DEWS2005 4-C-05

移動軌跡ストリームデータのための動的ヒストグラム構築手法

町田　陽二†　　石川　佳治††,†††　　北川　博之††,†††

† 筑波大学大学院理工学研究科　〒 305–8573 茨城県つくば市天王台 1–1–1
†† 筑波大学大学院システム情報工学研究科　〒 305–8573 茨城県つくば市天王台 1–1–1
††† 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 1–1–1
E-mail: †y-machida@kde.cs.tsukuba.ac.jp, ††{ishikawa,kitagawa}@cs.tsukuba.ac.jp

あらまし　GPS や通信技術の発展に伴い，移動する多数のオブジェクトの移動状況の追跡が容易になっている．こうした移動状況データを分析・予測に利用するには，ストリーム的に配信されてくる移動状況データを効率よく要約する必要がある．そこで，我々はマルコフ連鎖モデルに基づき移動データを要約する，移動ヒストグラムを動的に構築する手法の開発を進めている．提案手法ではヒストグラムを表現する物理的なデータ構造として木構造を採用し，移動パターンを複数の粒度で表現する．移動オブジェクトの移動軌跡が送られた際には，インクリメンタルにヒストグラムを更新する．また，本論文では，忘却の概念を用いた非定常的な移動パターンへの対応についても提案を行う．
キーワード　移動ヒストグラム，移動パターン，マルコフ連鎖モデル，インクリメンタル処理，忘却

A Dynamic Histogram Construction Method for Moving Trajectory Stream Data

Yoji MACHIDA†, Yoshiharu ISHIKAWA††,†††, and Hiroyuki KITAGAWA††,†††

† Graduate School of Science and Engineering, University of Tsukuba
†† Graduate School of Systems and Information Engineering, University of Tsukuba
††† Center for Computational Sciences, University of Tsukuba
1–1–1 Tennoudai, Tsukuba, Ibaraki, 305–8573 Japan
E-mail: †y-machida@kde.cs.tsukuba.ac.jp, ††{ishikawa,kitagawa}@cs.tsukuba.ac.jp

Abstract　With the recent progress of spatial information technologies and communication technologies, it becomes easy to track trajectories of many moving objects in real-time. To use obtained moving object trajectories for the analysis and prediction, we need to accumulate given trajectory streams in an efficient and accurate manner. For this purpose, we propose a mobility histogram construction method based on the Markov chain model. The histgoram is physically represented as a tree structure and represents movement patterns in multiple granularity. When a new trajectory sequence is obtained, it updates the histogram structure incrementally. We also show an extended histogram maintenance method for non-stationary moving patterns using decay factors.
Key words　mobility histogram, moving patterns，Markov chain model，incremental processing, forgetting

1. ま え が き

GPS や通信技術の発展に伴い，移動する多数のオブジェクトの移動状況の追跡が容易になっている．また，こうしたオブジェクトの移動状況を蓄積するのに，時空間データベースがますます一般的になっている [5]．大量の移動オブジェクトの移動状況をリアルタイムに追跡し，分析・予測に役立てるには，ストリーム的に配信されてくる移動状況データを効率よく要約することが求められる．さらに移動オブジェクトの移動状況を要約することは，オブジェクトの移動状況を蓄積した時空間データベースにおける問合せ処理でも有用である．

ストリーム配信されるデータを継続的に集計し，コンパクトに移動状況を要約することは重要な研究課題である．そこで本研究グループは，移動データを要約する**移動ヒストグラム**（mobility histogram）の概念を提案し，ストリーム的に配信されてくる移動状況データを効率よく集計するために動的なヒストグラム構築手法の研究を進めている [17]．このアプローチでは，マルコフ連鎖モデルに基づいて移動パターンの移動統計量

を集約する．移動オブジェクトの移動軌跡が送られるたびに，インクリメンタルにヒストグラムの更新を行う．精度を落とさずに効率的な処理が実現できること，また，コンパクトにヒストグラムを表現できることが課題となる．我々のグループの過去の研究 [17] では，木構造に基づくヒストグラム構造のアイデアを提案した．この手法では，移動軌跡データが少ない時点では少数のバケットからなる木構造に移動データを要約して管理する．移動データが追加されるにしたがって木構造を順次拡張し，指定された上限のサイズに達した時点で，必要に応じて木の枝刈りなどを行う．これにより，移動パターンの動的な変化（例：ある領域への移動が少なくなり，別の領域への移動が増加した）への対応をはかる．しかし，このアプローチは効率の面では優れていたものの，精度の面では不十分であった．初期の時点において移動軌跡データを要約して表現してしまうため，そこで発生した誤差が後々の段階まで波及してしまうという点がその理由であった．

そこで，本研究では，ヒストグラムに対する新たなデータ構造の提案を行う．提案手法では，初期段階からメモリを積極的に利用し，移動オブジェクトの移動状況を正確に反映することを目指す．加えて本稿では，忘却の概念の導入によるヒストグラム構造の拡張を図る．忘却の概念により，時間につれて移動パターンが変化するような非定常的な移動状況への対応を行う．また，忘却の概念を用いた木構造のヒストグラムのメンテナンス処理において，メモリサイズの制限も考慮することにより，コンパクトなデータ構造の表現もはかる．

2. 背景および関連研究

ヒストグラムは，データを要約するための一手法であり，データベースの分野でも，特に問合せ最適化や近似問合せにおいて盛んに研究開発が進められている [10]．これまで特に単一属性に関するヒストグラム構築の研究が進められてきたが，近年では複数属性に対するヒストグラム構築に関する技術開発も着目されている [3], [9], [12], [14], [19], [20]．また，動的なヒストグラムの管理に関しても研究が進められている [1], [6]〜[8], [13]．

本研究で対象とするヒストグラムは，特殊な制約はあるが多次元のヒストグラムと分類される．与えられたデータに対して最適なヒストグラムを構築することは NP 完全に属する困難な問題であり [10]，近似的な構築法が求められる．また，本研究の目的とする移動軌跡データのリアルタイムの集積を実現するためには，インクリメンタルな更新に対応するために，とくに効率性が求められる．

移動状況データをヒストグラム表現することで，大量の移動オブジェクトの移動状況を効率よく分析予測することが可能になる．例えば，移動オブジェクトがどこにいた可能性が高いのか，移動オブジェクトが今後どこに行く可能性が高いのか，ある領域においてオブジェクトがどのように移動するのかなどは統計量をもとに推定できる．関連研究 [4], [11], [16] と比較した本研究の他の特徴としては，ストリーム的な移動軌跡データのリアルタイム処理がある．静的なデータを集約するのではなく，大量のストリームデータを効率的に集約する点を特徴とする．

3. 移動パターンのモデリング

3.1 Z-ordering

まず，準備として，本手法における移動パターンのモデリングのために用いる，2 次元平面の番号付け手法について述べる．この手法は 2 次元平面を 1 次元で順序付けする手法の一つである **Z-ordering** [15] に基づいている．

3.1.1 セルの番号付け

図 1 のように，2 次元空間が各次元ごとに 2^P（P は分割レベルを表す）個ずつ，$C = 2^{2P}$ 個のセルに分割されているとする (図 1 は $P = 1, 2$ の場合)．図 1 に示した分割のことを**レベル P の分割**と呼ぶ．各セルには，$2P$ ビットのセル番号が付与されている．セル番号の上付き数字は，空間分割レベルを表している．

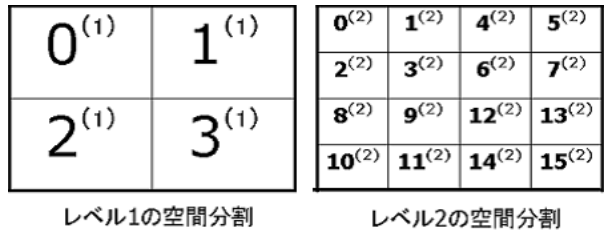

図 1　Z-Ordering

このレベルの概念を用いることで空間分割の粒度を指定できる．レベルの異なる分割を対比して，粗い分割の方を上位の空間分割，細かい分割の方を下位の空間分割と呼ぶ．図 2 にこの関係を例示する．

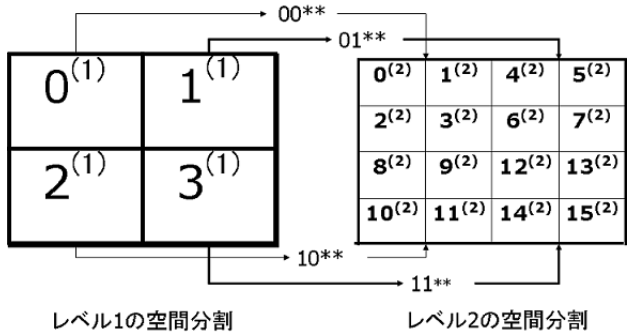

図 2　上位・下位の空間分割の関係

上位空間のセル番号をバイナリ表記し，末尾に 00，01，10，11 を付与すると 1 レベル下位の空間のセル番号が得られる．逆に，セル番号から下位の 2 ビットを削除すると 1 レベル上位の空間の対応するセル番号が得られる．これは，Z-ordering を用いたことによる特徴である．

3.2 マルコフ連鎖モデル

時空間データ分析における**マルコフ連鎖モデル**（Markov chain model）は，ある地域から別の地域へある期間内にどの程度の人口が移動したなどの，移動オブジェクトの時空間的な移動傾向の把握に用いられる [18]．以下では，マルコフ連鎖モデルの概念について述べる．

図 3 は，時刻 $t = \tau$ でセル 9 にいたオブジェクト A が，次の時刻 $t = \tau + 1$ でセル 12 に，そして $t = \tau + 2$ の時点でセル 6 に移動した状況を示している．なお，ここでは分割レベルの表記を省略している．

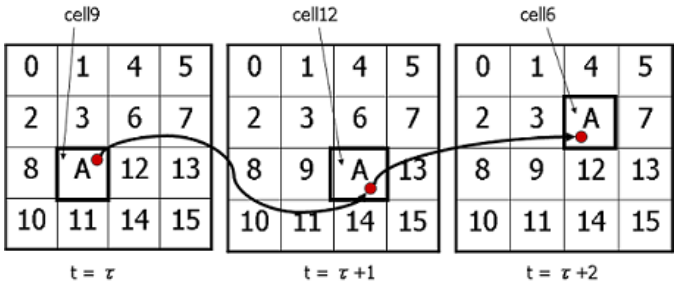

図 3　マルコフ連鎖モデルの概念

図 3 の 9 → 12 → 6 という移動の例は 2 次のマルコフ遷移である．このような遷移情報を集積しておけば，「9 → 12 と遷移したオブジェクトが次にどのセルに移動する確率が高いか」，「12 → 6 と移動したオブジェクトは，セル 12 の前にどこにいた可能性が高いか」といった質問に，確率の推定値をもとに答えることができる．また，ある領域においてオブジェクトがどのように移動するかといった移動パターンの分析などにも利用できる．

本稿で提案するヒストグラムは，指定された遷移の次数 n に対し，n 次のマルコフ遷移を表現する．ヒストグラムの構造について，次節で説明する．なお，以下の説明では $n = 2$ の場合を想定するが，アルゴリズムは他の n の値にも適用可能である．

4. 提 案 手 法

4.1 データ構造

4.1.1 基本的アイデア

入力として与えられる移動軌跡データにおいて，分割レベル m でセルの番号付けがなされているとする．m を**最大空間分割レベル**と呼ぶ．移動ヒストグラムは木構造で表現する．木の各ノードは 0 個以上 4 個以下の子を有し，そのノードに対応する移動軌跡の数を集積するためのカウントも有する．

ここで，遷移シーケンス $a^{(m)} \rightarrow b^{(m)} \rightarrow c^{(m)}$ を挿入することを想定する．$a^{(m)}$，$b^{(m)}$，$c^{(m)}$ はセル番号を表す．また，$a^{(m)}$ をステップ 0 のセル，$b^{(m)}$ をステップ 1 のセル，$c^{(m)}$ をステップ 2 のセルと呼ぶ．上記遷移シーケンスを木に挿入する場合を例にとって説明する．まず $a^{(m)}$，$b^{(m)}$，$c^{(m)}$ をバイナリ表記して，まず，$a^{(m)}$ の上位 2 ビットを取り出す．その内容が 00 $(= 0^{(1)})$，01 $(= 1^{(1)})$，10 $(= 2^{(1)})$，11 $(= 3^{(1)})$ のいずれかであるかに応じて，対応する辺を辿り，子ノードへと達する．ただし，そのような辺がない場合には，新たに辺を作成する．さらに $b^{(m)}$ の上位 2 ビットを取り出し，その値に応じて同様に子ノードにアクセスする．$c^{(m)}$ の上位 2 ビットについても同様である．以上のステップは空間分割レベル 1 に対応する．

次に，$a^{(m)}$ の次の 2 ビット，$b^{(m)}$ の次の 2 ビット，$c^{(m)}$ の次の 2 ビットをそれぞれ取り出し同様の処理を行う．このステップは空間分割レベル 2 に対応する．このようなステップを繰り返し，$2m$ ビットずつを処理した時点で末端ノードに至る．この末端ノードまで至る過程において，各ノードのカウントを 1 ずつインクリメントする．

図 4 に $m = 2$ の場合について，$3^{(2)} \rightarrow 6^{(2)} \rightarrow 9^{(2)}$ を追加する例を示す．各ステップをバイナリ表記すると，step 0 は $3^{(2)}$ から (0011) となり，同様に，step 1 は $6^{(2)}$ から (0110) となり，step 2 は $9^{(2)}$ から (1001) となる．各ステップの上位 2 ビットを用いて 00 → 01 → 10 を辿り，空間分割レベル 1 が終了する．続いて次の上位 2 ビットをもとに 11 → 10 → 01 を辿り，空間分割レベル 2 が終了して葉まで達する．この過程で各ノードにカウントが 1 追加される．

図 4 の点線はノードを辿るような遷移シーケンスが依然として到着していないことを表しており，実際にはそれ以下の部分木は存在しない．実線はノードを辿るような遷移シーケンスが過去に挿入されたことを表している．

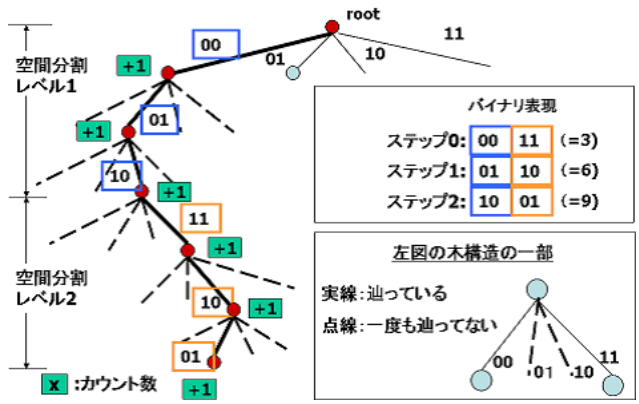

図 4　木構造 (最大空間分割レベル $m = 2$ のとき)

4.1.2 追加アルゴリズム

アルゴリズム 1 に，移動データが定常的である場合の遷移シーケンスの追加処理のアルゴリズムを示す．このアルゴリズムでは，最大空間分割レベルが m であり，マルコフ遷移の次数が n であることを想定している．ヒストグラムの木構造の各ノードは，カウント値と子ノードへの 4 個のポインタを有する．初期段階では，木構造はルートノードただ 1 つからなる．

アルゴリズムの概要を説明する．まず，ノードへのポインタ $node$ を，ヒストグラムのルートである $root$ で初期化する．次に，シフト量 $shift$ を初期化する．アルゴリズムは 2 重のループ構造からなり，木構造を末端まで辿りながら，その過程で訪れたノードのカウントをインクリメントする．外側のループは，粗い分割から細かい分割へ進む，分割の粒度に関するループであり，内側のループは 0 番目のステップから n 番目のステップという遷移シーケンスの遷移に対応するループである．ノードを辿る過程でまだ割り当てられていないノードが存在した場合，ノードを新たに割り当てる．このような処理により，1 つの遷移シーケンスの追加において辿る辺の数は $m \times (n + 1)$ 個となる．ヒストグラムの構造は主記憶上に置かれ，また，m, n の値は一般に小さいため，追加処理は高速に行える．

ここでヒストグラムのサイズについて考える．すべての移動

Algorithm 1 insert_sequence($tseq$)

```
1: 入力 tseq: 追加する遷移シーケンス
2: {root はヒストグラムのルートを表す }
3: node := root; {node はノードを指すポインタ }
4: shift := 2(m − 1); { シフト量を指定 }
5: for all i := 0 to m − 1 do
6:    for all j := 0 to n do
7:       node.count++ { カウント値をインクリメント }
8:       c := (tseq[j] >> shift) & 0x3; {2 ビットを抽出 }
9:       if (node.child[c] ≡ NULL) then
10:         node.child[c] := new_node(); { 新規ノードの作成 }
11:      end if
12:      node = node.child[c]; { 次ノードへ辿る }
13:   end for
14:   node.count++;{ 末端ノードのカウントの更新 }
15:   shift := shift − 2;
16: end for
```

パターンについて遷移シーケンスが木構造に挿入されることを考えると，最大 $4^{m\times(n+1)}$ のノードが必要になる．各ノードにおいて，カウンタが 4 バイト整数であり，4 個のポインタが 4 バイトであるとすると，各ノードのサイズは 20 バイトとなる．そのため，たとえば，$m = 4$，$n = 2$ の場合は最悪のヒストグラムのサイズは 320MB となる．一方，$m = 5$，$n = 2$ と一段詳細度を上げると，最悪のサイズは 20GB となり膨大となる．

ただし，実際の移動オブジェクトの移動パターンには大幅な偏りが存在する．ある時点である位置に存在するオブジェクトが，次の時点で大きく離れた地点に移動することは現実的にはありえないことから，たとえば図 3 において $0^{(2)} \rightarrow 15^{(2)} \rightarrow 0^{(2)}$ という遷移シーケンスは発生しえない．このため，実際の利用においては木構造のほんの一部のみが実体化されるため，実際に要求されるメモリ領域は上記よりは大幅に小さいと考えられる．

ヒストグラムの木構造に関しては，実装レベルで配列などを用いてよりコンパクトに表現することが考えられる．しかし，上記のアプローチは基本的にデータを追加していくだけであるため，データの追加に伴い木が巨大になる可能性は避けられない．また，過去の遷移シーケンスと最近の遷移シーケンスのカウントが全く同等の扱いになっているので，時間的に移動パターンが変化するような非定常的な状況に適用できない．4 節では，こうした問題に対し忘却の概念を導入することで解決を図る．

4.2 問合せ処理

次いで，ヒストグラムを用いて，移動軌跡のカウント数を求める問合せに答えるための問合せ処理方式について述べる．

4.2.1 基本的アイデア

まず，最大空間分割レベルが $m = 2$ の場合について，例を用いて処理の概要を説明する．

（1） 移動軌跡の各セルのレベルが最大空間分割レベルと一致する場合：たとえば，$3^{(2)} \rightarrow 6^{(2)} \rightarrow 9^{(2)}$ を考える．この場合が一般的で容易である．遷移シーケンスの挿入と同様に，各ステップをバイナリ表記して各レベルごとに 2 ビットずつ辿っていき，目的のカウントを返す．

（2） 移動軌跡の各セルのレベルが最大空間分割レベルよりも粗い場合：場合分けして考える．

（a） 全ステップが最大分割レベルよりも粗い場合：たとえば $0^{(1)} \rightarrow 1^{(1)} \rightarrow 2^{(1)}$ を考える．細かい粒度ではなく，おおざっぱな移動状況を捉えたい場合にこのような問合せが有効となる．この場合，(1) と同様に，各ステップをバイナリ表現にして，各レベルごとに 2 ビットずつ辿っていく．この場合は，空間分割レベル 1 を辿り終えたノードのカウントをそのまま返す．空間分割レベル 2 を辿る必要はない．

（b） 一部のステップが粗いレベルである場合：例として $1^{(1)} \rightarrow 1^{(1)} \rightarrow 9^{(2)}$ を考える．(1) と同様に，各ステップをバイナリ表現にして，各レベルごとに 2 ビットずつ辿っていく．この場合は，空間分割レベル 1 の $01 \rightarrow 01 \rightarrow 10$ の後，$\{00, 01, 10, 11\} \rightarrow \{00, 01, 10, 11\} \rightarrow 01$ の $4 \times 4 \times 1 = 16$ 通りの経路をすべて辿り各カウントを調べ，総和をとる必要がある．

（3） 移動軌跡のセルのレベルが最大空間分割レベルよりも大きい場合：たとえば，$3^{(2)} \rightarrow 25^{(3)} \rightarrow 9^{(2)}$ を考える．$3^{(2)} \rightarrow 25^{(3)} \rightarrow 9^{(2)}$ は，空間分割レベル 2 までの木では表現できないので，カウントを近似的に推定する必要がある．そこで，まず 25 をバイナリ表記すると，(011001) となる．この上位 4 ビットが $25^{(3)}$ の上位空間に相当する．すなわち，その上位空間は $3^{(2)} \rightarrow 6^{(2)} \rightarrow 9^{(2)}$ である．$3^{(2)} \rightarrow 6^{(2)} \rightarrow 9^{(2)}$ のカウントはヒストグラムのノードに保持されている．$6^{(2)}$ の下位空間には $\{24^{(3)}, 25^{(3)}, 26^{(3)}, 27^{(3)}\}$ があるので，4 等分したカウントで近似する．

4.2.2 問合せ処理のアルゴリズム

問合せ処理のアルゴリズムを**アルゴリズム 2** に示す．このアルゴリズムは再帰的な関数 est_count() を定義している．この関数は 4 つの引数 $node$, $qseq$, $levels$, $step$ をとる．$node$ は処理対象のノードへのポインタ，$step$ は現在処理対象の遷移ステップを表す．$qseq$, $levels$ はそれぞれ，問合せ移動シーケンス中の領域番号の配列およびレベル番号の配列である．たとえば，$3^{(1)} \rightarrow 14^{(2)} \rightarrow 15^{(2)}$ という遷移シーケンスが問合せとして与えられた場合，$qseq[0] = 3$, $qseq[1] = 14$, $qseq[2] = 15$ であり，$levels[0] = 1$, $levels[1] = levels[2] = 2$ となる．

ある遷移シーケンスの出現回数の推定を求める問合せがユーザから与えられると，まず 2 つの配列 $qseq$ と $levels$ を構築し，est_count($root, qseq, levels, 0$) という関数呼び出しを行う．$root$ はヒストグラムのルートを表し，0 はまず最初は 0 番目の遷移ステップに関する処理を行うことを意味する．

アルゴリズムの説明に移る．このカウント処理では，ルートから 1 ノードずつ辺を辿って，対象となるカウントを有するノードまで探索し，そのカウント値を返す．木の探索処理は再帰呼出しで実現する．ただし，場合によっては 1 つの問合せに対し複数の辺に分岐して探索を行う必要も生じることに注意する．

まず，最も一般的な場合は 8 行目から 22 行目までの部分である．この部分では，$node$ からその子に対し 1 つだけ辺を辿る処理を記述している．8 行目において $levels[step]$ の値が 0

Algorithm 2 est_count($node$, $qseq$, $levels$, $step$)

```
1: 入力  node: 対象ノード，qseq: 問合せの遷移シーケンス
2:       qlevel: 各遷移のレベルを表す配列
3:       step: 現在の処理対象のステップ
4: if (i = 0, ..., n について levels[i] ≡ 0) then
5:   { 問合せ処理が終了 }
6:   return node.count;
7: end if
8: if (levels[step] > 0) then
9:   { 分岐しない場合 }
10:  shift := 2(levels[step] − 1);
11:  c := (qseq[step] >> shift) & 0x3; { 探索する子ノード番号 }
12:  levels[step]−−;
13:  if (node.child[c] ≡ NULL) then
14:    { 探索途中だが辿る辺がない }
15:    count := node.count;
16:    for all i := 0 to m do
17:      count := count / 2^levels[i]; { カウントの推定値を計算 }
18:    end for
19:    return count;
20:  else
21:    return est_count(node.child[c], qseq, levels,
                        (step + 1) mod (n + 1));
22:  end if
23: else
24:  {4 つの子ノードに分岐し，計算結果を集計する }
25:  count := 0;
26:  for all c := 0 to 3 do
27:    if (nocde.child[c] ≢ NULL) then
28:      count += est_count(node.child[c], qseq, levels,
                            (step + 1) mod (n + 1));
29:    end if
30:  end for
31: end if
```

であるか否かにおいて分岐している．このアルゴリズムでは，与えられた問合せをどこまで処理したかを表現するのに $levels$ 配列の値を操作して表現している．11 行目で対応する子ノードの番号が得られると，12 行目で $levels[step]$ の値から 1 を引いているが，直感的には，これは「現在のノードに対応する問合せの該当部分を処理してしまった」ことを表す．典型的な場合では，13 行目の条件が負と判定され，21 行目の再帰呼び出しを実行する．これは，子に対してカウント処理を引き継ぐことを表している．$(step + 1)$ **mod** $(n + 1)$ は，n 番目の遷移を処理した後，次は再び 0 番目の遷移を対象とするための指示である．一方，13 行目が真になるのは，4.2.1 節の (3) の場合のように，問合せに対してヒストグラムの解像度が高くなく，近似的なカウント値を推定する必要がある場合の処理である．

24 行目以降は，8 行目で $levels[step] \equiv 0$ であった場合に相当する．この場合には分岐処理が必要となり，4.2.1 節の (2)(b) の場合に相当する．4 つの辺に対し，est_count() を 4 回再帰呼び出しし，その結果の和をとることでカウントの推定値とする．

5. 忘却を用いたカウント処理

5.1 忘却の概念の導入

前節で提案したアプローチには，データを追加していくだけなので，木が巨大になる可能性がある．また，時間的に移動パターンが変化しない定常的な状況には対応しているが，時間的に移動パターンが変化するような非定常的な状況には適用できない．そこで，木のサイズを一定に保ち，時間とともに移動パターン自体が変化するような非定常的な状況に対応するために忘却の概念を取り入れる．**忘却**（forgetting）とは，過去のデータのカウントを時間とともに低減させようとする考えである．カウントに λ という忘却係数を掛けることで，過去の遷移シーケンスのカウントを小さくできる．経過時間は，そのノードに最後に遷移シーケンスが追加された時刻 t' と，遷移シーケンスを追加する時刻 t の差で表す．以下に忘却を用いた場合のカウントの更新式を示す．

$$count_{\mathrm{new}} = (count_{\mathrm{old}} \times \lambda^{t-t'}) + 1 \quad (0 < \lambda < 1) \qquad (1)$$

λ は忘却の度合いを表すパラメータである．

5.2 データ構造の拡張

先ほどと同様に，遷移シーケンス $a^{(m)} \rightarrow b^{(m)} \rightarrow c^{(m)}$ を時刻 t の時に挿入することを想定する．木を構築する上で，各ノードに最終アクセス時刻を表すタイムスタンプを新たに用意する．空間分割レベル 1 に対し a，b，c それぞれ上位 2 ビットを順に辿り，各ノードのタイムスタンプをもとにカウントを更新し，そのアクセス時刻をタイムスタンプに更新する．続いて空間分割レベル 2 に対し a，b，c それぞれ次の上位 2 ビットを順に辿り，各ノードのタイムスタンプをもとにカウントを更新し，そのアクセス時刻を現在の時刻に更新する．これを空間分割レベル m まで続けていく．

以下に，$m = 2$ の場合について $3^{(2)} \rightarrow 6^{(2)} \rightarrow 9^{(2)}$ を追加する例を示す．カウントの更新処理は，末端の葉のカウントだけを更新するのではない．途中のノードとなる空間分割レベル 1 の 00，01，10，空間分割レベル 2 の 11，10 も更新の対象となる．

そこで，葉のカウントを 4，葉のタイムスタンプを 100 とする．また挿入時のタイムスタンプを 101 とすると，t=101 に挿入後の $3^{(2)} \rightarrow 6^{(2)} \rightarrow 9^{(2)}$ のカウントは式 (1) より，$4 \times \lambda^{101-100} + 1$ となる．図 5 の各ノードにもカウント，タイムスタンプをもっている．

5.3 部分木の削除処理

忘却を導入したことにより，新たな追加がなされない部分木については，そのカウント値は時刻とともに逓減する．カウント値が十分小さくなった部分木を削除することで, 木のサイズをコンパクトに抑えることが可能となる．以下では，基本的な考え方を示す．

式 (1) の $count_{\mathrm{old}} \times \lambda^{t-t'}$ の部分を**逓減項**と呼ぶ．新たな遷移シーケンスの挿入時には，この逓減項を考慮して以下のように木構造を維持管理する．

（1） 逓減項 $count_{\mathrm{old}} \times \lambda^{t-t'} \geqq \epsilon$ のとき

この場合は，4.2 節で述べた手法でノードのカウントを更新

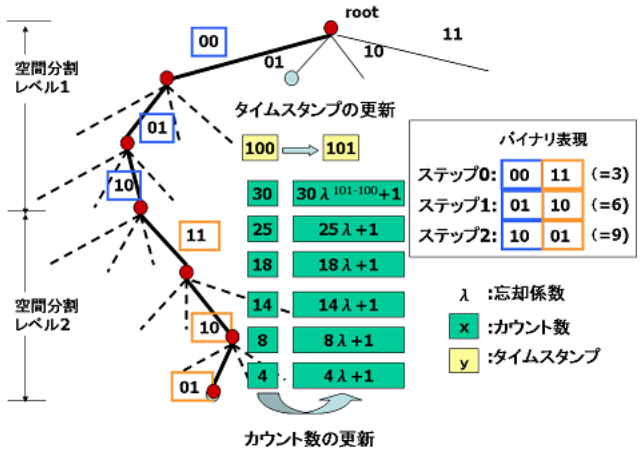

図 5　木構造 ($m = 2$, $t = 101$ に挿入)

する．

（2） 逓減項 $count_{\mathrm{old}} \times \lambda^{t-t'} < \epsilon$ のとき

例えば，$0^{(1)} \rightarrow 1^{(1)} \rightarrow 2^{(1)}$ を考える．図 6 の $t = 101$ 以降，$0^{(1)} \rightarrow 1^{(1)} \rightarrow 2^{(1)}$ を辿る遷移シーケンスが到着せず，$t = 601$ になって，改めて $0^{(1)} \rightarrow 1^{(1)} \rightarrow 2^{(1)}$ を追加したと想定する．また，$\lambda = 0.98, \epsilon = \frac{1}{1000}$ とする．その場合，$t = 101$ のときのカウント数 $18 \times 0.98 + 1$ を用いて，$t = 601$ での逓減項を表すと，$(18 \times 0.98 + 1) \times 0.98^{500}$ となり，閾値 $\frac{1}{1000}$ よりも小さいので，$0^{(1)} \rightarrow 1^{(1)} \rightarrow 2^{(1)}$ のノードより下の部分木をすべて削除する．$0^{(1)} \rightarrow 1^{(1)} \rightarrow 2^{(1)}$ のノードには，カウント 1 を設定し，ノード更新のタイムスタンプを $t = 601$ と再設定する．この様子を図 6 に示す．

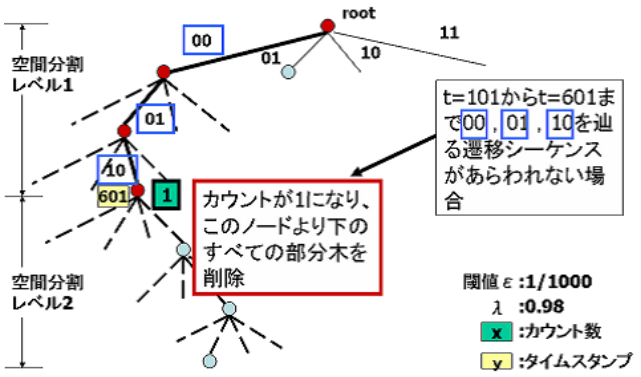

図 6　木構造 ($m = 2$, $t = 601$ に挿入)

忘却を考慮した場合のデータの追加アルゴリズムを**アルゴリズム 3** に示す．基本的な考え方は先に示したアルゴリズム 1 と同様である．忘却を考慮したカウント値の計算と，カウント値が逓減したノードの子ノードを削除する処理が追加された点が異なる．

5.4 問合せ処理

忘却を用いない場合と同様に，場合分けして考える．問合せ処理は 3 節で述べたものと基本的には同様であるが，以下の 2 点が異なる．

（1） 問合せ処理でノードへアクセスするたびごとに，忘却を考慮してそのノードのカウント値の修正を行う．

Algorithm 3 insert_sequence($tseq$)：忘却を導入した場合

入力 $tseq$: 追加する遷移シーケンス
{$root$ はヒストグラムのルートを表す }
{ 現在の時刻を t，対象ノードに最後に遷移シーケンスを挿入した時間を t' とする．}
$node := root$;
$shift := 2(m-1)$;
for all i := 0 to $m-1$ **do**
  for all j := 0 to n **do**
    if ($node.count \times \lambda^{t-t'} \geqq \epsilon$) **then**
      $node.count := (node.count \times \lambda^{t-t'}) + 1$;
      $node$ の子ノードを再帰的に削除する;
    else
      $node.count := 1$;
    end if
    $node.count$ *= $\lambda^{t-t'}$; {t' は対象ノードの更新時刻 }
    $c := (tseq[j] >> shift)$ & 0x3a;
    if ($node.child[c] \equiv NULL$) **then**
      $node.child[c] := new\_node()$
    end if
    $node := node.child[c]$;
  end for
  if ($node.count \times \lambda^{t-t'} \geqq \epsilon$) **then**
    $node.count := (node.count \times \lambda^{t-t'}) + 1$;
    $node$ の子ノードを再帰的に削除する;
  else
    $node.count := 1$;
  end if
  $shift := shift - 2$;
end for

（2） カウント値の修正により，ノードのカウント値が閾値 ϵ 以下になった場合，そのノード以下の部分木を削除する．すなわち，問合せ処理時にも木構造の維持管理を行う．

たとえば，$3^{(2)} \rightarrow 6^{(2)} \rightarrow 9^{(2)}$ を考える．遷移シーケンス挿入と同様に，各ステップをバイナリ表現にして各レベルごとに 2 ビットずつ辿っていく．問合せ時のタイムスタンプと最後に遷移シーケンスが追加されたときのタイムスタンプを使用して，式 (1) より忘却を考慮したカウントを返す．なお，アクセスしたノードに対し，現在の時刻 t とそのノードが前回アクセスされた時刻 t' をもとに，カウント値の修正を行う．カウント値が ϵ より小さい場合，そのノードをルートとする部分木を削除し，カウント値を 0 として扱う．アルゴリズムについては，アルゴリズム 2 から容易に導けるので，ここでは省略する．

6. 考　　察

今後の課題として，実装と実験による性能評価があげられる．[2] などのシミュレータにより生成された移動軌跡データを用いて実験を行い，提案するヒストグラム構築手法の有効性を示す．定常的な場合と非定常的な場合を対象に，特に以下の 2 つの項目について比較評価を行うことを検討している．

（1） 処理時間

- ヒストグラムの更新時間：ヒストグラム構築時間に関連

する値としては，入力データ数，忘却係数，逓減項の閾値，最大空間分割レベルがある．これらの値を変化させたときの構築時間の変化を比較する．忘却の概念を用いた場合には，ヒストグラムへのデータの追加の際にノードの削除が発生する可能性があり，そのためのコストについても評価する必要がある．

- 問合せ処理時間：問合せ処理時間は，ヒストグラムの木構造の大きさや，問合せの種類（例：深いレベルまで探索するか否か，枝分かれした探索処理が求められるか）に依存する．そこで，異なるサイズのヒストグラムに対し，複数の問合せパターンを与えた際の処理時間を比較する．

（2） 正確さ（精度）：ヒストグラムから求めた遷移シーケンスのカウントと実際のデータから得た遷移シーケンスのカウントを比較することで評価を行う．

また，たとえば，本手法をセンサ機器に組み込んで使用する場合には，特にメモリの制約が厳しくなると考えられる．メモリ量の上限がある際に，その限度の範囲内で精度よくヒストグラムを実現することは 1 つの課題である．忘却の概念を導入した手法の場合，各ノードに設定されたカウント値が逓減していくことから，メモリが不足した際にカウント値が大幅に逓減した部分木を効率よく探索し，メモリ領域を回収するアルゴリズムなどについても，今後検討の余地が存在する．

7. ま と め

本稿では，忘却を用いた木構造による移動統計量の推定手法を提案した．今後は忘却係数 λ や閾値 ϵ をどのように設定していくのか，またどのタイミングで木のメンテナンスを実施するか検討する必要がある．そして提案手法による精度向上の具合を検証していく．

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 (C)(2)(16500048)，旭硝子財団研究助成，稲森財団研究助成，文部科学省科学研究費特定領域研究 (2)(16016205) 及び CREST「自律連合型基盤システムの構築」による．

文 献

[1] A. Aboulnaga and S. Chaudhuri. Self tuning histograms: Building histograms without looking at data. In *Proc. SIGMOD*, pp. 181–192, 1999.

[2] T. Brinkhoff. A framework for generating network based moving objects. In *GeoInfomatica*, No. 6(2), pp. 153–180, 2002.

[3] N. Bruno, L. Gravano, and S. Chaudhuri. Stholes: A workload aware multidimensional histogram. In *Proc. SIGMOD*, 2001.

[4] Y. Choi and C. Chung. Selectivity estimation for spatio-temporal queries to moving objects. In *Proc. ACM SIGMOD*, pp. 440–451, 2002.

[5] Ines Fernando, Vega Lopezd, Richard T. Snodgrass, and Bongki Moon. Spatiotemporal agregate computation: A survey. Technical report, A TIMECENTER Technical Report, January 2004.

[6] P. Gibbons, Y. Mattias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proc. VLDB*, pp. 466–475, 1997.

[7] A. Gilbert, S. Guha, P. Indyk, Y. Kotadis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *STOC*, 2002.

[8] A. Gilbert, Y. Kotadis, S. Muthkrishnan, and M. Strauss. Surfing wavelets on streams: One pass summaries for approximate aggregate queries. In *VLDB*, pp. 79–88, 2001.

[9] D. Gunopulos, G. Kollios, V. Tsotras, and C. Domeniconi. Approximating range queries over real attributes. In *Proc. ACM SIGMOD*, 2000.

[10] Yannis Ioannidis. The history of histograms (abridged). In *Proc. VLDB*, 2004.

[11] 岸浩史, 田名部淳, 河野浩之. Σ-tree による時空間 OLAP 技術の交通データへの適用. データ工学ワークショップ (DEWS2002), 2002.

[12] J. Lee, D. Kim, and C. Chung. Multidimensional selectivity estimation using compressed histogram information. In *Proc. SIGMOD*, pp. 205–214, 1999.

[13] Y. Mattias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *Proc. VLDB*, pp. 101–111, 2000.

[14] V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proc. VLDB*, pp. 486–495, 1997.

[15] Shanshi Shekhar and Sanjay Chawla. *Spatial Databases*. Prentice Hall, 2002.

[16] Y. Tao, J. Sun, and D. Papadias. Selectivity estimation for predictive spatio-temporal queries. In *Proc. ICDE*, 2003.

[17] 塚本祐一, 石川佳治, 北川博之. 移動軌跡ストリームからの移動統計量推定のための動的ヒストグラム構築手法について. 電子情報通信学会データ工学ワークショップ (DEWS 2004), March 2004.

[18] G. J. G. Upton and B. Fingleton. *Spatial Data Analysis by Example, Volume II: Categorical and Directional Data*. John Wiley & Sons, 1989.

[19] J. Vitter and M. Wang. Approximate computation of multidimensional aggregates on sparse data using wavelets. In *Proc. SIGMOD*, pp. 193–204, 1999.

[20] Y. Wu, D. Agrawal, and A. E. Abbadi. Applying the golden rule of sampling for selectivity estimation. In *Proc. SIGMOD*, 2001.