# 配列 DBMS における時空間データの差分分析について

趙　セイ† 石川　佳治†† 河井　悠佑†† 杉浦　健人†

† 名古屋大学大学院情報科学研究科　〒 464–8601 愛知県名古屋市千種区不老町
†† 名古屋大学大学院情報学研究科　〒 464–8601 愛知県名古屋市千種区不老町
E-mail: †{zhao,yusuke,sugiura}@db.ss.is.nagoya-u.ac.jp, ††ishikawa@i.nagoya-u.ac.jp

**あらまし**　さまざまな分野においてビッグデータが注目されている今日では，データベースにおける大量のデータの高度な分析処理を行うためのデータアナリティクス（data analytics）について，研究が盛んに行われている．本稿では，時空間データの基本的な分析要求の一つとして考えられる差分（もしくは差異）（difference）に着目する．時空間的なデータでは，特に時間に関する変化をどのように検出するかが重要な要求の一つとして考えられる．このような背景に基づき，時空間データウェアハウスから差分・差異を検出するための差分演算について提案し，その実装に関するアプローチを示す．

**キーワード**　時空間データベース, データウェアハウス, 差分演算

## 1. はじめに

さまざまな分野においてビッグデータが注目されている今日では，データベースにおける大量のデータの高度な分析処理を行うための**データアナリティクス**（data analytics）について，研究が盛んに行われている [8]．**時空間データベース**（spatio-temporal database）においても，行動データ，移動軌跡データ，科学分野のデータなどのさまざまな領域において大規模な時空間データの分析が求められている．我々の研究グループでは，特に時空間的な大規模シミュレーションの結果を蓄積し，対話的な分析を可能とするための**時空間データウェアハウス**（spatio-temporal data warehouse）に関する研究を進めており，その考え方を**シミュレーションデータウェアハウス**（simulation data warehouse）と呼んでいる．特に津波・震災シミュレーションデータの分析に関して研究を進めている [6], [7]．

本稿では，時空間データウェアハウスにおける基本的な分析要求の一つとして考えられる**差分**（もしくは**差異**）（difference）に着目する．時空間的なデータでは，特に時間に関する変化をどのように検出するかが重要な要求の一つとして考えられる．また，パラメータが異なるシミュレーションデータや観測状況が異なる観測データが与えられたとき，異なるデータ間にどのような違いがあるかを検出することも求められる．例として，以下のような避難シミュレーションデータの分析の例を考える．

> 対象となるシミュレーションの空間領域において，地震が発生した 1 時間後 ($T_1$) と 4 時間後 ($T_2$) における避難者の数の分布について，顕著な差異がある領域を報告せよ．

この分析例では，時間帯 $T_1$ と $T_2$ において，避難者の数の分布にどのような違いがあるかを求めている．

差分演算のイメージを図 1 に示す．例えば，左図と中図はそれぞれ時区間 $T_1$ と $T_2$ における集計結果を表している．各セルの濃淡は集計値の大きさ（その時間帯にそのセルに属する移動ユーザの人数）に対応している．なお，実際の利用においては，二つの時区間 $T_1, T_2$ における集計値をそのまま用いるか，総数で割った正規化された集計値を用いるかという選択肢があり，これは対象データや応用に依存する．ここでは簡単のため，ユーザが前者の方式を選んだものとして進める．

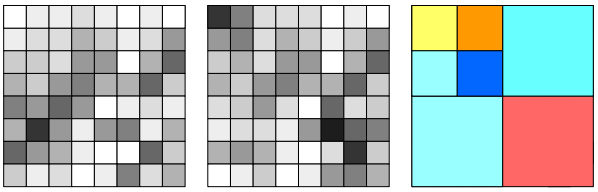


図 1　差分演算のイメージ

図 1 の右図は，時区間 $T_1, T_2$ における入力データに対し，$T_1$ の集計結果に対する $T_2$ の集計結果の差分を近似的に表したイメージを示している．ヒートマップ表現を用い，赤い色が強い領域ほど $T_2$ の時区間において $T_1$ 時区間と比較して集計値が大きいことを示す．一方，青い色が強いほど，逆の傾向があることを意味する．また，大まかな差分の傾向を表現するため，差分のトレンドが同じようなセルが隣接する場合にはそれらをまとめて一つのセルとして表現する．ここでは四分木（quadtree）のような空間分割を想定し，セルの辺の大きさが 2 のべき乗の長さになるようにしている．

このような出力結果の提示により，ユーザは，二つの時区間における変化を容易に把握することができる．しかし，差分演算としてどのようなものを考えればいいかは明らかではなく，対象のデータの性質や応用目的に依存してさまざまなものが考えられる．一方で，大量のデータの中から顕著な変化を検出することは容易ではなく，最新のデータベースシステム技術を効果的に活用した効率的なアルゴリズムの開発が必要となる．このような背景に基づき，本研究では:

（1） 時空間データウェアハウスから差分・差異を検出する

ための汎用的な演算を定義して，ヒストグラムに基づく手段を用いて時空間データの顕著な差分を検出することを目指す．

（2） 大規模配列データの管理および問合せに特化した配列指向型 DBMS である SciDB [3], [4], [5] を活用して差分演算を実装する．SciDB に時空間シミュレーションデータが分析用に蓄積されているものとし，必要に応じてそれに対して差分演算が適用されるというのが想定シナリオである．

本論文の構成は以下の通りである．2 章では差分演算の定義について述べる．3 章は提案手法の説明である．節 3.1 は最適手法について述べ, 節 3.2 と節 3.3 では，近似的なヒストグラム構築手法は提案される. 4 章で SciDB における実装について述べて，5 章は評価実験である．最後に，6 章でまとめと今後の課題について議論する．

## 2. 問題の定義

ここでは，本研究で想定する典型的な時空間データについて述べる．2 次元のユークリッド空間を細粒度で何分割するかというパラメータ $n$ が与えられるものとし，空間を $2^n \times 2^n$ 分割する．一方で，時間の間隔 $\tau$ が与えられており，時間は等間隔の時区間に分けられる．

[定義 1] **時空間配列** $\mathcal{A}_{\mathrm{st}}(X, Y, T, Atr)$ は次元と属性で構成される．次元については，具体的には，$2^n \times 2^n$ 空間グリッドにマッピングされる空間次元 $X, Y$，と等間隔の時区間で構成される時間次元 $T$ となる．なお，$X = \{1, 2, ..., 2^n\}$, $Y = \{1, 2, ..., 2^n\}$, $T = \{1, 2, ..., T/\tau\}$. 時空間配列において，各要素 $e_{i,j,k}$ （$x_i \in X$, $y_j \in Y$, $t_k \in T$）は次元によって位置付けられて，属性の集合 $Atr$ を有する．本研究では，属性については，数値データの集計値を対象として想定する．例えば，避難のシミュレーションデータにおいて，属性は避難の人数の集計値となる．

具体的な集計値の計算方法としては，まず，その時区間および対象セルに関するレコード数をカウントすることが考えられる．選択肢として，分布のパターンを見たいなら，全レコード数で割った頻度分布とすることも考えられる．また，他の種類の対象データに関しては，AVG や MAX などの集計関数の利用も考えられる．

時空間配列データを分析するため，時間に関する変化を検出することや空間上の大幅な分布を把握することは，重要な要求の一つとして考えられる．そこで本研究では，差分・差異を検出するための汎用的な演算を定義する．

[定義 2] 与えられた時空間配列 $\mathcal{A}_{\mathrm{st}}$ において，時区間 $t_a$ と $t_b$ における属性 $v$ の集計結果の差分を**差分配列** $\mathcal{A}_{\mathrm{diff}}(X, Y, \delta(t_a, t_b))$ と呼ぶ．差分の定義（差，比例など）は分析のシナリオによって変わっているが，ここでは，時区間 $t_a$ と $t_b$ の集計値の差を考慮する．

[定義 3] **差分演算** $Diff(\mathcal{A}_{\mathrm{st}}, t_a, t_b, v, B)$ は，差分配列 $\mathcal{A}_{\mathrm{diff}}(X, Y, \delta(t_a, t_b))$ を**最良な差分ヒストグラム** $H$ で表現する．差分ヒストグラム $H = \{b_1, b_2, ..., b_B\}$ は対象となる空間領域 $(X, Y)$ をカバーする $B$ 個のセルで構成される．各セル $b \in H$ は $\{x, y, \delta(t_a, t_b)\}$ （$x \in X, y \in Y$）を含め，空間セル $(x, y)$ における時区間 $t_a$ と $t_b$ の属性の集計値の差分 $\delta(t_a, t_b)$ を表す．また，空間領域がセル $b$ にカバーされる差分配列の要素の集合を

$$cov(b) = \{e \mid e \in \mathcal{A}_{\mathrm{diff}} \wedge contain(area(b), area(e))\} \quad (1)$$

で示す．ただし，area は $XY$ 平面における領域を返す関数で，contain は前者が後者を包含するときに真になる関数である．

最良な差分ヒストグラムは誤差が最小なもので定義される．ここでは，二乗誤差を用いて誤差の尺度とする．すなわち，差分ヒストグラム $H$ の誤差を

$$\mathrm{E}(H) = \sum_{b \in H} \sum_{e \in cov(b)} (e(v) - \delta(t_a, t_b))^2 \quad (2)$$

で定義する．つまり，差分配列 $\mathcal{A}_{\mathrm{diff}}$ における要素の属性値 $e(v)$ と差分ヒストグラムのセルの値の差の二乗である．

## 3. 提 案 手 法

定義 3 によって，最良な差分ヒストグラムはバケットの数 $B$ が与えられたとき，誤差を最小化することと定義されている. [1] では，対応する最良な多次元ヒストグラムを構築する問題は NP 困難と証明されている．本研究では，階層的な分割に着目して差分ヒストグラムを構築する. 特に，四分木の構造に基づいて，各分割は領域を平均的に四つの部分領域に分けることを考える. 主に二つの手法は考えられる．まず，分割統治法に基づく最適な手法を提案し，ボトムアップに統合を行い，最適なヒストグラムを見つける．次に，効率性を考慮するため，二つの近似手法も提案される．ここでは，提案手法は四分木を用いて説明されるが，四分木に限らず，$L \times L$ の分割に拡張できる（$L$:任意の正整数）．

### 3.1 最適手法：分割統治法に基づく手法

本節では，四分木の構造に対して，分割統治法に基づく最適手法を提案する．手法の目標としては，全体の空間領域をカバーする $B$ 個のバケットセットを見つけて，対応する誤差を最小化することとなっている．各領域の最適解は対応する最大四つの子ノード（四分木の場合）の可能な割り当て案を列挙し，計算できることが分かる．それで，アルゴリズムの基本的なアイデアは，再帰的に空間領域を部分領域に分割して，問題をサブ問題に分割する. さらに，サブ問題を解決しながら，ボトムアップに統合を行い，全体領域の最適解を計算することを考える．

具体的には，あるノード $n$ に対して，バケットの数は $k$ となる場合の最適解（$\mathrm{opt}(n, k)$ で示され）は $n$ の子ノード $n_c \in child(n)$ の可能なバケットの数の最適解によって計算できる. 計算式は以下の式となっている．

$$\mathrm{opt}(n, k) = \min \sum_{n_c \in child(n), k_c \in [1,k), \sum k_c \leqq k} \mathrm{opt}(n_c, k_c) \quad (3)$$

以上の式に基づき，最適手法は以下のステップによって，ボトムアップに差分ヒストグラムを構築する．

（1） ヒストグラム初期化: まず，葉ノードに対する初期統合

を行う. 四分木に基づいて，各新しい生成される親ノードには一つの最適リストを生成する. 最適リストでは，全ての可能なバケットの数に対応する最適解は含まれている. 例えば，ノード $n$ の最適リストは $optL(n) = \{opt(n,1), opt(n,b_{max})\}$ で構成される. つまり，バケットの数は 1 と $b_{max}$ である場合の最適解となっている. なお，$b_{max}$ はバケットの数の可能な最大値となり，$n$ の子ノードの数 $size(n)$（空セルがない四分木の場合：$size(n) = 4$）と $B$ によって計算される，i.e., $b_{max} = \min\{size(n), B\}$.

（2） 統合ステップ: ルートノードに着いたまで，ボトムアップに各ノードを親ノードに統合する. 具体的には，式 3 によって，子ノードの最適リストを用いて，親ノードの最適リストを生成する. 対応する疑似コードをアルゴリズム 1 に示す. 基本的なアイデアは，漸進的にバケットの数を増加して，対応する最適解を見つけることとなっている. 初期最適解はバケットの数は 1 となる場合の最適解，つまり親ノードとなっている（1 行目）. 分割の回数は 1 であるときの最適解は子ノードの集合となる（2-4 行目）. 次に，バケットの数は最大バケットの数 $b_{max}$ より大きくなるまで，分割回数 $pTimes$ を増えてながら最適解を見つける（5-10 行目）.8 行目の $findOpt()$ 関数は式 3 の方針によって最適解を見つける手法となっている.

**Algorithm 1:** Merging phase

**Input**: $optL(N_c)$: optimal lists of child nodes, $b_{max}$ : maximum bucket size

**Result**: $optL(n_p)$: optimal list of parent node

```
1  optL(n_p).push_back(n_p);   // initialize the optimal list
2  pTimes ← 1;    // initialize the partition time with 1
3  optS ← N_c; // initial optimal solution consists of the child nodes
4  curB ← N_c.size();          // initialize current size of buckets
5  while curB < b_max do
6      optL(n_p).push_back(optS); // update the optimal list
7      pTimes++;        // update the next partition time
8      optS ← findOpt(N_c, pTimes);      // find the optimal solution by pTimes times of partitionings
9      curB ← optS.size();
10 end
11 return optL(n_p);
```

#### 3.1.1 計算量の分析と効率化

最適手法の計算コストは，最適リストの計算回数と各ノードの可能なバケットの数によって決まっている. 最悪ケースでは，四分木の全てのノードを計算する必要があるため，$C \times log^C$ 回の最適リストを計算することとなっている. なお，$C$ は葉ノードの数である. 毎回の統合（最適リストを計算する）では，子ノードに割り当てるバケットの数によって結果は異なるため，すべての割り当てる案の数を列挙して，最適解を見つける. ここでは，可能な最大バケットの数は $B$ となるため，四つの子ノードがある場合，最大な可能な割り当て案の数は $B^3$ となっている. そこで，全体的な計算コストは $O(Clog^C B^3 t_E)$ である.$t_E$ は統合を行うとき，親ノードの誤差の計算時間となっている. 誤差の計算については，実際のデータの集計が必要となることから，大規模なデータに対して支配的な影響を与えうる．これについては，後述のようにデータベースシステムの集計機能を活用することや，計算済みの値を保持するなどの工夫で実際のコストを削減できる．具体的には，各ノードで計算済みの統計情報を保持して，ボトムアップに親ノードの誤差を計算する際に再利用される．つまり，以下の式を用いて，子ノードの統計情報（例，平均値など）を用いて，新たに生成される親ノードの誤差を計算できる．

$$n = \sum_i n_i \tag{4}$$

$$m = \frac{\sum_i m_i n_i}{n} \tag{5}$$

$$E(n_p) = \sum_i E_i + \sum_i m_i^2 n_i - m^2 n \tag{6}$$

$m$，$n$，$E(n_p)$ はそれぞれノード $n_p$ の属性の平均値，含まれている葉ノードの数と誤差である．$m_i$，$n_i$，$E_i$ はそれぞれノード $n_p$ の子ノード $n_i$ $(i = 1, 2, \ldots)$ の平均値，含まれている葉ノードの数と誤差である．これらによって，ボトムアップに誤差を計算しながら統合を行い，$t_E$ のコストを $O(1)$ になることができる．

### 3.2 近似手法 1：貪欲に基づく分割統治法

節 3.1.1 に述べたように，最適手法の計算コストは高いことが分かる.$t_E$ を $O(1)$ になる場合も，計算量は $O(Clog^C B^3)$ となる. 可視化の実用性を考慮するため，近似手法を提案する. ここでは，最適手法の分割統治法に基づき，貪欲的な方針を用いて解を見つける. つまり，各ノードに対して，バケットの数は 1 から増えて行って，誤差の減少が一番大きい分割を選んで，分割を行う. 具体的には，以下の式によって，ノード $n$ のバケットの数は $k$ のときの解は，一個前の分割までの結果（バケットの数は $k'$）と次の分割 $P(b)$ から一番誤差の減少が大きい分割を用いて計算される.

$$\mathrm{H}(n,k) = \mathrm{H}(n,k') + \max_{b \in \mathrm{H}(n,k')} \delta\{E(P(b))\} \tag{7}$$

また，分割 $P(b)$ により増加したバケットの数 $\delta b$ は次の式に満たす：$k' + \delta b \leqq k$.

本手法の計算量は $O(Clog^C B)$ となっている.

### 3.3 近似手法 2：Hybrid 手法

本手法では，次の簡単な観測に基づく：ヒストグラムの深さが大きい（細かい粒度）ほど，誤差値も小さくなりうる. そこで，深さがある程度大きい場合，近似的な（貪欲的な）方針で解（バケットリスト）を見つけて，深さが小さい場合，最適手法を用いてヒストグラムを構築すると，結果の有効性を高めることができると考えられている. ここでは，経験的に深さの閾値 $d_0$ を設定して，深さは $d_0$ より大きい場合，近似手法 1 を使って統合を行う. 一方で，深さは $d_0$ より小さい場合，最適手法を用いてヒストグラムを構築する. 節 5. で評価実験は示される.

## 4. SciDB における実装

科学データは基本的に順序付けられている（時間的にあるいは，位置的に）ため，順序付けられていない関係モデルより，配列データの形式はもっと適切であると考えられている [2]. 本研究では，配列指向型 DBMS である SciDB を活用して差分演算を実装する．SciDB では，大規模な配列データは複数のチャンクに分けられて，分散して SciDB クラスタに格納される．チャンクは SciDB における最小の I/O 単位である.SciDB クラスタは複数のインスタンスで構成されており，インスタンスごとに複数のチャンクが格納される．このようなデータ構造によって，SciDB は並列分散処理を行う [4]．問合せ処理の過程では，あるインスタンスがコーディネータ（coordinator）として動作し，問合せプランを生成してほかのワーカインスタンスに割り当てる．次に，ワーカインスタンスは関連するチャンクデータを処理して中間的な結果を生成する．最後に，コーディネータはワーカインスタンスの結果を統合して，最後の結果を出す．

SciDB では，さまざまなビルトイン演算が提供されている一方，特定な要求に対応するためのユーザ定義演算（UDOs）とユーザ定義集約（UDAs）もサポートされている．SciDB のプラグイン仕組みを用いることで，差分演算を C++言語で実装する．上記の SciDB の機能を活用して，本論文で提案したヒストグラム構築手法を実装する．実装の詳細は次のサブ節に示される.

### 4.1 疑似コード

最適手法の SciDB における実装はアルゴリズム 2 に示される. 統合の関数を変わることによって，簡単に近似手法に適応できる.

入力としては差分配列とバケットの数となっている. 差分配列は与えられる比較となる時間帯と配列データによって，SciDB のビルトイン演算を用いて計算されることを想定する. 出力は最適ヒストグラムであり，ルートノードの最適リストとなっている. アルゴリズム 2 は節 3.1 に紹介された二つのステップに基づき，ボトムアップに統合を行う. 初期統合と最適統合は UDA として実装し，それぞれ *initialMerge* と *optMerge* で示される. まず，SciDB の集約演算 *regrid* を用いて，ブロックサイズ $2 \times 2$ で統合を行う（1 行目）. 次に，ルートノードに着いたまで（つまり，深さは 0 となるまで），最適統合を行う（3-6 行目）.

**Algorithm 2:** Implement optimal algorithm on SciDB

**Input**: $\mathcal{DA}$: differential array, $B$ : bucket size, $depth_{max}$: maximum depth

**Result**: $optHist$: output array of optimal histogram

1 $optHist \leftarrow regrid(\mathcal{DA}, 2, 2, initialMerge(\text{delta}) \text{ as interNode})$ ; // initial merging

2 $depth_{par} \leftarrow depth_{max} - 2$ ; // depth of parent nodes

3 **while** $depth_{par} > 0$ **do**

4 $\quad optHist \leftarrow regrid(optHist, 2, 2, optMerge(\text{interNode}) \text{ as interNode})$ ; // merging:generate parent node with optimal list

5 $\quad depth_{par} - -$;

6 **end**

7 **return** $optHist$;

## 5. 評 価 実 験

### 5.1 対象となるデータセット

本研究で使っている二つのデータセットは，それぞれ大規模地震の発生時の，高知市と関東における人流の避難シミュレーションデータである．データセット-1 は東京大学関本研究室からの提供による．今回実験で用いるサンプル人流は，地震発生後に 6 時間分の約 4 万人の人が避難する状況を，パーソントリップデータに基づいてシミュレーションしている．データセット-2 は東京工業大学から提供されて，発災から 24 時間後の避難者の位置情報と状態などの情報を含める 194,226,288 ほどのレコードとなっている.

シミュレーションデータは $(id, time, x, y)$ という形式のレコードの集まりである．$id$ はユーザ ID，$time$ は時刻， $x, y$ は時刻 $time$ におけるユーザの位置である．シミュレーションデータは，シミュレーション後には変化しない静的なデータである．本研究が目的とするデータウェハウス的な問合せ処理においては，静的なデータであることの利点を生かし，前処理を行うことで対話的処理を効率化することが一般的である．そこで，今回のデータについては，空間的には $4,096 \times 4,096$ のレベルの最大粒度の分割を行う．この空間の細粒度の分割に基づき，各セル内で 1 分ごとにその中の避難者数を集計する．前処理の結果得られた集計データを SciDB にロードし，これを差分演算の問合せ処理の対象とした．また，対象となる配列データ Kouchi_eva と Kantou_eva のデータスキーマを以下に示す.

```
Kouchi_eva<Num:int64>
[X=1:4096:0:4096;Y=1:4096:0:4096;T=0:360:0:256]

Kantou_eva<Num:int64>
X=1:4096:0:4096;Y=1:4096:0:4096;T=0:1440:0:1024]
```

以上の配列では，座標 $X$,$Y$ と時間 $T$ という三つの次元があり，属性 $Num$ は対応するセルの避難人数の総数である．データセット-1 の非空セルの数は 144,169 個である. データセット-2 の密度はより高くて，879,333 個非空セルを含める.

### 5.2 評 価 基 準

今回は効率性と有効性に対して評価を行う. 効率性については，差分配列の生成時間 $T_d$ とヒストグラム構築手法の処理時間 $T_h$ の総和 $T$ を用いて比較を行う. 一方で，有効性は結果ヒストグラムの誤差を用いて，次の式で誤差率を計算する.

$$Error_{ratio} = (E - E')/E' \tag{8}$$

ここでは，$E$ は結果ヒストグラムの誤差であり，$E'$ は最適解の誤差となる.

次の二節では，それぞれ配列データのサイズ（非空セルの数：$C$）とバケットの数 $B$ を変化するとき，対応する結果を比較する. また，比較となる手法はそれぞれ表 5.2 に示されている.

Hybrid_3, Hybrid_4, Hybrid_5 はそれぞれ深さの閾値は 3,4,5 となる手法に対応する.

| 提案手法 | 表現 |
|---|---|
| 最適手法 | Exact |
| 近似手法 1 | Greedy |
| 近似手法 2 | Hybrid_3, Hybrid_4, Hybrid_5 |

### 5.3 配列データのサイズ $C$ を変化

データセット-1 とデータセット-2 に対して，それぞれ 30 分と 20 分の粒度で時間の次元で集計を行い，異なる非空なセル数（$C$）となる差分配列を生成する. 図 2 と図 3 によって，処理時間 $T$ は $C$ の数の増えることに従って，増える傾向があることが分かる. また, 最適手法 (Exact) の処理時間は一番長くて, 近似手法 1(Greedy) と近似手法 2（Hybrid）の処理時間はほとんど同じである. 図 4 と図 5 では, 対応する誤差率の結果は示される. 結果の有効性については, 次の順番となる : Hybrid_5 > Hybrid_4 > Hybrid_3 > Greedy.　これは,Hybrid 手法では, 深さの閾値は大きいほど, 結果の精度が高いためである. これらによって, バケットの数 ($B$) は 50 である場合,Greedy と Hybrid 手法の処理時間は近くて,Hybrid の方は結果の質が良いことが分かる.

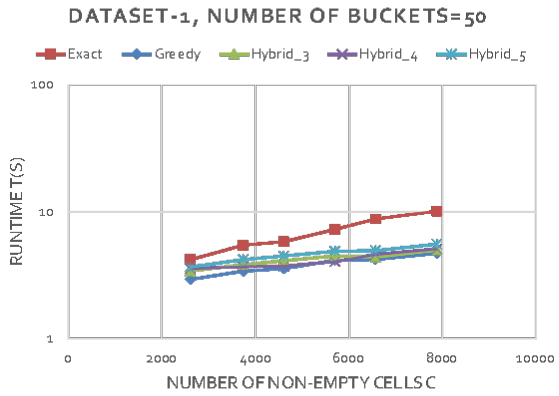


図 2　Runtime of varying $C$ (Dataset-1, $B = 50$)

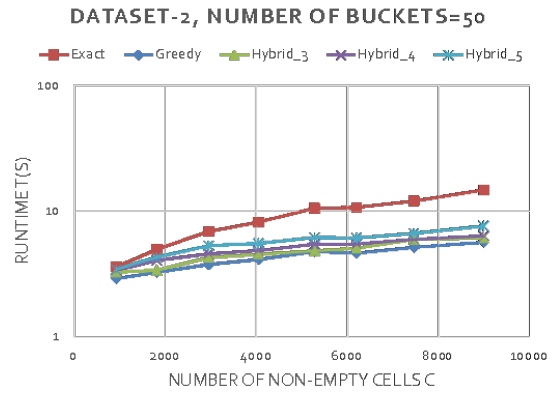


図 3　Runtime of varying $C$ (Dataset-2, $B = 50$)

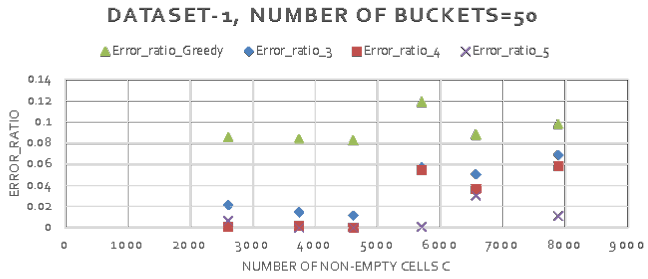


図 4　Error ratio of varying $C$ (Dataset-1, $B = 50$)

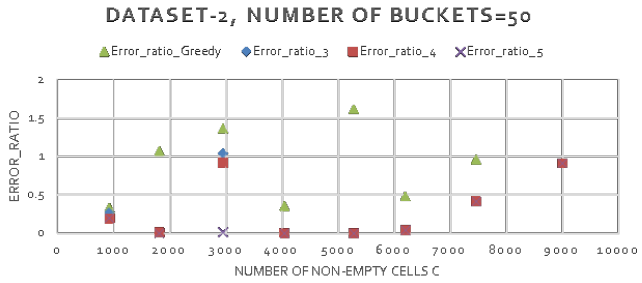


図 5　Error ratio of varying $C$ (Dataset-2, $B = 50$)

### 5.4 バケットの数 $B$ を変化

本節では, データセット-1 とデータセット-2 を用いて, それぞれ 2 次元の $4096 \times 4096$ の配列 diffArray_sparse と diffArray_dense を生成する. 配列 diffArray_sparse は 8460 個の非空セルを含めて,diffArray_dense は 15,248 個の非空セルを含める.

図 6 と図 7 は配列 diffArray_sparse に対して, バケットの数 $B$ を変化する場合の処理時間と誤差率となっている. バケットの数が増えるほど, 手法の処理時間と誤差率も増えて行くことが分かる. また,Greedy の効率性が一番良いが,Hybrid の方が結果の質が良い.

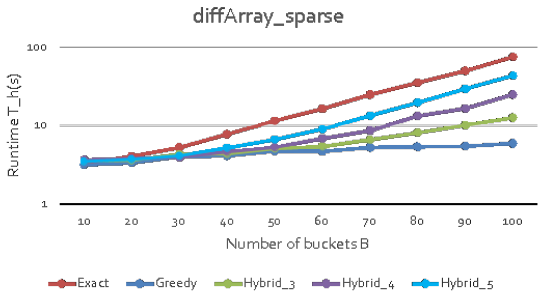


図 6　Runtime of varying $B$ (diffArray_sparse)

図 8 と図 9 は配列 diffArray_dense に対して, バケットの数 $B$ を変化する場合の処理時間と誤差率を示す. バケットの数は 50 より小さい場合,Hybrid と Greedy の処理時間はほとんど同じであり,Hybrid の結果の質が良い. また, バケットサイズが変わっても,Hybrid_4 と Hybrid_5 の誤差率はほとんど 0 である. 配列 diffArray_sparse と比べると, 配列 diffArray_dense の場合（データの密度が高い）,Hybrid_4 の効率性と結果の質両方とも良いことが分かる.

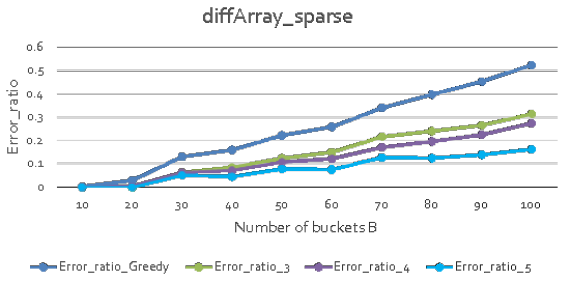


図 7　Error ratio of varying $B$ (diffArray_sparse)

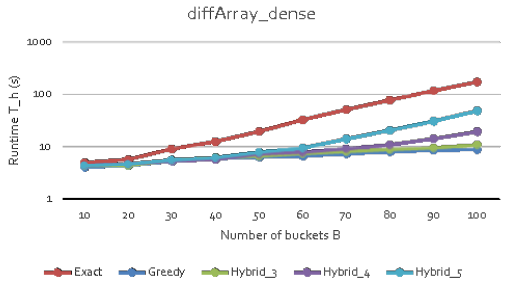


図 8　Runtime of varying $B$ (diffArray_dense)

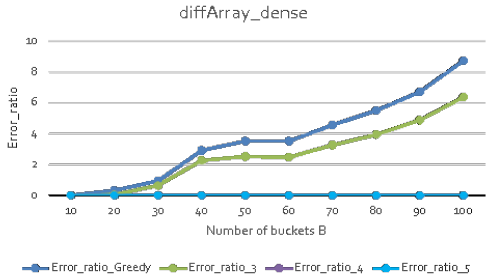


図 9　Error ratio of varying $B$ (diffArray_dense)

### 5.5　実験のまとめ

以上の実験結果により, 次の結論をまとめる.

（1） 配列データのサイズとバケットの数が増えるほど, 手法の処理時間も増えて行くが, 最適手法より, 近似手法の効率性が良いことが分かる.

（2） Greedy 手法の効率性は一番良いが,Hybrid 手法の方が結果の質が良い.

（3） バケットの数は小さい（50 程度）場合, 配列データのサイズが増えても,Hybrid 手法の効率性は Greedy とほとんど同じである.

（4） 配列データの密度が高い場合,Hybrid_4 の効率性と結果の質両方とも良いことが分かる.

## 6.　まとめと今後の課題

本稿では，時空間データの高度な分析機能を実現するため，差分演算に着目し，差分ヒストグラムの構築手法と効率化の手法を提案した．また，配列指向型 DBMS における実装と, 効率性と有効性を考慮した評価実験を行った.

本論文で示した差分演算では，2 つの時区間 $T_1, T_2$ がユーザにより指定されることを想定していた．しかし，このようなアプローチは，事前にユーザが着目すべき時区間についてすでにアイデアを有しているときにしか使えないことから，より多くのケースに対応することが必要である．さらに今後の課題としては，差分演算のセマンティクスの拡張が考えられる．今回は単純に差の大小のみに着目したが，増加傾向にあるか，減少傾向にあるか，また増加・減少の速さはどの程度かといった情報に着目した差分演算も開発したいと考えている．このような差分を考える場合，どのように可視化するかもポイントとなることから，可視化手法についても検討を行う.

## 謝　　辞

本研究の一部は，科研費（16H01722），CREST「大規模・高分解能数値シミュレーションの連携とデータ同化による革新的地震・津波減災ビッグデータ解析基盤の創出」，および文部科学省「実社会ビッグデータ利活用のためのデータ統合・解析技術の研究開発」による. 避難シミュレーションデータを提供いただいた東京大学関本研究室に感謝します.

## 文　　献

[1] S. Muthukrishnan, V. Poosala, and T. Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. In *ICDT*, volume 1540 of *LNCS*, pages 236–256. Springer, 1999.

[2] F. Rusu and Y. Cheng. A survey on array storage, query languages, and systems. *CoRR*, abs/1302.0103, 2013.

[3] Paradigm4: Creators of SciDB a computational DBMS. `http://www.paradigm4.com/`.

[4] M. Stonebraker, P. Brown, A. Poliakov, and S. Raman. The architecture of SciDB. In *SSDBM*, volume 6809 of *LNCS*, pages 1–16, 2011.

[5] M. Stonebraker, P. Brown, D. Zhang, and J. Becla. SciDB: A database management system for applications with complex analytics. *IEEE Computational Science & Engineering*, 15(3):54–62, 2013.

[6] J. Zhao, Y. Ishikawa, Y. Wakita, and K. Sugiura. Difference operators in simulation data warehouses. *Journal of Disaster Research*, 12(2):347–354, 2017.

[7] J. Zhao, K. Sugiura, Y. Wang, and Y. Ishikawa. Simulation data warehouse for integration and analysis of disaster information. *Journal of Disaster Research*, 11(2):255–264, 2016.

[8] 石川佳治. 大規模データアナリティクスに関する研究動向と展望. **電子情報通信学会論文誌** *D*, J97-D(4):718–728, 2014 年 4 月.