情報処理学会論文誌 Vol.58 No.12 1964–1976 (Dec. 2017)

Allenの時区間関係を条件とする時区間データの効率的な結合演算

山下 夏奈[1] 猪口 明博[1,a)]

受付日 2017年3月23日, 採録日 2017年9月5日

概要：Overlap Interval Partitioning（$\mathcal{OIP}$）は，交わりを持つ2つの時区間（タプル）の結合演算を効率良く解くためのデータ構造である．しかしながら，このデータ構造をAllenの時区間関係の1つを満たす2つの時区間の結合演算に用いると，結合の回数が増えたり，結合の必要のない2つのタプルに対して，条件節の真偽を確認する必要があり，非効率である．そこで，本稿では2次配列型のデータ構造を持つPartition Arrayを提案し，必要最低限の結合を行い，効率的な結合演算を可能にする．提案手法Partition Arrayと既存手法$\mathcal{OIP}$を実装し，計算時間の違いについて考察することにより，提案手法の有効性を評価・検証する．

キーワード：時制データベース，時区間データ，結合演算，索引，Allenの時区間関係

Efficient Join Operation for Temporal-interval Data under the Allen's Interval Algebra

Kana Yamashita[1] Akihiro Inokuchi[1,a)]

Received: March 23, 2017, Accepted: September 5, 2017

Abstract: Overlap interval Partitioning ($\mathcal{OIP}$) is a data structure for efficiently joining intersecting temporal intervals (tuples). However, when an $\mathcal{OIP}$ structure is used for two temporal intervals satisfying one of the Allen's interval relationships, the number of joins increases and the truth or falsehood of the conditional clause must be confirmed even for two tuples that do not need to be joined, which is inefficient. Therefore, in this paper, we propose a Partition array with a two-dimensional array data structure that performs the minimum number of joins and enables efficient join operations. We implemented the proposed Partition array method and the existing $\mathcal{OIP}$ method to evaluate the validity of the proposed method in terms of computation time.

Keywords: temporal database, temporal interval, join operation, index, Allen's interval algebra

1. はじめに

近年，情報通信技術の発展により，様々なデータを半自動的に収集できるようになった．たとえば，スーパーマーケットやコンビニエンスストアでは，店舗で商品を販売するごとに商品の販売情報を記録するPOSシステムの導入により，商品名や価格，数量などの購買データが蓄積される．また，インターネットショピングを利用すると，商品の閲覧履歴や注文履歴，購入日時などが記録される．多種多様なデータの中でも，市場の流行や消費者の購買行動，購買意識の変化を調べるときには時区間を持つデータの分析が重要である．ここで，時区間とは，始点と終点を持つ時間領域のことである．先ほどの例では，顧客の来店時間や商品の購入日時，注文日時などが時区間を持つデータである．これまでの業務システムにより蓄積された時区間データを分析することで，顧客ごとのニーズに基づいた商品やサービスの提供が可能になる．

時区間を含むタプルからなるリレーションが2つある

[1] 関西学院大学理工学部情報科学科
Department of Informatics, School of Science and Technology, Kwansei Gakuin University, Sanda, Hyogo 669–1337, Japan
[a)] inokuchi@kwansei.ac.jp

© 2017 Information Processing Society of Japan

とき，特定の時区間関係にあるタプルの対を抽出するために，すべてのタプルの対をとるのは効率が悪い．既存手法である Overlap Interval Partitioning（$\mathcal{OIP}$）では，リレーションの時間範囲を k 個の等しい長さの最小構成時区間に分割する．また，1 つ以上の隣接した最小構成時区間の集合でパーティションを構成する．リレーションに格納されている各タプルは，タプルの時区間を完全に覆うパーティションの中で最小のものに割り当てられる．そして，各パーティションを Lazy Partition List と呼ばれる木構造の索引で保持する．各パーティションを木構造で保持することで，抽出したいタプルが格納されている可能性があるパーティションを深さ優先探索して得ることができ，効率的である．$\mathcal{OIP}$ では，Allen の時区間関係の $after(A,B)$，$before(A,B)$ 以外の 11 個の時区間関係の論理和をサポートできる．しかしながら，論理和でなく，各時区間関係を条件とする結合演算に適しているとはいえない．そこで本稿では，各時区間関係を条件とする結合演算に対して効率的な結合演算を可能にする索引と，それに対する探索手法を提案する．

本稿では，オフラインの検索システムを対象とする．このため，索引の構築時間については言及しない．また，データベースに対するタプルの追加，削除，更新はなされないものとする．ただし，提案手法がタプルの追加，削除，更新に対して対応可能であることを，6 章において議論する．

本稿の構成は，以下のとおりである．2 章では，本稿で扱う問題を定義し，既存手法である $\mathcal{OIP}$ を説明し，3 章では，その課題を説明する．4 章では，$\mathcal{OIP}$ と Allen の時区間関係についての関連を述べ，提案手法について述べる．5 章では，人工データを用いて，提案手法と $\mathcal{OIP}$ の性能を比較し，その結果に基づいて，6 章では，提案手法と関連研究とデータベースの更新について議論する．最後に，7 章で本稿をまとめる．

2. 時区間の結合演算とその既存手法

2.1 問題定義

本稿では，離散時間領域 Ω^T は線形に並んだタイムポイントの集合で構成されているものとする．時区間 T はタイムポイントの対であり，$[T_S, T_E]$ で表される．ここで，T_S と T_E はそれぞれ T の始点と終点である．タイムポイントと時区間に対して，以下に記す演算を用いる．

- タイムポイント x が時区間 T に含まれている，つまり $T_S \le x \le T_E$ ならば，$x \in T$ と記す．
- 時区間 Q と T が交わる，つまり $x \in Q \wedge x \in T$ を満たすタイムポイント x が存在するならば，$Q \cap T \neq \emptyset$ と記す．
- 時区間 T が時区間 U に含まれている，つまり $\forall x \in T \Rightarrow x \in U$ を満たすタイムポイント x が存在するならば，$T \subseteq U$ と記す．
- $T_E - T_S$ は T_S と T_E のタイムポイントの差を意味する．
- $T_S - x$ は，タイムポイント T_S を x だけ過去にずらすことを意味し，$T_E - T_S = x \Rightarrow T_S + x = T_E$ が成り立つ．
- $|T| = (T_E - T_S) + 1$ は，時区間 T の存続時間を意味する．

時制リレーションスキーマを $R = (A_1, \cdots, A_m, T)$ と表す．ここで，$1 \le i \le m$ に対して，A_i は領域 Ω_i に属し，時区間 T は領域 $\Omega^T \times \Omega^T$ に属する．スキーマ R における時制リレーション $\mathbf{r}$ は，R におけるタプルの有限集合である．もし $\mathbf{r}$ のタプルの中で最小の始点が U_S であり，最大の終点が U_E であるならば，$\mathbf{r}$ の時間範囲は $U = [U_S, U_E]$ である．

文献 [4] で対象とされた問題は，2 つのリレーション $\mathbf{r}$ と $\mathbf{s}$ が入力として与えられたとき，時区間が交わりを持つタプルの対を求める結合演算 $\{(r,s) \mid r \in \mathbf{r} \wedge s \in \mathbf{s} \wedge r.T \cap s.T \neq \emptyset\}$ を効率良く求めることである．

2.2 パーティション

本節では，上記の問題を解く既存手法として $\mathcal{OIP}$ [4] を説明する．$\mathcal{OIP}$ ではリレーション $\mathbf{r}$ の時間範囲 $U = [U_S, U_E]$ は k 個の等しい長さの最小構成時区間に分割される．

定義 1 ($\mathcal{OIP}$ Configuration)．k が与えられたとき，$\mathbf{r}$ の $\mathcal{OIP}$ Configuration は $(\mathbf{r}, k)$ で表される[*1]． ■

パーティションは，1 つ以上の隣接した最小構成時区間の集合である．各タプルは，タプルの時区間を覆うパーティションの中で最小のものに割り当てられる．

定義 2 ($\mathcal{OIP}$ Partition)．$\mathcal{OIP}$ Configuration $(\mathbf{r}, k)$ が与えられ，最小構成時区間に対して，時間順に 0 から $k-1$ の ID が振られているものとし，最小構成時区間の長さ $d = \lceil \frac{|U|}{k} \rceil$ とする．$\mathcal{OIP}$ のパーティション $p_{i,j}$ は i 番目から j 番目の最小構成時区間にまたがり，その時区間は $[U_S + i \cdot d, U_S + (j+1) \cdot d - 1]$ である．ここで，$0 \le i \le j < k$ である．タプル $r \in \mathbf{r}$ は，$i = \lfloor \frac{r.T_S - U_S}{d} \rfloor$ かつ $j = \lfloor \frac{r.T_E - U_S}{d} \rfloor$ であるとき，$p_{i,j}$ に割り当てられる． ■

存在しうる $\mathcal{OIP}$ のパーティションの数は，$\frac{k(k+1)}{2}$ である．また，$c = p_{i,j}$ のとき，$c.i$ と $c.j$ はそれぞれパーティションの始点の ID と終点の ID を表す．

例 3. 図 1 にあるリレーション $\mathbf{s}$ は $\{s_1, \cdots, s_7\}$ の 7 つのタプルを含み，時間範囲は $U = [\text{2012-1}, \text{2012-12}]$ である．$k = 4$ であるとき，$d = \lceil \frac{|U|}{k} \rceil = \lceil \frac{|12|}{4} \rceil = 3$ である．また，$p_{0,1}$ の範囲は $[\text{2012-1}, \text{2012-6}]$ である．タプル s_6 について，$i = \lfloor \frac{s_6.T_S - o}{d} \rfloor = \lfloor \frac{\text{2012-6} - \text{2012-1}}{3} \rfloor = \lfloor \frac{5}{3} \rfloor = 1$，$j = \lfloor \frac{s_6.T_E - o}{d} \rfloor = \lfloor \frac{\text{2012-10} - \text{2012-1}}{3} \rfloor = \lfloor \frac{9}{3} \rfloor = 3$ であることより，s_6 はパーティション $p_{1,3}$ に割り当てられる．10 個の

[*1] 文献 [4] では，$\mathcal{OIP}$ Configuration は (k, d, U_S) で表される．しかし d や U_S は k と $\mathbf{r}$ から導出できるので，本稿では $(\mathbf{r}, k)$ の表現を用いる．

© 2017 Information Processing Society of Japan

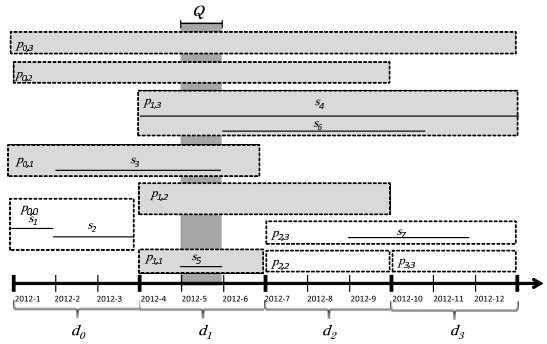

図 1 Configuration $(\mathbf{s}, 4)$ に対する $\mathcal{OIP}$
Fig. 1 $\mathcal{OIP}$ with Configuration $(\mathbf{s}, 4)$.

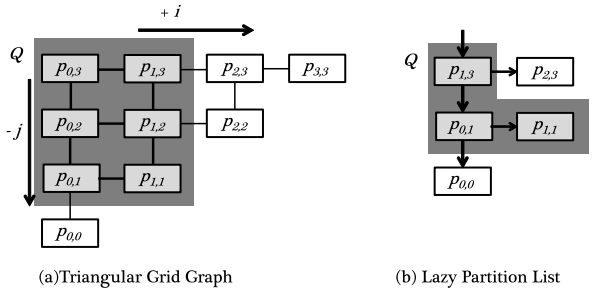

図 2 $\mathcal{OIP}$ Partition の管理
Fig. 2 Management of $\mathcal{OIP}$ Partitions.

うち 5 個のパーティションは空である． □

補題 4. $\mathcal{OIP}$ Configuration $(\mathbf{r}, k)$ とクエリ時区間 $q = [q.T_S, q.T_E]$ が与えられたとき，q と交わりを持つ可能性があるタプル $\{r \mid r \in \mathbf{r}, r \cap q \neq \emptyset\}$ は，$i \leq \lfloor \frac{q.T_E - U_S}{d} \rfloor$ かつ $j \geq \lfloor \frac{q.T_S - U_S}{d} \rfloor$ を満たすパーティション $p_{i,j}$ に存在する． ■

例 5. 図 1 に示す $\mathcal{OIP}$ Configuration $(\mathbf{s}, 4)$ とクエリ時区間 $q = [2012\text{-}5, 2012\text{-}5]$ が与えられているとする．クエリと交わりを持つ可能性があるタプルは，$i \leq \lfloor \frac{2012\text{-}5 - 2012\text{-}1}{3} \rfloor = \lfloor \frac{4}{3} \rfloor = 1$ かつ $j \geq \lfloor \frac{2012\text{-}5 - 2012\text{-}1}{3} \rfloor = \lfloor \frac{4}{3} \rfloor = 1$ であるパーティション $p_{i,j}$ にある．これを満たすパーティションは灰色に塗られている $p_{0,3}$，$p_{0,2}$，$p_{0,1}$，$p_{1,3}$，$p_{1,2}$，$p_{1,1}$ である． □

2.3 Lazy Partition List

図 2 (a) に，図 1 の $\mathcal{OIP}$ のデータ構造を示す．このグラフは Triangular Grid Graph と呼ばれ，インメモリで管理される．クエリ時区間と結合可能なすべてのパーティション $p_{i,j}$ は $j \geq \lfloor \frac{q.T_S - U_S}{d} \rfloor$ かつ $i \leq \lfloor \frac{q.T_E - U_S}{d} \rfloor$ を満たす．ここで U と d はそれぞれ $\mathbf{s}$ の時間範囲の始点とパーティションの最小構成時区間の長さである．これを満たすパーティションを見つけるためにグラフの左端最上部から探索を開始し，$j \geq \lfloor \frac{q.T_S - U_S}{d} \rfloor$ である間，j が減少する方向に移動する．そして $p_{0,j}$ であるノードから，$i \leq \min(j, \lfloor \frac{q.T_E - o_s}{d_s} \rfloor)$ である間，i が増加する方向に移動する．例 5 に示したクエリ時区間 q と結合可能なパーティションは，図 2 (a) の灰色で強調されている 6 個である．

一方，Lazy Partition List は，Triangular Grid Graph から空のパーティションを除いたものであり，各節点は right と down の 2 つのリンクを持つ．図 2 (b) は図 1 の Lazy Partition List を示している．この図に示されるように，クエリ時区間と交わりを持つタプルを含むパーティションは連結する部分グラフであり，必ず根を含む．したがって，根から探索することでこれらのパーティションに漏れなく，また無駄なく到達し，結合演算を取ることができる．

Algorithm 1: OIPJOIN

Data: Lazy Partition lists $\mathcal{L}_r$ and $\mathcal{L}_s$, $\mathcal{OIP}$ Configurations $(\mathbf{r}, k_r)$ and $(\mathbf{s}, k_s)$
Result: $\mathbf{z} = \{(r, s) \mid r \in \mathbf{r} \wedge s \in \mathbf{s} \wedge r.T \cap s.T \neq \emptyset\}$

```
1  z := ∅;
2  [U_S^r, U_E^r] := r の時間範囲;
3  [U_S^s, U_E^s] := s の時間範囲;
4  d_s := ⌈(U_E^s − U_S^s + 1) / k_s⌉ ;
5  d_r := ⌈(U_E^r − U_S^r + 1) / k_r⌉ ;
6  for node c_r in L_r do
7      Q_S := U_S^r + c_r.i · d_r;
8      Q_E := U_S^r + (c_r.j + 1) · d_r − 1;
9      if Q_E ≥ U_S^s ∧ Q_S < U_S^s + k_s · d_s then
10         s := ⌊(Q_S − U_S^s) / d_s⌋;
11         e := ⌊(Q_E − U_S^s) / d_s⌋;
12         c_s := L_s.head;
13         while c_s ≠ null ∧ c_s.j ≥ s do
14             x := c_s;
15             while x ≠ null ∧ x.i ≤ e do
16                 z := z ∪ { joined tuples from c_r and x};
17                 x := x.right;
18             c_s := c_s.down;
19 return z;
```

2.4 Lazy Partition List を用いた $\mathcal{OIP}$ の実行

リレーション $\mathbf{r}$ と $\mathbf{s}$ に対する OIPJOIN アルゴリズムを Algorithm 1 に示す．Lazy Partition List $\mathcal{L}_r$ が持つ各パーティションの時区間 c_r と交わりと持つパーティションを $\mathcal{L}_s$ から探索する．$\mathcal{L}_s$ の根から探索をはじめ，right のリンクを優先に深さ優先探索する．もし，c_r と交わりを持つパーティション x に到達したら，2 つのパーティションの結合演算をとり，結果を $\mathbf{z}$ に追加する．一方，c_r と交わりを持たない節点に到達したら，それ以降に c_r と交わりを持つパーティションは存在しないので，バックトラックする．これを図 1 の $\mathbf{s}$ と**図 3** の $\mathbf{r}$ に対して適用することで最終的に $\mathbf{z} = \{(r_3, s_4), (r_3, s_6), (r_3, s_7), (r_1, s_3), (r_1, s_4), (r_1, s_5), (r_2, s_4), (r_2, s_6)\}$ が出力される．

© 2017 Information Processing Society of Japan

3. $\mathcal{OIP}$ の課題

OIPJOIN のアルゴリズムは，$\mathcal{L}_r$ の各パーティション c_r と結合可能なパーティション x を $\mathcal{L}_s$ から探しだす．そして，タプル $r \in c_r$ と交わりを持つタプル $s \in x$ があればそれらを結合する．結合の条件が 2 つの時区間 A と B の交わり $A \cap B = true$ であれば，このアルゴリズムは，重複なく，かつ漏れなく結合可能なパーティションの対を探し探し出すことができるが，Allen の時区間関係 [5] の 1 つである *contains* を条件とする結合演算に適しているとはいえない．その具体例を次に示す．

例 6. 時区間関係 $contains(A, B)$ の条件を満たすタプル A と B の結合を取るときの例を示す．ここで $contains(A, B)$ は $A.T_S < B.T_S \wedge B.T_E < A.T_E$ を満たす場合に真を返す 1 階述語である．時間範囲 U = [2012-1, 2012-12] であるリレーション $\mathbf{r}$ と $\mathbf{s}$，および $k = 4$ を入力とする．このときの Lazy Partition List $\mathcal{L}_s$ は図 2 となる．この時区間関係に関して $\mathcal{L}_r$ のパーティション $p_{1,1}$ に含まれるタプル A と結合可能なタプル B を含む $\mathcal{L}_s$ のパーティションは $p_{1,1}$ である．この節点は，$\mathcal{L}_s$ の根ではないので，結合可能なタプルを含まないパーティションに相当する $\mathcal{L}_s$ の節点を探索する必要がある．したがって，Algorithm 1 に示される既存手法では，非効率である． □

このように，既存手法では各リレーションのパーティションを木構造で保持しているため，$contains(A, B)$ を満たす結合演算に対して必要最低限のパーティションのみをたどることができない．Allen の時区間関係は 13 個の関係で構成されているが，そのうち 12 個に対しても同様である．

また，既存手法では $\mathbf{r}$ と k により Configuration が決まるため，結合対象となるパーティションの数が増えて効率的な演算ができない以下のような場合がある．

例 7. 時間範囲 [2012-1, 2012-12] であるリレーション $\mathbf{r}$ と，[2012-2, 2012-12] であるリレーション $\mathbf{s}$ を入力とする．また，各 $\mathcal{OIP}$ Configuration を $(\mathbf{r}, 3)$ と $(\mathbf{s}, 3)$ とする．このとき $\mathbf{r}$ と $\mathbf{s}$ のパーティションの一部とタプルの一部を図 4 に示す．$\mathbf{r}$ のパーティション $p_{1,1}$ に存在するタプル r と，$contains(A, B)$ の関係にあるタプル s が格納されている可能性がある $\mathbf{s}$ のパーティションは，$p_{0,0}$ と $p_{1,1}$ である．よって，2 回結合演算をとる必要がある． □

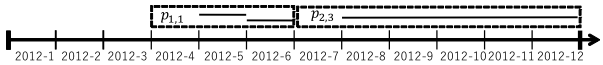

図 3 Configuration $(\mathbf{r}, 4)$ に対する $\mathcal{OIP}$
Fig. 3 $\mathcal{OIP}$ with Configuration $(\mathbf{r}, 4)$.

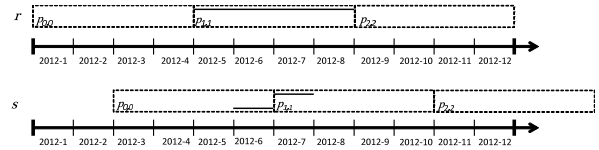

図 4 $contains(A, B)$ の例
Fig. 4 Example of $contains(A, B)$.

そこで本研究では，これらの課題を解決するため，Allen の時区間関係を条件とする結合演算に対して，効率的な計算を可能にするデータ構造とそれに対するアルゴリズムを提案する．

4. 提案手法

4.1 Partition Array

本稿で扱う Allen の 13 個の時区間関係を**表 1** に示す．この 13 個の関係の集合を $\Theta = \{before, meets, overlaps, during, starts, after, met\text{-}by, overlapped\text{-}by, finishes, equal, finished\text{-}by, started\text{-}by, contains\}$ とする．本稿が対象とする問題は，2 つのリレーション $\mathbf{r}$ と $\mathbf{s}$，および関係 $\theta \in \Theta$ が入力として与えられたとき，$\{(r, s) \mid r \in \mathbf{r} \wedge s \in \mathbf{s} \wedge \theta(r.T, s.T) = true\}$ を効率良く求めることである．

前節で述べた課題を克服するために，パーティションを構成するための設定を改良する．リレーション $\mathbf{r}$ に対する $\mathcal{OIP}$ Configuration は，$(\mathbf{r}, k)$ で表される．これに対して，提案手法では，$(\mathbf{r}, d, o)$ を入力として与える．ここで d はパーティションの最小構成時区間の長さであり，o はある時刻である．パーティションへの分割数は $k = \lceil \frac{U_E - o}{d} \rceil$ で決められる．ここで $\mathbf{r}$ の時間範囲 U の始点 U_S と o は，必ずしも一致するとは限らないことに注意されたい．提案手法では，2 つのリレーション $\mathbf{r}$ と $\mathbf{s}$ が与えられたとき，各リレーションに対して $(\mathbf{r}, d, o)$ と $(\mathbf{s}, d, o)$ によりパーティションの最小構成時区間を構成する．ここで，2 つのリレーションに対する d，および o は同一の値が用いられる．これにより，$\mathbf{r}$ のあるパーティション $p^r_{i,j}$ と $\mathbf{r}$ のあるパーティション $p^s_{i,j}$ は等しくなるので，前節で述べた 2 つ目の課題

表 1 Allen の時区間関係
Table 1 Allen's temporal relations.

Allen の時区間関係	2 つの時区間の始点，終点の条件
$before(A, B)$	$A.T_E < B.T_S$
$meets(A, B)$	$A.T_E = B.T_S$
$overlaps(A, B)$	$A.T_S < B.T_S < A.T_E < B.T_E$
$during(A, B)$	$B.T_S < A.T_S \wedge A.T_E < B.T_E$
$starts(A, B)$	$A.T_S = B.T_S \wedge A.T_E < B.T_E$
$after(A, B)$	$B.T_E < A.T_S$
$met\text{-}by(A, B)$	$A.T_S = B.T_E$
$overlapped\text{-}by(A, B)$	$B.T_S < A.T_S < B.T_E < A.T_E$
$finishes(A, B)$	$B.T_S < A.T_S \wedge A.T_E = B.T_E$
$equal(A, B)$	$A.T_S = B.T_S \wedge A.T_E = B.T_E$
$finished\text{-}by(A, B)$	$A.T_S < B.T_S \wedge A.T_E = B.T_E$
$started\text{-}by(A, B)$	$A.T_S = B.T_S \wedge B.T_E < A.T_E$
$contains(A, B)$	$A.T_S < B.T_S \wedge B.T_E < A.T_E$

© 2017 Information Processing Society of Japan

表 2 $q_{i',j'}$ と結合演算が可能なパーティションの集合
Table 2 a set of partitions which can join with $q_{i',j'}$.

θ	$S(\mathbf{r}, q_{i',j'}, \theta)$
before	$\{p_{i,j} \in P(\mathbf{r}) \mid j' \leq i \leq j \leq k-1\}$
meets	$\{p_{i,j} \in P(\mathbf{r}) \mid i = j' \leq j \leq k-1\}$
overlaps	$\{p_{i,j} \in P(\mathbf{r}) \mid i' \leq i \leq j' \leq j \leq k-1\}$
during	$\{p_{i,j} \in P(\mathbf{r}) \mid 0 \leq i \leq i' \wedge j' \leq j \leq k-1\}$
starts	$\{p_{i,j} \in P(\mathbf{r}) \mid i = i' \wedge j' \leq j \leq k-1\}$
after	$\{p_{i,j} \in P(\mathbf{r}) \mid 0 \leq i \leq j \leq i'\}$
met-by	$\{p_{i,j} \in P(\mathbf{r}) \mid 0 \leq i \leq i' = j\}$
overlapped-by	$\{p_{i,j} \in P(\mathbf{r}) \mid 0 \leq i \leq i' \leq j \leq j'\}$
finishes	$\{p_{i,j} \in P(\mathbf{r}) \mid 0 \leq i \leq i' \wedge j = j'\}$
equal	$\{p_{i,j} \in P(\mathbf{r}) \mid i = i' \wedge j = j'\}$
finished-by	$\{p_{i,j} \in P(\mathbf{r}) \mid i' \leq i \leq j' = j\}$
started-by	$\{p_{i,j} \in P(\mathbf{r}) \mid i = i' \leq j \leq j'\}$
contains	$\{p_{i,j} \in P(\mathbf{r}) \mid i' \leq i \leq j \leq j'\}$

を克服することができる．

$P(\mathbf{r})$ を $\mathbf{r}$ のパーティションの集合とする．i' 番目から j' 番目のパーティションにまたがるパーティション $q_{i',j'}$ と時区間関係 $\theta \in \Theta$ が与えられたとき，$\theta(q_{i',j'}, p_{i,j})$ を満たす $\mathbf{r}$ のパーティションの集合を $S(\mathbf{r}, q_{i',j'}, \theta)$ とする．すなわち

$$S(\mathbf{r}, q_{i',j'}, \theta) = \{p_{i,j} \in P(\mathbf{r}) \mid \theta(q_{i',j'}, p_{i,j}) = true\} \quad (1)$$

である．$\theta = equal$ のとき，式 (1) は

$$S(\mathbf{r}, q_{i',j'}, equal) = \{p_{i,j} \in P(\mathbf{r}) \mid i = i' \wedge j = j'\}$$

となる．これは θ として $equal$ が指定されたとき，$q_{i',j'}$ に含まれるタプルと結合可能なタプルを含むパーティションが $p_{i',j'}$ のみであることを意味している．同様にして，Θ の各要素に対して，$S(\mathbf{r}, q_{i',j'}, \theta)$ をまとめたものが**表 2** である．ここで，すべての $\theta \in \Theta$ に対して $S(\mathbf{r}, q_{i',j'}, \theta)$ が o や d に依存しないことに注意されたい．これに対して，既存手法では，補題 2.4 に示されたように，クエリ時区間と結合可能なタプルを含むパーティションを求めるには，パーティションの最小構成時区間の長さと $\mathbf{r}$ の時間範囲の始点が必要になる．

図 5 は $S(\mathbf{r}, q_{i',j'}, \theta)$ が返すパーティション $p_{i,j}$ に相当する点 (i, j) の集合が存在する領域を図示したものである．いずれの領域も連続する 1 つの矩形領域，あるいは三角形領域である．この領域の点 (i, j) を効率良く返すために，**図 6** に示す Partition Array を用いる．図 6 は $k = 4$ の場合の Partition Array である．Partition Array は 2 次配列であり，その (i, j) 要素がパーティション $p_{i,j}$ に相当する．全要素のうち，タプルが存在するパーティションに相当する要素は青色で描かれている．また，パーティション $p_{i,j}$ に対して $i \leq j$ を満たすので，この図の右下三角領域は空となる．したがって，これらの空のパーティションに対してメモリを確保する必要はない．また，それ以外の空のパーティションをたどることなく，$S(\mathbf{r}, q_{i',j'}, \theta)$ を求めるために，各要素は down と right のリンクを持つ．また，$p_{0,j}$ や $p_{i,k-1}$ が空の場合もあるため，H_Index と V_Index というリンクの配列を持ち，各行，各列の要素のうち，空でない先頭の要素に対するリンクを保持する．Lazy Partition List と Partition Array が必要とするメモリ量の違いは，後者のほうが (1) H_Index と V_Index を必要とすること，(2) 空のパーティションに対して，メモリを必要とすることの 2 点である．Lazy Partition List と Partition Array とで，down が指すパーティションが異なるという構造上の違いはあるものの，2 つの手法に対する k が等しいとき down と right に対して必要とされるメモリ量は同じである．2 つの手法が必要とするメモリ量の違いは，(1) に対し

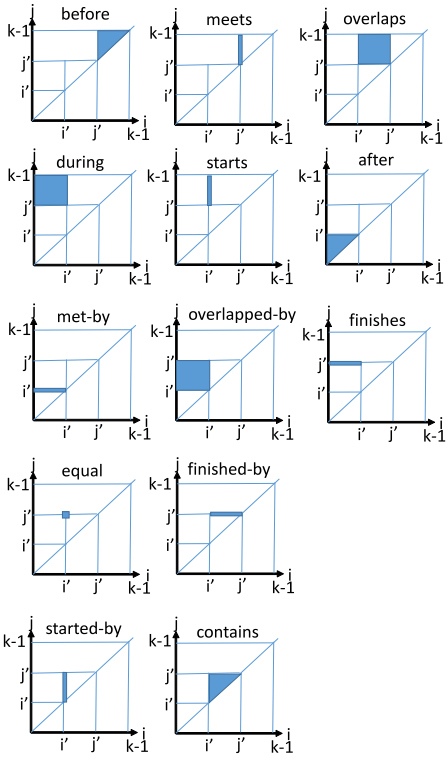

図 5 $q_{i',j'}$ と結合演算が可能な領域
Fig. 5 Area for partitions which can join with $q_{i',j'}$.

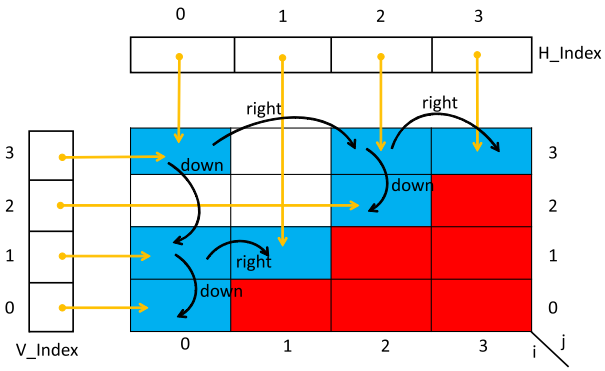

図 6 Partition Array の例
Fig. 6 Example of Partition Array.

Algorithm 2: Proposed Method

Data: Partition Arrays $\mathcal{A}_r$ and $\mathcal{A}_s$, relations $\mathbf{r}$ and $\mathbf{s}$, Allen's relation θ
Result: $\mathbf{z} = \{(r,s) \mid r \in \mathbf{r} \wedge s \in \mathbf{s} \wedge \theta(r.T, s.T) = true\}$

```
1 z := ∅;
2 for Partition q in A_r do
3     S := S(A_s, q, θ) ;
4     for Partition p in S do
5         z := z ∪ { joined tuples from q and p};
6 return z;
```

Algorithm 3: $S(\mathcal{A}_s, q_{i',j'}, before)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result: a set of Partitions S

```
1 S := ∅;
2 for i ∈ [j', k − 1] do
3     x := H_Index[i];
4     while x ≠ null ∧ x.j ≥ x.i do
5         S := S ∪ {x};
6         x := x.down;
7 return S;
```

Algorithm 4: $S(\mathcal{A}_s, q_{i',j'}, meets)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result: a set of Partitions S

```
1 S := ∅;
2 x := H_Index[j'];
3 while x ≠ null ∧ x.j ≥ j' do
4     S := S ∪ {x};
5     x := x.down;
6 return S;
```

て $16k$ バイト，(2) に対して最大 $2k(k+1)$ バイト[*2]が必要となるが，5 章の評価実験で述べるように，k は 200 程度が適切であり，主記憶に十分に格納できる量である．

4.2 結合アルゴリズム

Algorithm 2 は，Allen の時区間関係 θ と 2 つのリレーション $\mathbf{r}$ と $\mathbf{s}$ が与えられたとき，θ に従い結合演算を行うための擬似コードである．$\mathbf{r}$ の各パーティション q に対して，表 2 に従い，q と結合可能な $\mathbf{s}$ のパーティション集合 S を取得する．その後，q と S の各要素 p との結合をとり，その結果を $\mathbf{z}$ に追加する．

Algorithm 3 から Algorithm 12 は各 $\theta \in \Theta$ に対して

[*2] 存在しうるパーティションの数は，$\frac{k(k+1)}{2}$ であり，空であるパーティションに対して 4 バイトが必要となる．すべてのパーティションが空ではないので，最大でも $2k(k+1)$ となる．

Algorithm 5: $S(\mathcal{A}_s, q_{i',j'}, overlaps)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result: a set of Partitions S

```
1 S := ∅;
2 for i ∈ [i', j'] do
3     x := H_Index[i];
4     while x ≠ null ∧ x.j ≥ j' do
5         S := S ∪ {x};
6         x := x.down;
7 return S;
```

Algorithm 6: $S(\mathcal{A}_s, q_{i',j'}, during)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result: a set of Partitions S

```
1 S := ∅;
2 for i ∈ [0, i'] do
3     x := H_Index[i];
4     while x ≠ null ∧ x.j ≥ j' do
5         S := S ∪ {x};
6         x := x.down;
7 return S;
```

Algorithm 7: $S(\mathcal{A}_s, q_{i',j'}, starts)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result: a set of Partitions S

```
1 S := ∅;
2 x := H_Index[i'];
3 while x ≠ null ∧ x.j ≥ j' do
4     S := S ∪ {x};
5     x := x.down;
6 return S;
```

Algorithm 8: $S(\mathcal{A}_s, q_{i',j'}, after)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result: a set of Partitions S

```
1 S := ∅;
2 for j ∈ [0, i'] do
3     x := V_Index[j];
4     while x ≠ null ∧ x.j ≥ x.i do
5         S := S ∪ {x};
6         x := x.right;
7 return S;
```

$S(\mathbf{r}, q_{i',j'}, \theta)$ を求めるための擬似コードである．これを互いに素な 3 つのグループに分ける．1 つ目は，図 5 の結合演算が可能な領域が $j = k-1$ と接している時

© 2017 Information Processing Society of Japan

Algorithm 9: $S(\mathcal{A}_s, q_{i',j'}, met\text{-}by)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result: a set of Partitions S

```
S := ∅;
x := V_Index[i'];
while x ≠ null ∧ x.i ≤ i' do
    S := S ∪ {x};
    x := x.right;
return S;
```

Algorithm 10: $S(\mathcal{A}_s, q_{i',j'}, overlapped\text{-}by)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result: a set of Partitions S

```
S := ∅;
for j ∈ [i', j'] do
    x := V_Index[j];
    while x ≠ null ∧ x.i ≤ i' do
        S := S ∪ {x};
        x := x.right;
return S;
```

Algorithm 11: $S(\mathcal{A}_s, q_{i',j'}, finishes)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result: a set of Partitions S

```
S := ∅;
x := V_Index[j'];
while x ≠ null ∧ x.i ≤ i' do
    S := S ∪ {x};
    x := x.right;
return S;
```

Algorithm 12: $S(\mathcal{A}_s, q_{i',j'}, equal)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result:

```
S := ∅;
if A_s[i'][j'] ≠ null then
    S := {A_s[i'][j']};
return S;
```

区間関係 $\Theta_1 = \{before, meets, overlaps, during, starts\}$ である．2 つ目は，Θ_1 以外の時区間関係で，結合演算が可能な領域が $i = 0$ に接している時区間関係 $\Theta_2 = \{after, met\text{-}by, overlapped\text{-}by, finishes\}$ である．3 つ目は，$\Theta_1 \cup \Theta_2$ 以外の時区間関係 $\Theta_3 = \{equal, finished\text{-}by, started\text{-}by, contains\}$ である．グループ 1 の領域は $j = k - 1$ に接するため，H_Index を利用することができる．

Algorithm 13: $S(\mathcal{A}_s, q_{i',j'}, finished\text{-}by)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result: a set of Partitions S

```
S := ∅;
for i ∈ [i', j'] do
    if A_s[i][j'] ≠ null then
        x := A_s[i][j'] ;
        break ;
while x ≠ null ∧ x.i ≤ j' do
    S := S ∪ {x};
    x := x.right;
return S;
```

Algorithm 14: $S(\mathcal{A}_s, q_{i',j'}, started\text{-}by)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result: a set of Partitions S

```
S := ∅;
for j ∈ [i', j'] do
    if A_s[i'][j] ≠ null then
        x := A_s[i'][j] ;
        break ;
while x ≠ null ∧ x.j ≥ i' do
    S := S ∪ {x};
    x := x.down;
return S;
```

Algorithm 15: $S(\mathcal{A}_s, q_{i',j'}, contains)$

Data: Partition array $\mathcal{A}_s$, query Partition $q_{i',j'}$, and Allen's relation
Result: a set of Partitions S

```
S := ∅;
for i ∈ [i', j'] do
    for j ∈ [i', j'] do
        if A_s[i][j] ≠ null then
            x := A_s[i][j] ;
            break ;
    while x ≠ null ∧ x.i ≤ x.j do
        S := S ∪ {x};
        x := x.down;
return S;
```

したがって，Algorithm 3 から Algorithm 7 は H_Index から適切な要素を取得し，各要素の down をたどることで，$q_{i',j'}$ と結合可能なパーティションの集合を取得することができる．グループ 2 の領域は $i = 0$ に接するため，V_Index を利用することができる．したがって，Algorithm 8 から Algorithm 11 は V_Index から適切な要素を取得し，各要

© 2017 Information Processing Society of Japan

素の right をたどることで，$q_{i',j'}$ と結合可能なパーティションの集合を取得することができる．これらのアルゴリズムは，空のパーティションにアクセスすることはなく，かつ $q_{i',j'}$ と結合できないパーティションにアクセスすることはないため，効率が良い．一方，3 つ目のグループの領域は $j = k-1$ にも $i = 0$ にも接しない．よって，2 次配列のインデックスを利用し，空でないパーティションを探索する．空でないパーティションに到達したら，right か down をたどり，$q_{i',j'}$ と結合可能なパーティションの集合を取得する．これらのアルゴリズムは，空のパーティションにアクセスすることはあるが，$q_{i',j'}$ と結合できないパーティションにアクセスすることはないため，効率が良い[*3]．$S(\mathbf{r}, q_{i',j'}, \theta)$ が返すパーティション $p_{i,j}$ に相当する点 (i,j) の集合が存在する領域は，いずれも 1 つの矩形，あるいは三角形であるので，複雑な条件を含まない二重ループ文で表現できるために，実装が容易である．

ここで本研究と既存研究 [4] の関係を述べる．文献 [4] で扱われた問題は，$A \cap B \neq \emptyset$ である時区間 A と B を扱うものであった．これは，Allen の時区間関係を用いることで，$meets(A,B) \vee overlaps(A,B) \vee during(A,B) \vee starts(A,B) \vee met\text{-}by(A,B) \vee overlapped\text{-}by(A,B) \vee finishes(A,B) \vee equal(A,B) \vee finished\text{-}by(A,B) \vee started\text{-}by(A,B) \vee contains(A,B)$ を満たす時区間 A と B を扱う問題と表現することができる．$\Theta_s = \{meets, overlaps, during, starts, met\text{-}by, overlapped\text{-}by, finishes, equal, finished\text{-}by, started\text{-}by, contains\}$ とすると，

$$S(\mathbf{r}, q_{i',j'}, \Theta_s) = \{p_{i,j} \in P(r) \mid \bigvee_{\theta \in \Theta_s} \theta(q_{i',j'}, p_{i,j})\} = \bigcup_{\theta \in \Theta_s} S(\mathbf{r}, q_{i',j'}, \theta)$$

と表すことができる．

5. 評価実験

5.1 評価実験の設定

本研究で用いた実験データについて述べる．実験データは，ID，START，END という 3 つの属性を持つタプルが格納されている $\mathbf{r}$ と $\mathbf{s}$ とする．ここで，ID，START，END はすべて整数型とする．START の値を一様分布で決定し，時区間の長さを平均 $\lambda = 100$ のポアソン分布で決定してタプルの時区間を作成し，IBM DB2 に格納した．既存手法 $\mathcal{OIP}$ と提案手法のアルゴリズムを Java で実装し，JDBC（Java Database Connectivity）を介して，DB2 にアクセスする．具体的には，Lazy Partition List や Partition Array は Java を用いたデータ構造に格納され，Algorithm 2 の 5 行目において結合された時区間の対の集合を DB2 から得る．計算時間は，$\mathcal{OIP}$ の場合，Lazy Partition List $\mathcal{L}$ をたどり始めてから結果を得るまでの時間とし，提案手法の場合は，Partition Array $\mathcal{A}$ をたどり始めてから結果を得るまでの時間とする．

本実験で使用した計算機の仕様は，プロセッサが 3.5 GHz の Intel (R) Xeon (R)，主記憶が 72 GB，OS が Windows 8.1 Professional（64 bit）である Workstation である．

[*3] Algorithm 12 の 2 行目で，j には $[i', j']$ の整数が降順に代入される．

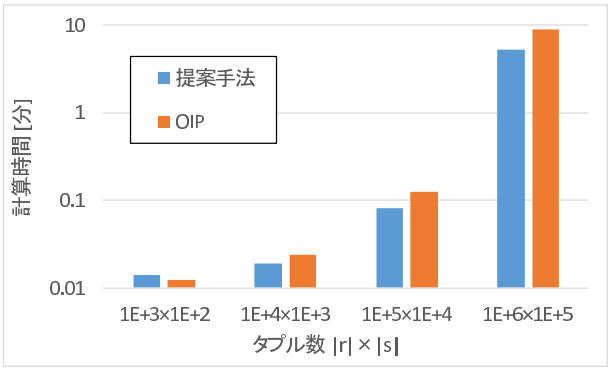

図 7 タプル数を変化させたときの計算時間（$overlaps(A,B)$）
Fig. 7 Computation time for changing the number of tuples ($overlaps(A,B)$).

表 3 人工データの時間範囲
Table 3 Time ranges for artificial datasets.

	$\mathbf{r}$ の時間範囲	$\mathbf{s}$ の時間範囲
データ 1	$[0, 2^{10}-1]$	$[2^5-1, 2^{10}-1]$
データ 2	$[0, 2^{15}-1]$	$[2^5-1, 2^{15}-1]$
データ 3	$[0, 2^{20}-1]$	$[2^5-1, 2^{20}-1]$

5.2 タプル数を変化させたときの結合に要する計算時間

はじめに $\mathbf{r}$ と $\mathbf{s}$ に格納されるタプル数を変化させて計算時間を測定した．$\mathbf{r}$ と $\mathbf{s}$ の時区間はそれぞれ時間範囲 $[0, 2^{15}-1]$ と $[2^5-1, 2^{15}-1]$ に収まるようにランダムに選ばれた．この実験では，最も時間を要する $overlaps(A,B)$ を結合の条件とし，$\mathcal{OIP}$ の最小構成時区間数 $k = 20$ とし，提案手法の最小構成時区間数が既存手法と同じになるように $d = 1639$ とした．その結果を**図 7** に掲載する．計算時間はタプル数の積に比例して大きくなる．タプル数が 100,000 × 10,000 以下であれば，計算時間は 5 秒程度であるが，1,000,000 × 100,000 になると 5 分から 10 分程度の計算時間を要する．ただし，これはパーティションの最小構成時区間数やその長さを調整しなかった場合の結果であり，適切にこれらの値を調整することで，計算時間を小さくできることを次節で示す．

5.3 結合条件をを変化させたときの結合に要する計算時間

$\mathbf{r}$ と $\mathbf{s}$ の時間範囲が異なる 3 種の人工データを**表 3** の時間範囲に従って作成した．ここで $\mathbf{r}$ と $\mathbf{s}$ の時間範囲の始点が異なるように設定している．**図 8**，**図 9**，

© 2017 Information Processing Society of Japan

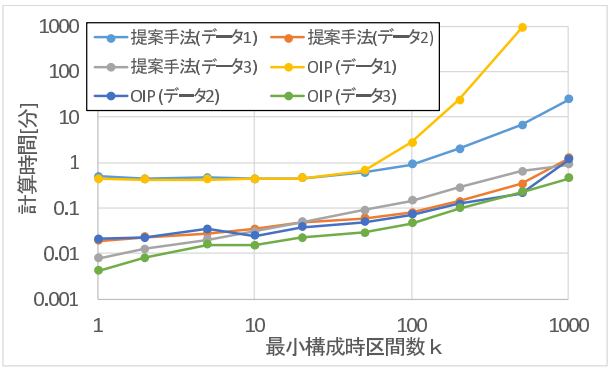

図 8 結合条件 $meets(A, B)$ の場合の計算時間
Fig. 8 Computation time for the join condition $meets(A, B)$.

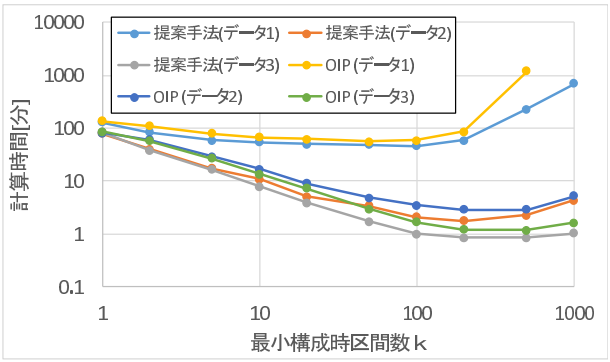

図 9 結合条件 $overlaps(A, B)$ の場合の計算時間
Fig. 9 Computation time for the join condition $overlaps(A, B)$.

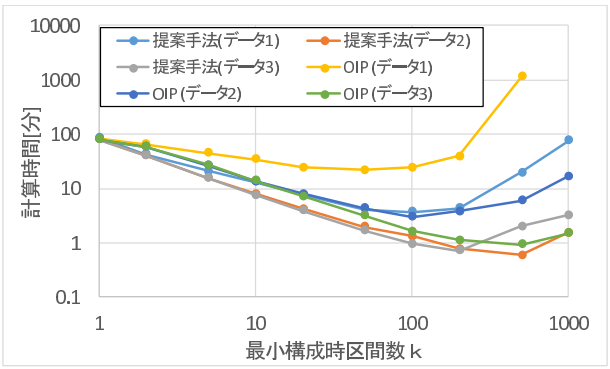

図 10 結合条件 $during(A, B)$ の場合の計算時間
Fig. 10 Computation time for the join condition $during(A, B)$.

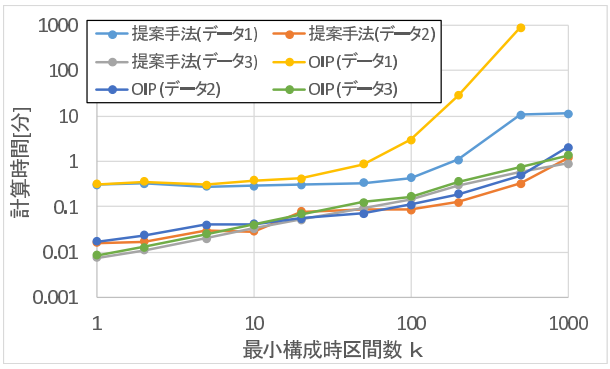

図 11 結合条件 $starts(A, B)$ の場合の計算時間
Fig. 11 Computation time for the join condition $starts(A, B)$.

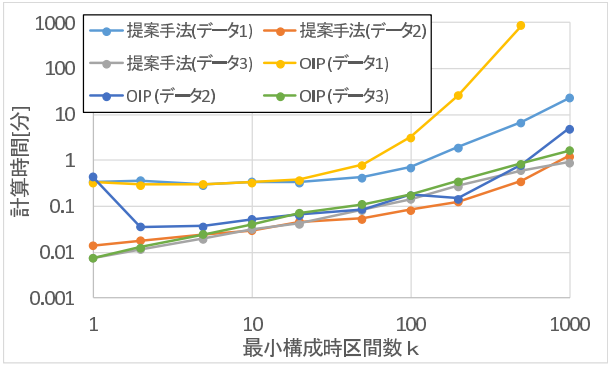

図 12 結合条件 $finishes(A, B)$ の場合の計算時間
Fig. 12 Computation time for the join condition $finishes(A, B)$.

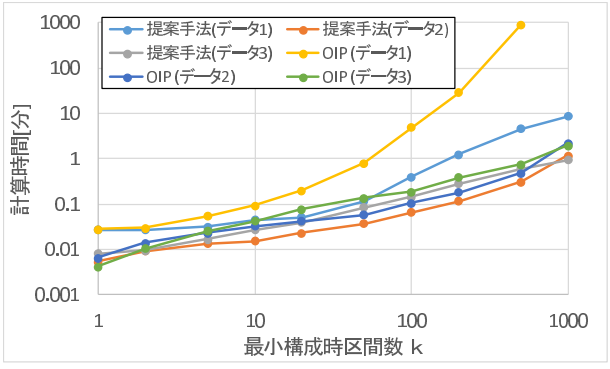

図 13 結合条件 $equals(A, B)$ の場合の計算時間
Fig. 13 Computation time for the join condition $equals(A, B)$.

図 10，**図 11**，**図 12**，**図 13** にパーティションの最小構成時区間数やその長さを変えたときの計算時間を示す．ここで，$meets(A, B)$ と $met\text{-}by(A, B)$，$overlaps(A, B)$ と $overlapped\text{-}by(A, B)$，$during(A, B)$ と $contains(A, B)$，$starts(A, B)$ と $started\text{-}by(A, B)$，$finishes(A, B)$ と $finished\text{-}by(A, B)$ について，対の関係にある時区間関係の計算時間の傾向はほぼ同じであった．よって，$meets(A, B)$，$overlaps(A, B)$，$during(A, B)$，$starts(A, B)$，$equal(A, B)$，$finishes(A, B)$，の実験結果のみを示す．また，**図 14** は，結合の条件 $meets$, $overlaps$, $during$, $starts$, $met\text{-}by$, $overlapped\text{-}by$, $finishes$, $equal$, $finished\text{-}by$, $started\text{-}by$, $contains$ が等確率で使われるものとして，これら 11 個の実験における計算時間の平均をとったものである．最小構成時区間数を $k = 1$ から大きくすると計算時間は減少する．これは，k が大きくなると，各タプルがそれを覆うパーティションにタイトに囲まれるので，結合の条件に合致しない 2 つのタプルの結合を回避できる，つまり false hit を回避できるからである．さらに最小構成時区間数を大きくすると計算時間は増加する．

© 2017 Information Processing Society of Japan

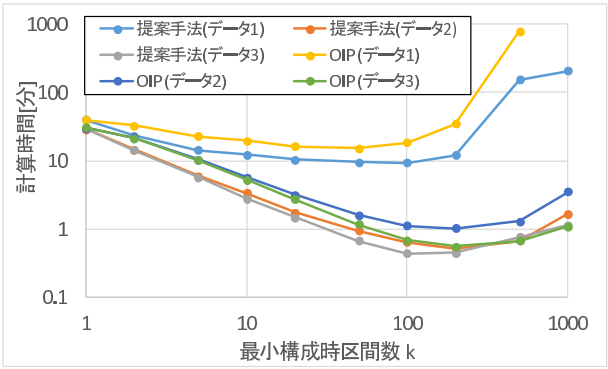

図 14 11 種の結合条件に対する計算時間の平均
Fig. 14 Average computation time for the 11 join conditions.

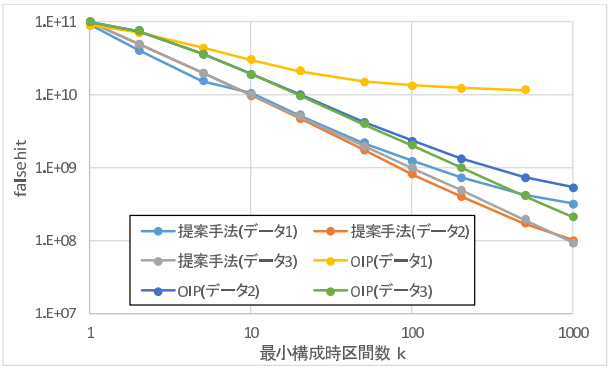

図 15 False Hit
Fig. 15 False Hit.

これは，最小構成時区間数が必要以上に多くなると **r** や **s** に格納されているリレーションが過度に細分化され，パーティションどうしの結合の回数が増えるからである．

続いて，上記で述べた false hit やパーティションどうしの結合について詳細に調査した．**図 15** と**図 17** は，それぞれ，結合の条件を $overlap(A, B)$ としたときの false hit とパーティションの結合回数を表している．他の結合条件でも実験したが同様の結果であり，紙面の制約上，この条件の結果のみを示す．k が大きくなると，各タプルがそれを覆うパーティションにタイトに囲まれるので，false hit を回避できる．この傾向は図 15 から読み取れる．また，2 つの手法の false hit の差を**図 16** に示す．提案手法の false hit は，$\mathcal{OIP}$ よりも非常に小さい．一方，k が大きくなると，リレーションが過度に細分化されるので，パーティションどうしの結合回数が増える．この傾向は図 17 から読み取れる．また，2 つの手法のパーティションの結合回数の差を**図 18** に示す．提案手法の結合回数も，$\mathcal{OIP}$ よりも非常に小さい．

図 14 に対して，各データごとに最小の計算時間をまとめたものが**表 4** である．この結果より，パーテイションの最小構成時区間数やその長さを適切に調整することにより，提案手法は既存手法 $\mathcal{OIP}$ よりも約 1.25 倍から 2 倍高速に動作する．Lazy Partition List $\mathcal{L}$ は，各リレーショ

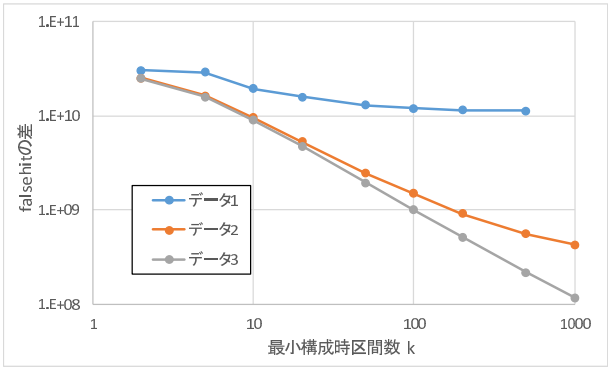

図 16 False Hit の差
Fig. 16 Difference of False Hit.

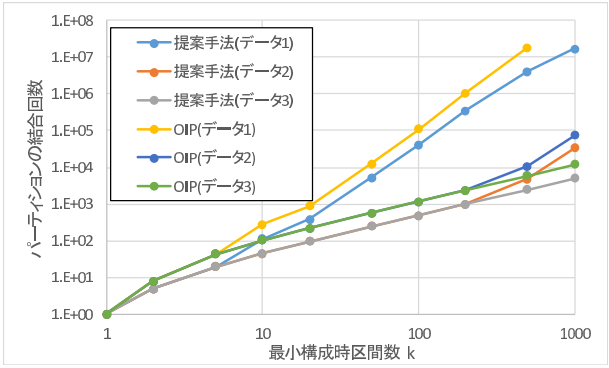

図 17 パーティションの結合回数
Fig. 17 Numbers of times for joining partitions.

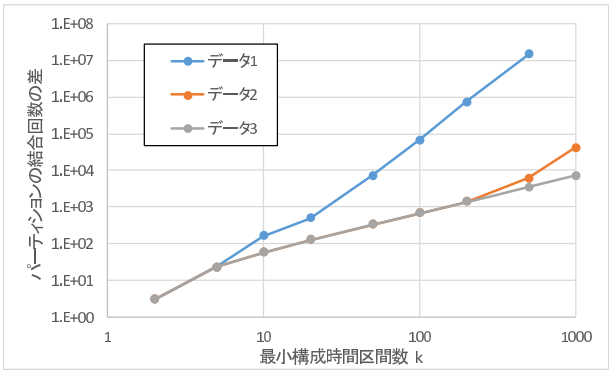

図 18 パーティションの結合回数の差
Fig. 18 Difference of numbers of times for joining partitions.

表 4 各人工データに対する計算時間のまとめ
Table 4 Summary of computation times for each artificial dataset.

	提案手法の平均計算時間 A［分］	$\mathcal{OIP}$ の平均計算時間 B［分］	高速化 B/A
データ 1	9.37	15.10	1.61
データ 2	0.52	1.02	1.96
データ 3	0.38	0.56	1.25

ンのパーティションを木構造で保持しているため，つねに根から探索をする必要がある．よって，必要最低限のパーティションのみをたどり，結合演算をすることができない．

© 2017 Information Processing Society of Japan

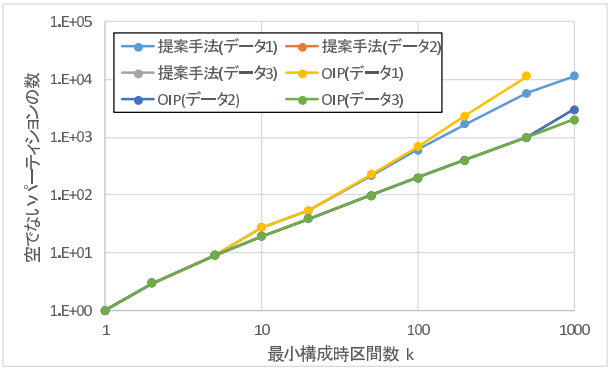

図 19 s に対する空でないパーティションの数
Fig. 19 Number of non-empty partitions for **s**.

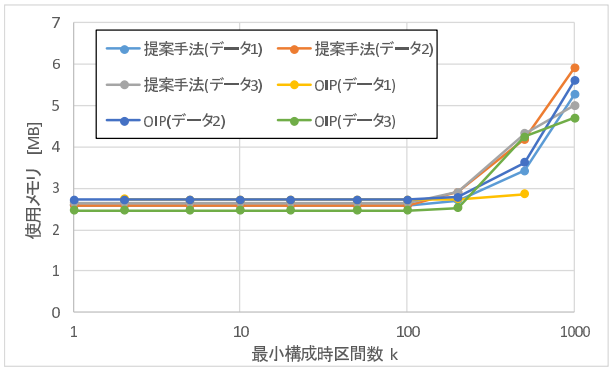

図 20 Java プログラムのメモリ使用量
Fig. 20 Memory usage of the Java program.

しかし，Partition Array $\mathcal{A}$ は 2 次配列型のデータ構造であるため，指定された時区間関係ごとに結合対象とするパーティションを選択することで，必要最低限のパーティションのみをたどり，結合演算をすることができる．よって，既存手法よりも効率的な結合演算が可能になる．さらに，提案手法では共通の最小構成時区間の長さ d を用いてパーティションを作成している．このため，2 つの時区間が交わりを持つ時区間関係となる $meets(A,B)$，$met\text{-}by(A,B)$，$overlaps(A,B)$，$overlapped\text{-}by(A,B)$，$finishes(A,B)$，$finished\text{-}by(A,B)$，$starts(A,B)$，$started\text{-}by(A,B)$，$contains(A,B)$，$during(A,B)$，$equal(A,B)$ の関係にある時区間の対を抽出するときには，既存手法よりも結合回数を減らすことができる．これらのことから，Partition Array を使う効果が得られていることが分かる．

また，データ 1，2，3 に対する計算時間を比較すると，データ 1 に対する計算時間が最も大きい．これは，**r** と **s** の時間範囲が小さいため，相対的に各タプルの時区間の長さが大きくなる．このため，タプルの多くが最小構成時区間のパーティションに格納されるのではなく，長いパーティションに格納される．よって，空でないパーティションが増えることにより，結合されるパーティションの数が増え，計算時間も増加する．

5.4 索引に関する評価実験

図 19 は **s** に対するパーティションのうち，空でないものの数を表している．最小構成時区間数 k に対する，存在しうるパーティションの数は $\frac{k(k+1)}{2}$ 個あるが，平均 λ に従うポアソン分布で時区間長を決めたので，極端に長いパーティションは存在せず，k^2 に比例するほどは増加しない．また，図 20 は **s** に対する Lazy Partition List や Partition Array をメモリ上に格納し，結合演算を実施する直前の Java プログラムのメモリの使用量の測定結果である．Partition Array が空なパーティションに対して 4 バイト必要とするのに対して，Lazy Partition List はそれに対してメモリを割り当てないので，後者のほうが必要とするメモリ量は少ない．いずれの手法も数 MB であり，主記憶に十分に格納できる量である．

6. 議論

本稿では，時区間を含むタプルからリレーションの結合に関する手法を提案したが，提案手法に関わる関連技術に関して議論する．四分木（Quadtree）[6], [8] は時間範囲を再帰的に 2 分割しながら，パーティションを構成し，木構造の索引を構築する．たとえば，時間範囲が $[1,32]$ [*4]の場合，$[1,32]$ を根のパーティション，$[1,16]$ を根の左の子のパーティション，$[17,32]$ を根の右の子のパーティションとする．各タプルは，$\mathcal{OIP}$ 同様に，それを覆う最小のパーティションに割り当てられる．このため，タプル $[16,17]$ というタプルが存在したとき，このタプルは根のパテーションに割り当てられるが，割り当てられたパーティション $p=[p_S,p_E]$ のうち時区間 $x=[x_S,x_E]$ でない時刻 $\{x \mid t \in p, t \notin x\}$ が大きいため，false hit の可能性が大きく上がる．これに対し，根のパーティション $[1,32]$ を，たとえば，$[1,17]$ と $[16,32]$ に分割する方法が，緩和四分木（Loose Quadtree）[3], [9] である．この緩和四分木ではタプル $[16,17]$ は根よりも深い節点のパーティションに割り当てられるために，四分木に比べ false hit の可能性が下がるが，パーティションの数が増え，木が深くなる傾向がある．

セグメント木（Segment Tree）[2] は，四分木と同様に，時間範囲を再帰的に 2 分割しながら，パーティションを構成し，木構造の索引を構築する．各タプル x は，$p \subseteq x$ を満たすパーティションに対応する節点の中で，その親が $p \subseteq x$ を満たさない節点に対応付けられる．各タプルは複数の節点に関連付けられるので，特に長い時区間を持つタプルから構成されるリレーションに対して，索引が $\mathcal{OIP}$ よりも大きくなり，それを探索するコストも大きくなる．

関係区間木（Relational Interval Tree）[1], [7] は，Edelsbrunner の区間木を RDB で実装したものであり，リレーションに格納されたタプルの時区間の始点と終点のそれぞ

*4 文献番号と時区間の記号が類似しているので，注意されたい．

© 2017 Information Processing Society of Japan

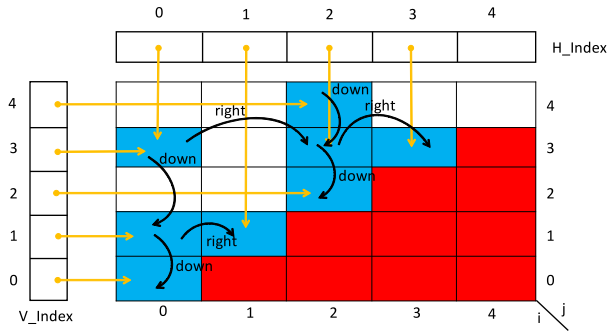

図 21 図 6 の Partition Array へのパーティション $p_{2,4}$ の追加
Fig. 21 Insertion of the partition $p_{2,4}$ to the Partition Array shown in Fig. 6.

れに対して構築された B$^+$ 木を用いて，結合演算の効率化を図っている．セグメント木同様に，結合の際には，木の多くの節点が結合に使われるので，長い時区間を持つタプルから構成されるリレーションに対して効率が劣化する．

文献 [4] では，これらの手法が，長いタプルに対して結合の効率が下がることが評価実験においてい実証されている．これに対して，データ 1 を用いた評価実験で示したように，提案手法は時間範囲に対して相対的に長いタプルを持つデータに対して，$\mathcal{OIP}$ よりも効率的である．また，上記の何れの手法も，Allen の時区間関係を条件とする結合演算に関する議論はなく，本稿はそれを実証した点において新規性があるといえる．

次に，時区間が挿入，削除，更新されることにより起こる Partition Array の更新について議論する．その前提として，追加される時区間や更新後の時区間の始点は，Partition Array を構築する際に設定される o 以上であることを想定する．つまり，$\min_{r\in\mathbf{r}} r.T_S < o$ となることは想定しない．あるいは，つねに $\min_{r\in\mathbf{r}} r.T_S \geq o$ となるように o が十分に小さく設定されているものとする．時区間 T がリレーションに追加されるとき，T が属するパーティション $p^T_{i,j}$ が空でない場合と，空の場合で処理が異なる．$p^T_{i,j}$ が空でない場合，Partition Array は更新されず，T を $p^T_{i,j}$ に追加するのみでよい．一方，$p^T_{i,j}$ が空の場合，Partition Array を更新する必要がある．時区間を追加することにより，リレーションの時間範囲の終点 U_E が変わることがありうるが，提案手法では，パーティションの始点や終点は，o と d のみから決まり，U_E から決まらない．このため，時区間を追加することによりパーティションの始点や終点が変わることはないため，Partition Array 全体の再構築は必要なく，**図 21** に示されるように $p^T_{i,j}$ からのポインタの張り替えと H_Index と H_Index の伸張のみで実現できる．一方，$\mathcal{OIP}$ のパーティションの始点や終点は，U_E に依存する．このため，時区間の追加により U_E が変わると，Partition List 全体の再構築が必要となる．

次に，時区間の削除について，議論する．時区間 T を削除することにより，それが属するパーティションが空にならない場合，単にその時区間をリレーションから除くだけである．一方，区間 T を削除することにより，それが属するパーティションが空になる場合は，$p^T_{i,j}$ からのポインタを削除し，$p^T_{i,j}$ へのポインタを更新する必要がある．最後に，時区間の更新について，議論する．更新の前後で時区間 T が属するパーティションが変わらなければ，単に T の始点か終点を更新するのみでよい．一方，更新の前後で時区間 T が属するパーティションが変わる場合は，T を削除し，更新された T を挿入することで実現できる．

最後に，結合演算のコストの見積りに関して議論する．一般に，索引を用いない場合のネストループ結合のコストは，リレーション $\mathbf{r}$ のページ数を $P(\mathbf{r})$，結合に関する $\mathbf{r}$ のページ数を $P'(\mathbf{r})$ とすると，$P(\mathbf{r}) + P'(\mathbf{r}) \times P(\mathbf{s})$ で表される．これに対して，主キー以外の属性での非クラスタリング索引を利用する場合のコストは

$$P(\mathbf{r}) + P'(\mathbf{r}) \left(I + \frac{|\mathbf{s}|}{D_x(\mathbf{s})} \right)$$

で表される．ここで，I は索引をたどるコストであり，$D_x(\mathbf{s})$ はリレーション $\mathbf{s}$ の属性 x の値の種類数である．索引に B$^+$ 木を用いるとき，I はこの木の高さとなる．索引が Partition Array であり，結合の条件が $\Theta_1 \cup \Theta_2$ のいずれかの場合では，直接，結合対象とならない $\mathbf{s}$ のパーティションをたどることはないので，I は定数となる．一方，結合の条件が Θ_3 のいずれかの場合は，結合対象とならない $\mathbf{s}$ のパーティションをたどることはあるが，その上界は k となる．さらに，$D_x(\mathbf{s})$ は存在するパーティションの数となる．同様にして，ソートマージ結合やハッシュ結合の結合のコストを見積もることができるが，それらの見積りの検証については今後の課題としたい．

7. まとめ

本研究では，Allen の時区間関係に対する既存の演算手法である Overlap Interval Partitioning（$\mathcal{OIP}$）の課題について議論し，その課題を解決する手法として Partition Array を提案した．Partition Array は，2 次配列型のデータ構造を持ち，各要素をパーティションに対応させて結合演算を行う手法である．提案手法と $\mathcal{OIP}$ を実装し，計算時間の違いについて考察することにより，提案手法の有効性を評価・検証した．本研究で得られた成果についてまとめる．本稿の評価実験は，時区間データについて，$\mathcal{OIP}$ と提案手法 Partition Array の計算時間の違いについて比較を行った．提案手法では，必要最低限のパーティションのみを結合対象とし，効率的な結合演算ができた．また，共通の最小構成時区間 d を用いてパーティションを作成することで，2 つの時区間が交わりを持つ時区間関係にあるタプルを抽出するときには，$\mathcal{OIP}$ より結合回数を減らすことができた．よって，$\mathcal{OIP}$ より有用性があるといえる．

© 2017 Information Processing Society of Japan

参考文献

[1] Kriegel, H.-P., Pötke, M. and Seidl, T.: Managing Intervals Efficiently in Object-Relational Databases, *Proc. International Conference on Very Large Data Bases* (*VLDB*), pp.407–418 (2000).

[2] de Berg, M., Cheong, O., van Kreveld, M. and Overmars, M.: Chapter 10. More Geometric Data Structures, *Computational Geometry* (3rd Ed.), pp.219–241, Springer Berlin Heidelberg (2008).

[3] Samet, H., Sankaranarayanan, J. and Auerbach, M.: Indexing Methods for Moving Object Databases: Games and Other Applications, *Proc. International Conference on Management of Data* (*SIGMOD*), pp.169–180 (2013).

[4] Dignös, A., Böhlen, M.H. and Gamper, J.: Overlap Interval Partition Join, *Proc. International Conference on Management of Data* (*SIGMOD*), pp.1459–1470 (2014).

[5] Allen, J.F.: Maintaining Knowledge about Temporal Intervals, *Comm. ACM*, Vol.26, pp.832–843 (1983).

[6] Samet, H.: *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann Publishers Inc. (2005).

[7] Enderle, J., Hampel, M. and Seidl, T.: Joining Interval Data in Relational Databases, *Proc. International Conference on Management of Data* (*SIGMOD*), pp.683–694 (2004).

[8] Finkel, R.A. and Bentley, J.L.: Quad Trees: A Data Structure for Retrieval on Composite Keys, *Acta Informatica*, Vol.4, pp.1–9 (1979).

[9] DeLoura, M.: *Game Programming Gems*, Vol.1, Chap.4.11, pp.444–453, Charles River Media (2000).

山下 夏奈

1993 年生．2016 年関西学院大学理工学部情報科学科卒業．同年富士通テン（株）に入社．データマイニング，データベースの研究に興味を持つ．

猪口 明博 （正会員）

1975 年生．2000 年大阪大学大学院工学研究科通信工学専攻博士前期課程修了．同年日本アイ・ビー・エム（株）に入社，東京基礎研究所に配属．2004 年大阪大学大学院工学研究科通信工学専攻博士後期課程修了．博士（工学）．大阪大学産業科学研究所助教を経て，現在に至る．データマイニング，機械学習，テキストマイニング，データベースの研究・開発に従事．2002 年 Journal of Computer Aided Chemistry 論文賞，人工知能学会 2003 年度研究会（知識ベースシステム研究会）優秀賞受賞．

© 2017 Information Processing Society of Japan