計算するのは自明ではない．素朴なアルゴリズムとして，属性のペアを順に列挙して，この埋め込み上で順序従属性が成り立つか計算することを繰り返すものが考えられる．しかし，このアルゴリズムは属性数に対して階乗のコストがかかり，大規模なリレーショナルテーブルにスケールしない．例えばクエリ最適化において埋め込み順序従属性を利用する際，上記のアルゴリズムを用いると埋め込みの計算がボトルネックとなり，クエリ最適化に対するゲインが失われる．

そのため，本問題を解くための高速アルゴリズムが重要となる．

1.3 貢　　献

本研究では，上記の課題を解決する効率的なアルゴリズムを提案する．有効な埋め込みを計算するには，順序従属性に反するタプルが NULL の属性を持っているかどうかに注目すればよく，そのような属性を埋め込みに加えることで，順序従属性に反するタプルを無視できる．このアイデアを利用することにより，属性数に対する階乗の計算コストを取り除く．本研究の貢献は以下の通りである．

- 埋め込み順序従属性を提案する．この概念はこれまでに提案されていない．また，本研究では，与えられた属性のリストのペアおよび埋め込みがに対して順序従属性が成り立つか検証する問題に取り組む．
- 本問題に対して，行（タプル）数および列（属性）数に対する多項式時間アルゴリズムを提案する．
- 実世界データを用いて実験し，提案アルゴリズムの有効性を確認する．

構成． 以降では，2 章で問題を定義し，提案アルゴリズムを 3 章で紹介する．4 章で実験結果を報告し，5 章で関連研究を紹介する．最後に，6 章で本論文をまとめる．

2 予 備 知 識

2.1 定　　義

$\mathbf{R}$ をリレーショナルスキーマとし，$\mathbf{r}$ をリレーショナルデータベースのインスタンス（テーブル）とする．A および B を $\mathbf{R}$ の一つの属性とし，$\mathbf{X}$ および $\mathbf{Y}$ を属性のリストとする．また，s および t を $\mathbf{r}$ におけるタプル（行）とし，s_A を s における属性 A の値とする．理解を容易にするため，まず順序従属性を定義する．

定義 1. 順序従属性 [19]．任意の $s, t \in \mathbf{r}$ に対して，$s_X \leqq t_X \Rightarrow s_Y \leqq t_Y$ であるとき，順序従属性 $\mathbf{X} \mapsto_{\leqq} \mathbf{Y}$ は成り立つ（$\mathbf{X}, \mathbf{Y} \in \mathbf{R}$ である）．

ここで，$\mathbf{E}$ を $\mathbf{R}$ の部分集合とし，埋め込みと呼ぶ．このとき，$\mathbf{r}^{\mathbf{E}} \in \mathbf{r}$ は，$\mathbf{r}$ において $\mathbf{E}$ に NULL が無いタプルの集合を表す．次に，埋め込み順序従属性を定義する．

定義 2. 埋め込み順序従属性．任意の $s, t \in \mathbf{r}^{\mathbf{E}}$ に対して，$s_X \leqq t_X \Rightarrow s_Y \leqq t_Y$ であるとき，埋め込み順序従属性 $\mathbf{E} : \mathbf{X} \mapsto_{\leqq} \mathbf{Y}$ は成り立つ（$\mathbf{X}, \mathbf{Y} \in \mathbf{E} \subseteq \mathbf{R}$ である）．

2.2 問 題 定 義

本論文では，以下の問題を解く．

表 2: split，merge，および swap の例

id	A	B	C	D
t_1	1	3	1	2
t_2	2	7	1	3
t_3	3	4	3	4
t_4	4	5	5	5

入力： $\mathbf{E} : \mathbf{X} \mapsto_{\leqq} \mathbf{Y}$ が与えられた時，これが真であるか検証し，
出力： 真であれば，**valid** を返す．偽であれば，以下の二つのいずれかを返す：(1) $\mathbf{E}' : \mathbf{X} \mapsto_{\leqq} \mathbf{Y}$ が真である $\mathbf{E}'$ が存在する場合，**valid with** $\mathbf{E}'$ を返す．(2) 存在しない場合，**not valid** を返す．

上記の問題を解くにあたり，*split*，*swap* [19]，および *merge* [12] と呼ばれる概念を紹介する．これらは，順序従属性が成り立たない時のタプルのペアの関係性を定義したものである．

定義 3. Split．$s, t \in \mathbf{r}$ および $A, B \in \mathbf{R}$ において，$s_A = t_A$ だが $s_B \neqq t_B$ の場合，split していると呼ぶ．

定義 4. Merge．$s, t \in \mathbf{r}$ および $A, B \in \mathbf{R}$ において，$s_A \neqq t_A$ だが $s_B = t_B$ の場合，merge していると呼ぶ．

定義 5. Swap．$s, t \in \mathbf{r}$ および $A, B \in \mathbf{R}$ において，$s_A < t_A$ だが $s_B > t_B$ の場合，swap していると呼ぶ．

表 2 において，split，merge，および swap の例を紹介する．

例 1. $A \mapsto_{\leqq} B$ において，t_2 および t_3 は $t_{3_A} > t_{2_A}$ だが $t_{3_B} < t_{2_B}$ であるため，swap している．$A \mapsto_{<} C$ において，t_1 および t_2 は $t_{2_A} > t_{1_A}$ だが $t_{2_C} = t_{1_C}$ であるため，merge している．また，$C \mapsto_{\leqq} D$ において，t_1 および t_2 は $t_{2_C} = t_{1_C}$ だが $t_{2_D} > t_{1_D}$ であるため，split している．

上の定義から，(A, B) 上の split は，(B, A) 上の merge に相当する（逆も成り立つ）．また，split は “$\leqq$” による演算での概念，merge は “$<$” による演算での概念，さらに，swap はどちらの演算でも用いられる概念である．この観測に基づき，“$\leqq$” および “$<$” における埋め込み順序従属性がどのような場合において成り立つか紹介する．以下の（補助）定理は，文献 [12] で紹介された順序従属性における性質を埋め込み順序従属性に適応したものである．

補助定理 1. $\mathbf{E} : A \mapsto_{\leqq} B$ は，A, B で split および swap が無い場合に真である．

補助定理 2. $\mathbf{E} : A \mapsto_{<} B$ は，A, B で merge および swap が無い場合に真である．

定理 1. $\mathbf{E} : A \mapsto_{<} B$ は真である．$\Leftrightarrow \mathbf{E} : B \mapsto_{\leqq} A$ は真である．

この定理から，“$<$” の場合の検証を行うだけで良いことが分かる．

Algorithm 1: NAÏVE

Input: $\mathbf{E} : A \mapsto_{\leqq} B$

$\mathbf{S} \leftarrow \varnothing$ /* tuples that cause swaps */
$\mathbf{M} \leftarrow \varnothing$ /* tuples that cause merges */
$\mathbf{S}, \mathbf{M} \leftarrow$ FINDERRORS$(\mathbf{S}, \mathbf{M}, A, B)$
$\mathbf{S}, \mathbf{M} \leftarrow$ CHECKFORERRORDELETION$(\mathbf{S}, \mathbf{M}, \mathbf{E})$
if $(\mathbf{S} = \varnothing) \wedge (\mathbf{M} = \emptyset)$ **then**
  return valid /* no swaps or merges */
for $\forall \mathbf{E}' \in \mathbf{R}$ *such that* $\mathbf{E} \subset \mathbf{E}'$ **do**
  $\mathbf{S}' \leftarrow \varnothing, \mathbf{M}' \leftarrow \varnothing$
  $\mathbf{S}', \mathbf{M}' \leftarrow$ FINDERRORS$(\mathbf{S}', \mathbf{M}', A, B)$
  $\mathbf{S}', \mathbf{M}' \leftarrow$ CHECKFORERRORDELETION$(\mathbf{S}', \mathbf{M}', \mathbf{E}')$
  if $(\mathbf{S}' = \varnothing) \wedge (\mathbf{M}' = \varnothing)$ **then**
    return $\mathbf{E}'$
return not valid

2.3 素朴なアルゴリズム

ここで，本問題を解く素朴なアルゴリズム（Algorithm 1）を考える．定理 1 から，$\mathbf{E} : A \mapsto_{\leqq} B$ が与えられた時，$\mathbf{E} : B \mapsto_{<} A$ を検証すれば良い（“<” の際のエラーチェックの方が計算が容易である）．素朴なアルゴリズムは以下のステップで構成されている．

（1） $B \mapsto_{<} A$ に対する swap および merge を計算する．

（2） これらのエラーが $\mathbf{r}^{\mathbf{E}}$ で消える場合，valid を返す．

（3） 消えない場合，$\mathbf{E} \subset \mathbf{E}'$ である全ての $\mathbf{E}' \in \mathbf{R}$ に対して上のステップを繰り返す．$\mathbf{E}' : B \mapsto A$ が真の場合，$\mathbf{E}'$ を返す．

（4） 上のような $\mathbf{E}'$ が存在しなかった場合，not valid を返す．

このアルゴリズムは本問題を正確に解くが，$\mathbf{E}'$ が存在しない場合に，属性数に対して階乗の計算コストが発生する．そのため，本問題を効率的に解くアルゴリズムを設計する必要がある．

3 提案アルゴリズム

本章では提案アルゴリズムである VALIDEOD（Algorithm 2）を紹介する．

メインアイデア． $\mathbf{E} : A \mapsto_{<} B$ について考える．これが成り立たない場合，素朴なアルゴリズムは $\mathbf{E} \subset \mathbf{E}'$ である全ての $\mathbf{E}' \in \mathbf{R}$ を列挙しようとする．しかし，この順序従属性に反する swap または merge を起こしているタプルのペアが NULL を持っているかに着目すれば，考慮する必要がある属性のみを埋め込みに追加できる．（例えば，これらのタプルが NULL を持っていない場合，いかなる $\mathbf{E}' \supset \mathbf{E}$ でも $\mathbf{E}' : A \mapsto_{<} B$ は成り立たない．）このアイデアにより，属性数に対して線形時間のみ必要なアルゴリズムを設計できる．以下で，このアイデアの具体例を表 3 を用いて示す．

例 2. $AB : A \mapsto_{\leqq} B$ について考える．定理 1 から，$AB : B \mapsto_{<} A$ を検証する，表 3 において，t_2, t_3 が swap しているため，$AB : B \mapsto_{<} A$ は偽であり，$AB : A \mapsto_{\leqq} B$ も偽である．そのため，この順序従属性が成り立つ埋め込み $\mathbf{E}' \supset \mathbf{E}$ について，NULL を持つ属性に着目すると，$t_2.D = \perp$ である．属性 D を埋め込み AB に加えれば，t_2 が無視されるため，$ABD : B \mapsto_{<} A$ が成り立つ．そのため，埋め込みを列挙することなく ABD を返すことができる．

表 3: ValidEOD におけるメインアイデア（例 2 および 3）

id	A	B	C	D	F	G	H
t_1	4	1	1	8	20	10	1
t_2	6	2	3	$\perp$	30	$\perp$	2
t_3	5	3	5	10	$\perp$	50	3
t_4	7	4	5	12	40	100	$\perp$

Algorithm 2: VALIDEOD

Input: $\mathbf{E} : A \mapsto_{\leqq} B$

$\mathbf{S} \leftarrow \varnothing$ /* tuples that cause swaps */
$\mathbf{M} \leftarrow \varnothing$ /* tuples that cause merges */
$\mathbf{N} \leftarrow$ GETMISSINGVALUECOLUMNS
$\mathbf{S}, \mathbf{M} \leftarrow$ FINDERRORS$(\mathbf{S}, \mathbf{M}, A, B)$
$\mathbf{S}, \mathbf{M} \leftarrow$ CHECKFORERRORDELETION$(\mathbf{S}, \mathbf{M}, \mathbf{E})$
if $(\mathbf{S} = \varnothing) \wedge (\mathbf{M} = \varnothing)$ **then**
  return valid
else
  return UPDATEEMBEDDING$(\mathbf{E}, \mathbf{S}, \mathbf{M}, \mathbf{N})$

例 3. 次に，$CB : C \mapsto_{\leqq} B$，つまり，$CB : B \mapsto_{<} C$ について考える．このとき，t_3, t_4 に merge があるため，$CB : B \mapsto_{<} C$ は成り立たない．上の例と同様に，F を埋め込み CB に加えると，t_3 が無視される，つまり merge が消えるため，$CBF : B \mapsto_{<} C$ が成り立つ．これにより，全ての埋め込みを列挙することなく，$C \mapsto_{\leqq} B$ が成り立つ埋め込み（CBF）を計算できる．

概要． VALIDEOD は 2 フェーズにより構成されている．1 フェーズ目で与えられた埋め込み順序従属性が真か検証する．偽の場合，順序従属性が真となる埋め込みが存在するか計算する．

3.1 埋め込み順序従属性の検証

ある埋め込み順序従属性が与えられた際，VALIDEOD は以下によりこれが真であるか計算する．まず，NULL が含まれる属性の集合 $\mathbf{N}$ を計算する．次に，全ての swap と merge を FINDERRORS [12] で計算し，それらをそれぞれリスト $\mathbf{S}$ および $\mathbf{M}$ で管理する．（FINDERRORS は行数に対して線形時間しか必要としない．）その後，CHECKFORERRORDELETION でこれらのエラーが埋め込み $\mathbf{E}$ 上で消えるか計算する．（CHECKFORERRORDELETION の時間は $|\mathbf{S}|$ および $|\mathbf{M}|$ に対して線形である．）エラーが無い場合，valid を返す．エラーが存在する場合，次のフェーズに進む．

3.2 埋め込みの更新

次に，VALIDEOD は UPDATEEMBEDDING（Algorithm 3）により $\mathbf{E}' : A \mapsto_{\leqq} B$ が成り立つ $\mathbf{E}' \supset \mathbf{E}$ を探す．メインアイ

Algorithm 3: UPDATEEMBEDDING

Input: $\mathbf{E}$: a given embedding, $\mathbf{S}$: a list of tuples that cause swaps, $\mathbf{M}$: a list of tuples that cause merges, and $\mathbf{N}$: a set of attributes with missing values.

$\mathbf{E}' \leftarrow \mathbf{E}$
for *each* $s \in \mathbf{S}$ **do**
  for *each* $A \in \mathbf{N}$ **do**
    $\mathbf{F} \leftarrow \mathbf{E}' + A$
    if s *is removed on* $\mathbf{r}^{\mathbf{F}}$ **then**
      $\mathbf{E}' \leftarrow \mathbf{F}$
      $\mathbf{S} \leftarrow \mathbf{S} - s$
      break
  if $s \in \mathbf{S}$ **then**
    return not valid
for *each* $m \in \mathbf{M}$ **do**
  for *each* $A \in \mathbf{N}$ **do**
    $\mathbf{F} \leftarrow \mathbf{E}' + A$
    if m *is removed on* $\mathbf{r}^{\mathbf{F}}$ **then**
      $\mathbf{E}' \leftarrow \mathbf{F}$
      $\mathbf{M} \leftarrow \mathbf{M} - m$
      break
  if $m \in \mathbf{M}$ **then**
    return not valid
return valid under $\mathbf{E}'$

デアで述べたように，swap または merge が起きているタプルが NULL となる属性を持っているかだけ確認すれば良い．(もしそのような属性がなければ，順序従属性は成り立たない．) このアイデアを用いて，swap および merge が消えるように埋め込み $\mathbf{E}$ を更新する．UPDATEEMBEDDING は以下の手順で $\mathbf{E}$ を更新する：

(1) 各 swap $s \in \mathbf{S}$ に対して，$\mathbf{N}$ に含まれる属性が s を消せるか確認する．消せるならば，該当の属性を $\mathbf{E}'$（初期は $\mathbf{E}$）に追加し，s を $\mathbf{S}$ から取り除く．これを $\mathbf{N}$ に含まれる全ての属性に対してテストしても s が消えないならば，$\mathbf{S} \neq \varnothing$ である．この時，UPDATEEMBEDDING は not valid を返す．

(2) 上のステップで not valid を返していなければ，同様の操作を各 merge $m \in \mathbf{M}$ に対して行う．

(3) 上のステップでエラーが消えれば，UPDATEEMBEDDING は valid with $\mathbf{E}'$ を返す．

3.3 分　　析

空間計算量． VALIDEOD の空間計算量は，$O(|\mathbf{S}|+|\mathbf{M}|+|\mathbf{N}|)$ である．

時間計算量． 1 フェーズ目の計算量は，$O(n+|\mathbf{S}|+|\mathbf{M}|)$ であり，n は $\mathbf{r}^{\mathbf{E}}$ の行数である．2 フェーズ目の計算量は，$O(|\mathbf{N}|(|\mathbf{S}|+|\mathbf{M}|))$ である．また，$|\mathbf{S}|+|\mathbf{M}|$ は最悪の場合に $O(n^2)$ である．そのため，VALIDEOD の時間計算量は，$O(|\mathbf{N}|n^2)$ である．これにより，VALIDEOD の時間計算量は属性数に対して線形であり，素朴なアルゴリズムのような階乗コストは発生しないことが分かる．(また，実践的には $|\mathbf{S}|+|\mathbf{M}| \ll O(n^2)$ である．)

表 4: 各データの行数および列数

データ	行（データ）数	列（属性）数
China Weather	262,000	18
Adult	32,000	15
Hepatitis	155	20
Echocardiogram	132	13
NCVoter	256,000	19

4 実　　験

本章では評価実験の結果を報告する．全ての実験は，Ubuntu 20.04.5 を搭載した Intel Core i9-12900@2.4GHz および 64GB RAM の計算機を用いて行った．

4.1 設　　定

データ． 本実験では，以下の実データ [1][2]を用いた．

- China Weather：中国における気象データ．
- Adult：アメリカ合衆国における国勢調査データ．
- Hepatitis: 肝炎患者に関するデータ．
- Echocardiogram：心発作を起こした患者に関する分類データ．
- NCVoter：North Carolina 州の投票者の個人データ．

アルゴリズム． 本実験では，Algorithm 2 を Algorithm 1 と比較した．これらのアルゴリズムはシングルスレッドであり，C++で実装され，g++ 9.4.0 および-O3 による最適化でコンパイルされている．

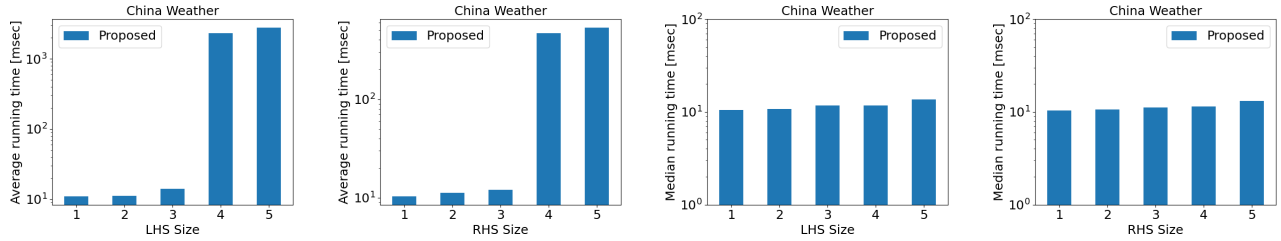
パラメータ． 性能評価のため，LHS および RHS のサイズを変えて実験を行った．$\mathbf{X} \mapsto_{\leq} \mathbf{Y}$ において，$\mathbf{X}$（$\mathbf{Y}$）が LHS（RHS）である．デフォルトの LHS および RHS のサイズは 1 とし，LHS と RHS は属性集合からランダムにサンプルした属性によるリストである．

各実験は，ランダムに作成した LHS と RHS 上でのアルゴリズムの実行を 10 回繰り返している．実行時間はその平均である．

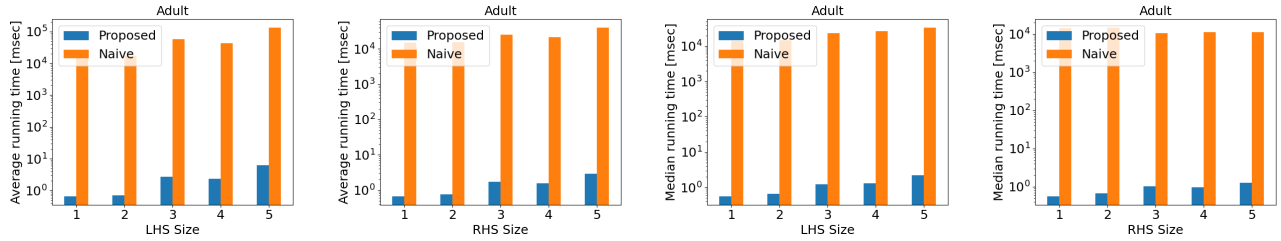
4.2 結　　果

図 1 は，提案アルゴリズム（Algorithm 2）および素朴な（Naïve）アルゴリズム（Algorithm 1）の各 LHS（RHS）のサイズに対する実行時間の平均と中央値を表している．また，China Weather および NCVoter において，素朴なアルゴリズムは 10 回の繰り返しに 2 時間以上要したため，実験を中断し，

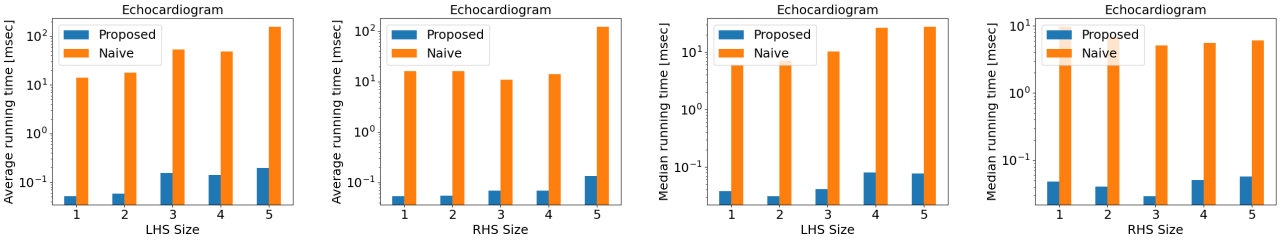
1：https://www.ncsbe.gov/
2：https://archive.ics.uci.edu/



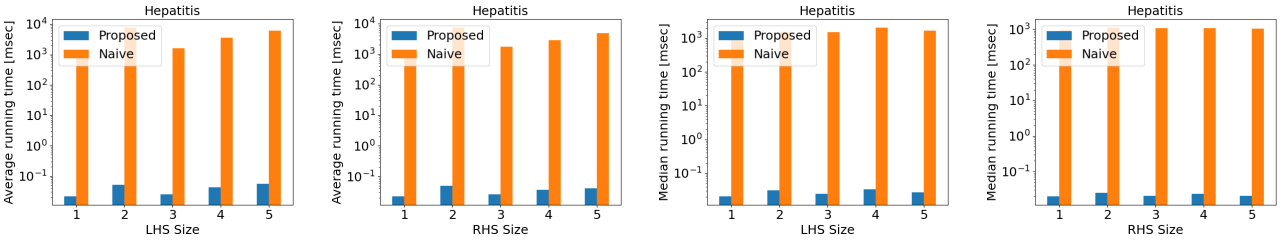
(a) China Weather



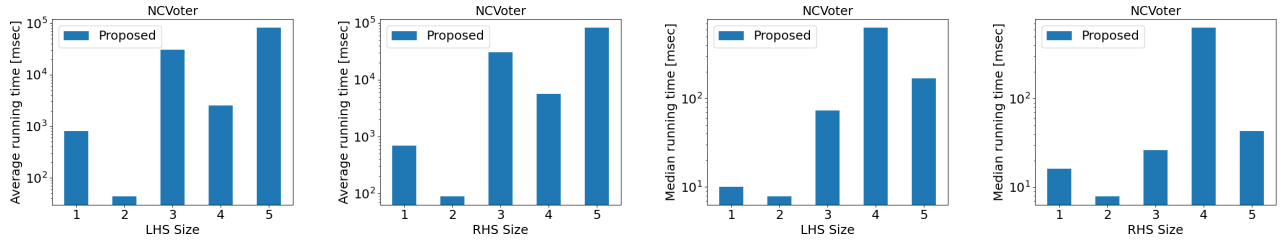
(b) Adult



(c) Echocardiogram



(d) Hepatitis



(e) NCVoter

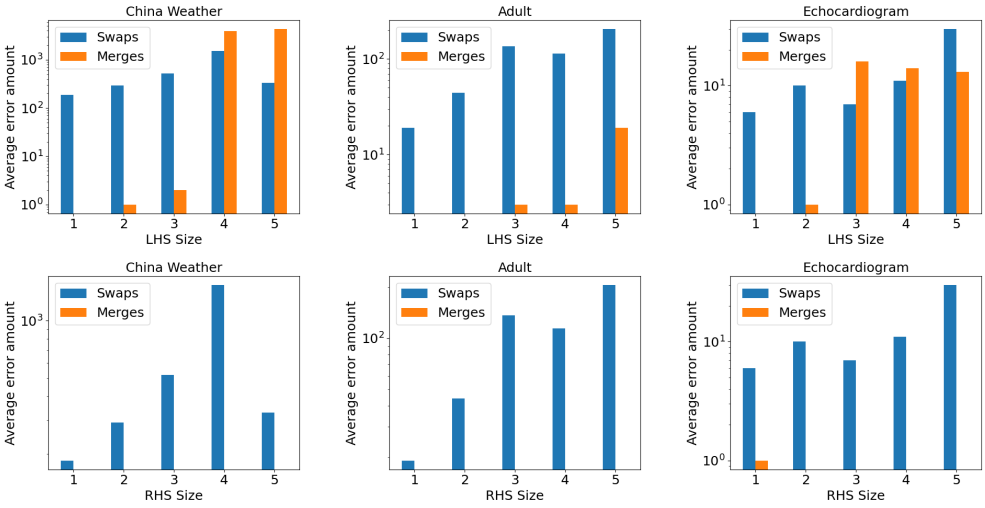
図 1: LHS および RHS のサイズの影響

結果を割愛している．

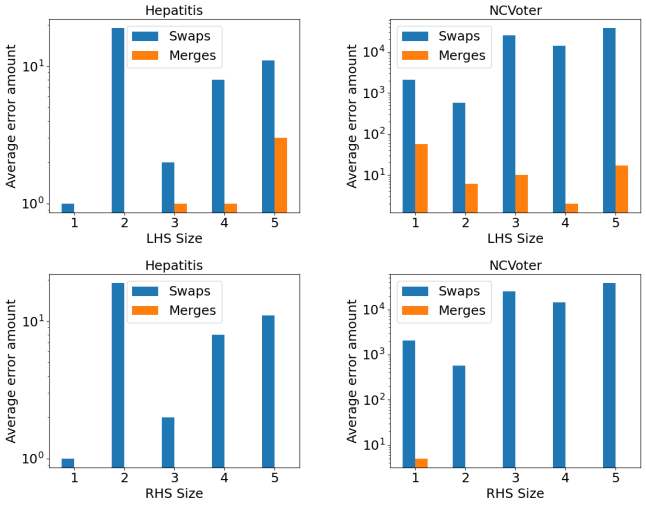
まず，提案アルゴリズムは素朴なアルゴリズムよりも何百倍以上も早いことが分かる．本実験で用いたデータは 10 以上の属性を持っているため，この数に対する階乗コストは非常に大きい．そのため，素朴なアルゴリズムは実行時間が膨大となるが，提案アルゴリズムには影響が小さい．

また，提案アルゴリズムは LHS（RHS）のサイズに対して線形な影響がないことも分かる．これは，3.3 節における分析から自明であり，LHS（RHS）のサイズと $|\mathbf{S}| + |\mathbf{M}|$ に線形な関係はない．図 2 に，$|\mathbf{S}|$ および $|\mathbf{M}|$ の平均を示す．図 1 における提案アルゴリズムの平均実行時間と比較すると，$|\mathbf{S}| + |\mathbf{M}|$ と提案アルゴリズムの平均実行時間は相関しており，理論分析通りの挙動であることが分かる．

最後に，いくつかのデータ（例えば China Weather や NCVoter）において，実行時間の平均と中央値に乖離があることが分かる．これは，いくつかの LHS および RHS にペアは，大量の swap および merge を発生していることを示しており，このようなペアによって平均時間が長くなる．中央値の結果に



(a) China Weather (b) Adult (c) Echocardiogram



(d) Hepatitis (e) NCVoter

図 2: $|\mathbf{S}|$ および $|\mathbf{M}|$ の平均

着目すると，先述のようなペアとなる確率は高くなく，提案アルゴリズムが高速に解を返す場合が多いことも分かる．

5 関 連 研 究

順序従属性という概念は，文献 [7] によって提唱され，順序従属性の計算複雑性について議論されている．文献 [19, 20] では，理論的な複雑性が証明されており，順序従属性の公理が示されている．また，順序従属性が関数従属性を包含していることも示された．

5.1 厳密な順序従属性

文献 [12] では，与えられた関係表で成り立つ全ての順序従属性を列挙するアルゴリズムである ORDER が提案されている．ORDER はいつくかのフィルタリング技術が組み込まれているものの，属性数に対して階乗のコストが生じる．文献 [17] は，特定のソート方法における順序従属性を集合に基づく表現に変換させることにより，属性数に対する最悪時間を階乗から指数時間に削減できることを示した．

順序従属性は NULL の無いデータに対して有効であるが，NULL が一つでも含まれるデータには使えず，潜在的な順序従属性を発見できない．(埋め込み順序従属性はこれを見つける可能性を残している．) また，順序従属性は埋め込み順序従属性のスペシャルケースであり，埋め込みが全属性の場合であることを記す．

5.2 近似順序従属性

文献 [17, 18] では，近似順序従属性が提案されている．これは，いくつかの（順序従属性に反している）タプルのペアを除く場合に成り立つ順序従属性を計算する問題である．しかし，この問題は行数および列数に対して大きな計算コストがかかることが示されている．文献 [8, 9, 11] で，この課題に対応するヒューリスティックアルゴリズムが提案されている．

上記の研究において，全てのデータに NULL がないことを

想定しており，NULL がある場合の近似順序従属性は定義されていない．そのため，近似順序従属性は完全性制約に対応できないという欠点がある．

5.3 埋め込み関数従属性

NULL ありデータに対応する関数従属性である埋め込み関数従属性は，文献 [22] で提案された．また，文献 [21] において，埋め込み関数従属性を満たす属性集合のペアを高速に計算するアルゴリズムが提案されている．埋め込み関数従属性も完全性制約や整合性制約に使えるが，順序従属性には対応できない．上記の通り，順序従属性は関数従属性を包含しており，埋め込み順序従属性も埋め込み関数従属性を包含していることは自明である．

6 まとめ

本論文では，埋め込み順序従属性という新しい概念を提案し，NULL ありデータに対する順序従属性，統合性，および完全性への対応を実現した．また，与えられた属性のリストのペアおよび埋め込みが成り立つか検証する問題に取り組み，これを効率的に解くアルゴリズムを提案した．提案アルゴリズムは，データ数および属性数に対して多項式時間で動作することを示した．実世界データを用いた実験により，提案アルゴリズムの有効性を示した．

謝辞

本研究の一部は，JST さきがけ（JPMJPR1931），JST CREST（JPMJCR21F2），および文部科学省科学研究費補助金・基盤研究 (A)(18H04095) の支援を受けたものである．

文献

[1] S. Abiteboul, R. Hull, and V. Vianu, Foundations of databases, vol.8, 1995.
[2] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, “Conditional functional dependencies for data cleaning,” In ICDE, pp.746–755, 2006.
[3] Q. Cheng, J. Gryz, F. Koo, T.C. Leung, L. Liu, X. Qian, and B. Schiefer, “Implementation of two semantic query optimization techniques in db2 universal database,” In VLDB, vol.99, pp.687–698, 1999.
[4] E.F. Codd, “Further normalization of the data base relational model,” Data base systems, vol.6, pp.33–64, 1972.
[5] C. Consonni, P. Sottovia, A. Montresor, Y. Velegrakis, Z. Kaoudi, H. Galhardas, I. Fundulaki, B. Reinwald, M. Herschel, C. Binnig, et al., “Discovering order dependencies through order compatibility,” In EDBT, 2019.
[6] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, “Conditional functional dependencies for capturing data inconsistencies,” ACM Transactions on Database Systems, vol.33, no.2, pp.1–48, 2008.
[7] S. Ginsburg and R. Hull, “Order dependency in the relational model,” Theoretical computer science, vol.26, no.1-2, pp.149–195, 1983.
[8] Y. Jin, Z. Tan, J. Chen, and S. Ma, “Discovery of approximate lexicographical order dependencies,” IEEE Transactions on Knowledge and Data Engineering, 2021.
[9] Y. Jin, Z. Tan, W. Zeng, and S. Ma, “Approximate order dependency discovery,” In ICDE, pp.25–36, 2021.
[10] Y. Jin, L. Zhu, and Z. Tan, “Efficient bidirectional order dependency discovery,” In ICDE, pp.61–72, 2020.
[11] R. Karegar, P. Godfrey, L. Golab, M. Kargar, D. Srivastava, and J. Szlichta, “Efficient discovery of approximate order dependencies,” arXiv preprint arXiv:2101.02174, 2021.
[12] P. Langer and F. Naumann, “Efficient order dependency detection,” The VLDB Journal, vol.25, no.2, pp.223–241, 2016.
[13] P. Li, J. Szlichta, M. Bohlen, and D. Srivastava, “Discovering band order dependencies,” In ICDE, pp.1878–1881, 2020.
[14] P. Li, J. Szlichta, M. Böhlen, and D. Srivastava, “Abc of order dependencies,” The VLDB Journal, vol.31, no.5, pp.825–849, 2022.
[15] F. Naumann, “Data profiling revisited,” ACM SIGMOD Record, vol.42, no.4, pp.40–49, 2014.
[16] D. Simmen, E. Shekita, and T. Malkemus, “Fundamental techniques for order optimization,” In SIGMOD, pp.57–67, 1996.
[17] J. Szlichta, P. Godfrey, L. Golab, M. Kargar, and D. Srivastava, “Effective and complete discovery of order dependencies via set-based axiomatization,” arXiv preprint arXiv:1608.06169, 2016.
[18] J. Szlichta, P. Godfrey, L. Golab, M. Kargar, and D. Srivastava, “Effective and complete discovery of bidirectional order dependencies via set-based axioms,” The VLDB Journal, vol.27, no.4, pp.573–591, 2018.
[19] J. Szlichta, P. Godfrey, and J. Gryz, “Fundamentals of order dependencies,” Proceedings of the VLDB Endowment, vol.5, no.11, pp.1220–1231, 2012.
[20] J. Szlichta, P. Godfrey, J. Gryz, and C. Zuzarte, “Expressiveness and complexity of order dependencies,” Proceedings of the VLDB Endowment, vol.6, no.14, pp.1858–1869, 2013.
[21] Z. Wei, S. Hartmann, and S. Link, “Algorithms for the discovery of embedded functional dependencies,” The VLDB Journal, vol.30, no.6, pp.1069–1093, 2021.
[22] Z. Wei and S. Link, “Embedded functional dependencies and data-completeness tailored database design,” Proceedings of the VLDB Endowment, vol.12, no.11, pp.1458–1470, 2019.
[23] L. Zhu, X. Sun, Z. Tan, K. Yang, W. Yang, X. Zhou, and Y. Tian, “Incremental discovery of order dependencies on tuple insertions,” In DASFAA, pp.157–174, 2019.