RDBを用いた移動履歴からの移動パターン問合せ処理手法

堀田　孝司†　　石川　佳治††　　眞野　将徳†

† 名古屋大学大学院情報科学研究科　〒 464–8601 名古屋市千種区不老町
†† 名古屋大学情報基盤センター　〒 464–8601 名古屋市千種区不老町
国立情報学研究所　〒 101–8430 東京都千代田区一ツ橋 2–1–2
E-mail: †{hotta,mano}@db.itc.nagoya-u.ac.jp, ††ishikawa@itc.nagoya-u.ac.jp

あらまし　本稿では，センサなどによって得られた位置情報を RDB に蓄積し、その蓄積された位置情報からの移動パターン問合せを行うための問合せ処理手法を提案する．移動パターンを決定性有限オートマトン用いて表現し，そのオートマトンから移動パターン問合せを表現する SQL に変換する．本手法により，RDBMS のパワーを活かして，柔軟なイベント問合せを実現しながら，実装コストを抑えたイベント処理エンジンを実現できる．また，本手法によって生成された SQL を RDBMS 上で実行し，本手法の有効性を確認した．

キーワード　移動パターン問合せ，移動履歴，RDBMS

Moving Pattern Query Processing from Location Histories based on Relational Database

Koji HOTTA†, Yoshiharu ISHIKAWA††, and Masanori MANO†

† Graduate School of Information Science, Nagoya University　Furo-cho, Chikusa-ku, Nagoya, 464–8601 Japan
†† Information Technology Center, Nagoya University　Furo-cho, Chikusa-ku, Nagoya, 464–8601 Japan
National Institute of Informatics　2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430 Japan
E-mail: †{hotta,mano}@db.itc.nagoya-u.ac.jp, ††ishikawa@itc.nagoya-u.ac.jp

1. はじめに

近年，センシングデバイスの高性能化，小型化，低コスト化に伴い，様々な環境下において位置情報を容易にかつ大量に取得できるようになった．そのため，これらのセンシングされた大量の位置情報からの複雑なイベント問合せの処理要求が高まっている．このようなセンシングされたデータからイベント問合せを実現するために，ストリームデータに対するイベント問合せの研究が数多くなされている [1], [4], [5], [7].

ストリームデータ処理においては，事前に登録したクエリによりイベント問合せを実行するが，データを永続化しないことが一般的である．しかし，ストリームデータを保持することは，事後検索や過去のデータとの比較が可能となるため有用である．そこで，データベースとの連携が可能なシステムの提案 [6], [12] や，ストリームデータの DB へのアーカイブ処理に関する研究も行われている [10], [11].

履歴データに対するイベント問合せの処理を実現するには，イベント問合せ処理エンジンを独自に実装することが一般的であるが，独自のイベント問合せ処理エンジンでは，フルスクラッチで実装するためにコストが高くなってしまう．さらに，条件の追加や集約問合せなどへの拡張，イベント問合せの最適化なども必要となるため，さらなる実装コストを要し，柔軟性にも欠ける面がある．

このような背景から，センサデータなどを RDB に蓄積し，その蓄積されたデータからのイベント問合せについて研究を行っている．RDBMS をイベント問合せエンジンに用いることで，RDBMS のパワーを活かしながら柔軟性のあるイベント問合せを実現し，実装コストを抑えることも可能である．本研究ではイベント問合せとして，指定された移動パターンに合致するものを抽出する移動パターン問合せを検討する．

本稿の構成は以下の通りである．まず，2. では関連研究について触れる．3. では移動パターン問合せについて述べ，その後 4. で移動パターン問合せを実現するフレームワークについて説明する．5. では，移動パターン問合せの処理について述べ，6. で本フレームワークによって生成された SQL を用いて評価実験を行い，最後に 7. でまとめと今後の課題について述べる．

2. 関連研究

情報爆発時代の到来に伴い，増加したデータをリアルタイム処理するためのデータストリームに対するイベント問合せの研究が盛んに行われている．SASE/SASE+ [2], [7]，Cayuga [1]，ZStream [5]，Lahar [4] などが，ストリームデータに対する処理要求を実現する基盤システムとしてあげられ，それらの研究開発が進められている．

SASE/SASE+，Cayuga はともにオートマトン型のプランを用いたイベント問合せ処理システムである．Cayuga は，複数のイベント問合せを同時に効率よく処理することができ，SASE/SASE+ではスケーラビリティを維持できる問合せ処理システムとなっている．ZStream はオートマトンの代わりに木構造型の問合せプランを用いたイベント問合せシステムである．木構造を用いることにより，ストリームの性質変化に合わせた動的かつ適応的なプラン選択が可能となる．これらのイベント問合せ処理システムでは，ストリームデータを対象としたものであり，履歴データを対象としたものではない．

Lahar は，曖昧な位置情報に基づいたイベント問合せシステムである．センサデータの欠落，誤差に対応させるために，RFID などによって得られた位置情報から存在確率と遷移確率を推定する．そして，その推定された確率的位置情報を用いて確率的なイベント問合せを実現している．Lahar もオートマトン型の問合せプランを用いたイベント問合せ処理システムであり，履歴情報に対しての確率的イベント問合せを実現している．

上記にあげた全てのストリームデータ処理システムは，独自のイベント処理エンジンで実装されている．そのため，実装コストが高くなり，柔軟性に欠けてしまう．本研究の手法では，RDBMS を利用することにより，実装コストを抑えて，かつ，柔軟性の高いイベント問合せ処理システムを実現できる．

3. 移動パターン問合せ

3.1 研究の想定

本研究で考える移動パターン問合せは，以下の流れで問合せが行われることを想定している．

（1） センサによる位置情報の取得: 移動オブジェクトの位置情報をセンサにより取得する．特定のセンサデバイスに依存せず，屋内外問わない．センシングされる時間間隔はアプリケーションに応じて変更する．

（2） 位置情報のデータベースへの格納: センシングで得られた位置情報を RDBMS に保存する．位置情報は，本提案手法で移動パターン問合せを実行する際に利用しやすいリレーションで保持する．この際に，得られた位置情報や時刻はそのままデータベースに追加する．

（3） 問合せ実行: 移動パターン問合せの実行のために，問合せ発行者は問合せ言語で移動パターン問合せを記述し，データベースに保持された位置情報に対して，移動パターン問合せを行う．

本研究における提案手法と従来手法との違いを図 1 に示す．従来手法におけるイベント処理システムは，本研究においては問合せ生成エンジンと RDBMS に置き換わる．本手法を利用することのメリットは以下の点である．

- ストリームの一括処理: 従来の手法では，複数のストリームに対してイベント問合せを実行する場合，ストリーム処理のプロセスを複数用意し，各プロセスにおいてイベント問合せを行う必要がある．本手法では，結合処理などの演算を行うことで，複数のストリームデータに対して一括処理を行うことができる．
- 柔軟な問合せが可能: RDBMS によって，結合演算，集約演算などが利用できるため，イベント問合せの条件追加，問合せの集約問合せなどの拡張も容易に可能である．
- 最適化が容易: RDBMS の機能を利用することで，問合せ最適化が可能である．
- 実装コストが低い: 本手法を利用することにより，イベント問合せを実行する SQL を作成する問合せプラン生成エンジンのみ実装だけとなるので，実装コストが低い．

また，商用 DBMS，オープンソースの DBMS でも実現可能であるので，現実的な手法といえる．

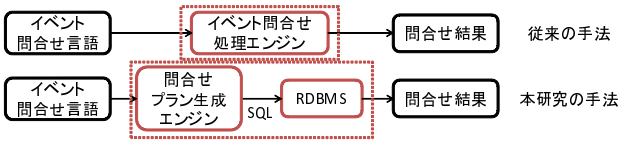

図 1 従来手法と提案手法

3.2 移動パターン問合せの定義

本研究における移動パターンは領域で構成され，その領域の順序のこと移動パターンとする．例として，図 2 に示した移動パターン P を考える．移動パターンの最初の領域を出発領域，最後の領域を到着領域と呼ぶ．P の場合は，O1 が出発領域，H2，O2 が到着領域となる．その間の領域を経由領域と呼ぶ．P の場合，O2 が経由領域となる．移動パターンにおけるどの領域に関して，任意の場所に移動したといったように指定しないことも許すが，出発領域，経由領域，到着領域のいずれかは指定する必要がある．今回図 2 では，枝分かれした移動パターンを示したが，その他にも特定領域に滞在する，特定領域を行き来するパターンなどが考えられる．

本研究における移動パターン問合せは，上記の定義を満たした移動パターンとパターンへの追加条件で構成される．つまり，移動パターン問合せは，追加条件を満たし，かつ，指定された移動パターンに合致するものを抽出する問合せとなる．

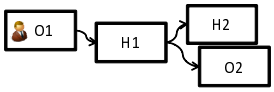

図 2 移動パターンの例 P

3.3 パターンへの追加条件

パターンへの追加条件は，時刻条件，時間条件，属性条件の 3 つがある．

時刻条件は，移動パターンが行われた時刻の指定，移動パターンの特定の領域についた時刻を指定といった条件である．この条件により，「13 時から 14 時の間に移動パターン P に基づいて移動したオブジェクトを提示せよ」といった問合せが可能となる．時間条件は，移動パターンに基づいて移動するのにかかった時間の指定や，特定の領域に滞在した時間の指定といった条件である．この条件により，「出発領域を出発してからから 10 分以内の間に，移動パターン P に基づいて移動したオブジェクトを提示せよ」といった問合せが可能になる．属性条件は，データベースに保持されているオブジェクトの属性を指定した条件である．属性に関するデータが人の年齢や性別等であ

れば,「20 歳代の男性で，移動パターン P に基づいて移動した男性を提示せよ」といった問合せが可能となる．

3.4 移動パターン問合せの種類

本研究で扱う移動パターン問合せは，大きく分ければ 2 つの問合せに分類できる．1 つ目は移動パターンの検出を目的とした問合せである．この問合せは，さらに 2 つに分類でき，単独のオブジェクトに対する移動パターン問合せと，複数のオブジェクトに対する移動パターン問合せに分けられる．前者は,「移動パターン P に基づいて移動したオブジェクトを提示せよ」といった問合せであり，後者は,「移動パターン P に基づいて，2 つのオブジェクトが一緒に行動したのはいつか」といった問合せである．もう一方は，問合せ結果に集約を要する問合せである．この問合せの例としては,「移動パターン P に基づいて移動したオブジェクトの数はいくつか」といった問合せである．

本稿では，単独のオブジェクトが特定の移動パターンに基づいた移動の検出を目的とした問合せについて検討することとし，今後は単独のオブジェクトに対する問合せのみについて述べる．

4. 移動パターン問合せ処理のフレームワーク

4.1 フレームワークの概要

3.1 で述べた想定をもとに，移動パターン問合せを実現するフレームワークの概要について述べる．フレームワークの全体像を図 3 に示す．

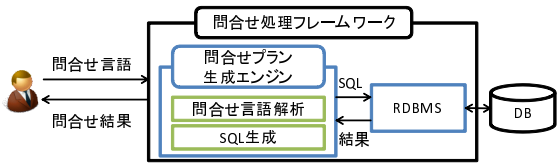

図 3 移動パターン問合せ処理のフレームワーク

移動パターン問合せ処理のフレームワークは，問合せプラン生成エンジンと RDBMS で構成する．また，移動パターン問合せを記述する問合せ言語をユーザに提供する．それぞれの構成要素について説明する．

- 問合せ言語：移動パターン問合せを記述するための問合せ言語．本稿では言語の提案は行わない．移動パターンを容易に記述できる問合せ言語を想定している．
- 問合せプラン生成エンジン：問合せ言語を解析し，SQL に変換するエンジンである．問合せプラン生成エンジンは，問合せ言語解析モジュールと SQL 生成モジュールによって構成される．
  - 問合せ言語解析モジュール：問合せ言語を解析する．問合せに記述された移動パターンを表現した決定性有限オートマトン，パターンへの条件，問合せ内容の 3 つを解析結果として，SQL 生成モジュールに渡す．本稿では，問合せ言語について言及しないため，このモジュールの詳細は省略する．
  - SQL 生成モジュール：問合せ言語解析モジュールから得られた解析結果から，移動パターン問合せを実現する SQL を生成するモジュール．決定性有限オートマトンから SQL 生成への変換には，XPath を SQL に変換する手法 [3] を応用する．
- RDBMS とデータベース：RDBMS は，問合せプラン生成エンジンによって作成された SQL を受け取り，イベント問合せを実行し，問合せ結果をユーザに返す．データベースには，オブジェクトの位置情報やオブジェクトの属性などを保持している．

4.2 保持するデータ

データベースには，オブジェクトの位置情報と属性情報を保持する．属性情報はアプリケーションに必要となる情報を自由に保持する．例えば，人であれば，性別，年齢といったようなその人特有のデータといったものを保持する．位置情報については，具体例として，表 1 に位置情報のリレーションの例を示す．

表 1 位置情報を保持したリレーションの例

tid	oid	time	loc	next	prev
T17	A	13	O1	T97	T1
T18	B	13	O2	T98	T2
T19	C	13	H2	T99	null
⋮	⋮	⋮	⋮	⋮	⋮
T97	A	14	H1	T156	T17
⋮	⋮	⋮	⋮	⋮	⋮

表 1 は，各オブジェクトがどの地点に存在していたかを表現している．例えば，時刻 13 においてオブジェクト A は O1(オフィス 1) にいて，時刻 14 においては H1(廊下 1) に存在していたことを表している．位置情報のリレーションの各属性は 6 つの属性を持つ．

- tid: タプル ID．ユニークな値が与えられる．
- oid: オブジェクトの ID．各オブジェクトに対してユニークな ID が与えられる．
- time: オブジェクトが位置 loc に存在する時刻．
- loc: オブジェクトが時刻 time の時に存在する位置．
- next: オブジェクトの次の時刻における位置を表しているタプル ID を指す．
- prev: オブジェクトの前の時刻における位置を表しているタプル ID を指す．

位置情報のリレーションスキーマの特徴は，全ての位置情報に次の時刻のタプル ID と前の時刻のタプル ID が与えられていることである．今回の例では，位置情報を予め領域ごとに分けておき，その領域に名前を与えていたが，位置情報の表現方法は様々な方法がある．例えば，全ての位置がグリッドで分割された上での表現，座標上の点での表現などが考えられる．

5. 移動パターン問合せ処理

解析結果から移動パターン問合せを表す SQL への変換処理について述べる．今後，説明のために，位置情報を格納したリレーションを R_L，オブジェクトの属性を格納したリレーションを R_A とする．

5.1 解析結果の定式化

5.1.1 決定性有限オートマトンの定式化

移動パターンを表現する決定性有限オートマトンは，以下の

定義で与える．

［定義 1］（移動パターンを表現する決定性有限オートマトン）
移動パターンを表現する有限オートマトンは，5 つ組 $A = (Q, \Sigma, \delta, S_0, F)$ で表現される．それぞれの要素は，以下のとおりである．

- Q: 状態の有限集合．移動パターンを表す状態数が n 個ある場合，$Q = S_0, S_1, \cdots, S_{n-1}, S_n$ となる．
- Σ: 入力の有限集合．遷移関数の入力は領域である．また，領域の全体集合を U とし，移動パターンに利用され，特定されている領域全体を U_P とする．また，特定されていない領域を $*$ と表すこととする．つまり，$* = U - U_p$ である．
- δ: 遷移関数．領域の集合を L と定義すると，$\delta(S_i, L) = S_j$ は，状態 S_i にいる時に領域集合 L の入力を受け取ると S_j へ遷移するということを表す．
- $S_0 \in Q$: 初期状態．
- $F \subset Q$: 受理状態の集合．

以下では，上記に述べた決定性有限オートマトンのことを単にオートマトンと呼ぶこととする．

5.1.2 パターンへの追加条件の定式化

まず，解析結果で得られる，時刻条件，時間条件，属性条件の 3 つの条件について，定式化する．

［定義 2］（時刻条件） 解析結果で得られた時刻条件 $t_{interval}$ は，3 つ組 $t_{interval} = (S_s, S_e, [t_s, t_e])$ で表現される．それぞれの要素は，以下のとおりである．

- S_s: 時刻条件のかかる範囲内の中でのオートマトンの開始状態．$S_s \in (Q - S_0)$ である．
- S_e: 時刻条件のかかる範囲内の中でのオートマトンの最終状態．$S_e \in (Q - S_0)$ である．
- $[t_s, t_e]$: 状態 S_s から始まり，状態 S_e に遷移するまでのオートマトンが行われた時区間．

［定義 3］（時間条件） 解析結果で得られた時間条件 $t_{duration}$ は，3 つ組 $t_{duration} = (S_s, S_e, t)$ で表現される．それぞれの要素は，以下のとおりである．

- S_s: 時間条件のかかる範囲内の中でのオートマトンの開始状態．$S_s \in (Q - S_0)$ である．
- S_e: 時間条件のかかる範囲内の中でのオートマトンの最終状態．$S_e \in (Q - S_0)$ である．
- t: 状態 S_s から始まり，状態 S_e に遷移するまでのオートマトンに要した時間．

［定義 4］（属性条件） 解析結果で得られた属性条件 $attr$ は，2 つ組 $attr = (V, c)$ で表現される．それぞれの要素は，以下のとおりである．

- V: 属性条件に利用する R_A の属性名．
- c: 属性条件に利用する V の値．

さらに，以上の定義を利用して，解析結果で得られたパターンへの追加条件の定式化は，以下のとおりとなる．

［定義 5］（パターンへの追加条件） 解析結果で得られたパターンへの追加条件は，時刻条件の集合 $T_{interval}$，時間条件の集合 $T_{duration}$，属性条件の集合 $Attr$ での 3 つ組 $C = (T_{interval}, T_{duration}, Attr)$ で表現される．

この表記によって，複数の条件があった場合の表記も可能になる．

5.1.3 今回検討する移動パターン

本稿では，今回は移動パターン問合せとして，現実的に実行されうる移動パターンに限定し，基本的な移動パターンを変換するための問合せ変換アルゴリズムを示す．そのため，今回検討する移動パターンは，5.1.1 で表現されたオートマトンに対して制約を設ける．

［制約 1］（状態遷移に対する制約） 任意の 3 つの状態 S_i, S_j, S_k $(i \neq j, j \neq k, i \neq k)$ において，$\delta(S_i, L) = S_k, \delta(S_j, L') = S_k$ となる遷移が存在しない．

この制約を言い換えれば，自分の状態を除く他の状態からの遷移は 1 つに限るということである．遷移先と遷移後の状態が同じである遷移は，この制約に反しないことを注意されたい．この制約により，複数の特定領域を不特定回ループするオートマトン，2 つの状態から 1 つの状態に流入する遷移があるオートマトンは考えない．今回は，基本的な移動パターン問合せが実現できることを示すために，現実的に実行されうる基本的な移動パターンに限定している．

5.2 変換アルゴリズム

SQL 生成モジュールで行われる変換処理について述べる．まず本研究で利用する，XPath から SQL に変換する手法 [3] について説明する．

5.2.1 XPath における SQL 変換

XML を関係データベースで扱う際には，XML の木構造を表現するため，リレーションの属性に From，To を記述する．From は自分自身の親ノードを指し，To は自分の子ノードを指す．この属性を利用することで，SQL が XML の木構造を辿ることが可能となる．さらに，[3] では，最小不動点 (Least FixPoint; LFP) 演算子を利用して再帰のある XPath を SQL に変換している．LFP 演算子は，SQL99 にて導入された再帰 SQL を記述する構文 *with recursive* を利用し，一つのリレーションに対して再帰的に SQL を実行する操作である．LFP 演算子を利用することによって，XML の再帰的な木構造を探索することができる．

入力されたリレーションを R とすると，LFP Operator $\Phi(R)$ の定義は以下のとおりである．

$$R^0 \leftarrow R$$
$$R^i \leftarrow R^{i-1} \cup (R^{i-1} \bowtie_C R^0) \quad (1)$$

C は結合の際に利用する Boolean 型の条件である．この構文は，Oracle や DB2 などの多くの商用 RDBMS，オープンソース RDBMS の PostgreSQL にもおいてもサポートされている．

本研究においても，リレーションに次の時刻の位置情報を保持しているタプル ID を指す next と前の時刻の位置情報を保持しているタプル ID を指す prev がある．これらの属性を利用することで，RDBMS を用いてオブジェクトの移動シーケンスを辿ることができる．

5.2.2 本研究への適用

本研究においては，この LFP 演算子はそのまま利用する

には不便な点がある．5.1 で述べたオートマトンに対して，LFP 演算子を移動パターンに利用すると，特定の領域に対して移動し，そのまま滞在するという遷移関数 $\delta(S_i, L) = S_{i+1}$ と $\delta(S_{i+1}, L) = S_{i+1}$ を表す．しかしながら，特定の領域に移動し，その後，直前の領域とは違う領域に滞在する，もしくは，滞在しないで次の領域に向かう，という遷移関数 $\delta(S_i, L) = S_{i+1}, \delta(S_{i+1}, L') = S_{i+1}$ の表現には，LFP 演算子は扱えない．そこで本研究では，LFP 演算子を拡張した演算子 $\Phi'(R_s, R_r)$ を定義する．R_s は初期のリレーションに利用するものであり，R_r は再帰的に利用するリレーションである．式 (1) より，$\Phi'(R_s, R_r)$ は以下のように定義できる．

$$R^0 \leftarrow R_s$$
$$R^i \leftarrow R^{i-1} \cup (R^{i-1} \bowtie_C R_r) \qquad (2)$$

$R_s = R_r$ とすることで，元々の LFP 演算子も実現できる．変換処理のアルゴリズムでは，演算子 $\Phi'(R_s, R_r)$ を利用し，移動オブジェクトが同じ場所に滞在するといった再帰的に繰り返される移動パターンの変換を行う．

5.3 変換アルゴリズム

アゴリズムの詳細を述べる前に，まず概要を説明する．図 4 は変換アルゴリズムの流れを示したものである．

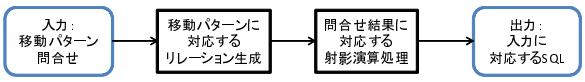

図 4　変換アルゴリズムの流れ

変換アルゴリズムは，2 つのステップで SQL への変換を実行する．

（1） 移動パターンに対応するリレーションの生成：解析結果で得られた移動パターンとパターンへの追加条件を利用して，問合せ結果に必要なリレーションを生成する．つまり，このステップは SQL の FROM 句と WHERE 句における記述の処理に相当する．

（2） 問合せ結果に対応する射影演算処理：(1) で得られたリレーションと解析結果で得られた問合せ内容から，問合せ結果を出力するための射影演算の処理を行う．つまり，このステップは SQL の SELECT 句における記述の処理に相当する．

上記の 2 つのステップにおいて，移動パターン問合せの変換が終了する．移動パターン問合せを実現する上では，(1) のステップが重要である．(1) のステップでどのようにリレーションを結合演算するか，また，パターンへの追加条件に対してどのように選択演算をするかが，移動パターン問合せを SQL へ変換する手法としてのキーとなる．(2) のステップは，(1) のステップで得られたリレーションに対して，射影演算処理を行うだけであり，本質的には重要ではない．そのため，本稿では (2) のステップにおける処理に関しての説明は省略する．これからは，ステップ (1) における移動パターンに対応するリレーション生成の流れについて説明する．

図 5 はステップ (1) のフローチャートである．基本的に，オートマトンのパターンの開始状態の次の状態から順に 1 つずつ処理し，全状態の処理が終わったら移動パターンに対応するリレーションを出力するといった流れとなる．ただ，次の状態を処理する前に，オートマトンに分岐があるかをチェックする．分岐がある場合には，分岐先の各オートマトンの処理に移る．この処理においては，分岐後の各オートマトンに対して，リレーション生成のためのアルゴリズムを再帰的に実行する処理となる．

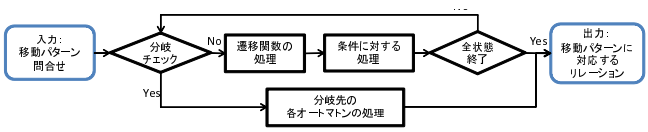

図 5　移動パターンに対応するリレーション生成のフローチャート

5.3.1 遷移関数の入力に対する処理

遷移関数の処理では，遷移関数に入力される値に応じて処理が行われる．ここで，遷移関数の入力 L に対応するリレーションを得る関数を $input(L)$ とし，入力 L に対して，関数 $input(L)$ の処理について，例を上げて説明する．

[例 1]（特定領域への移動） $L = l$ のように，特定の領域 l に移動という入力に対しては，

$$input(L) = \sigma_{loc \in l} R_L$$

となる．本研究において，$p \in L$ は位置 p は領域 L の境界上を含む内部に存在すると定義する．つまり，この処理は位置情報を保持したリレーションに対して，領域 l の内部に存在した時点のタプルを選択するという処理になる．

[例 2]（複数の特定領域への移動） $L = l, l'$ のように，領域 l または l' に移動という入力に対しては，

$$input(L) = \sigma_{loc \in l \vee loc \in l'} R_L$$

となる．この処理は位置情報を保持したリレーションに対して，領域 l または l' の内部に存在した時点のタプルを選択するという処理になる．

[例 3]（任意領域への移動） $L = U$ のように，任意の領域への移動という入力に対しては，

$$input(L) = R_L$$

となる．

[例 4]（特定領域以外への移動） $L = U - l$ のように，特定の領域 l 以外に移動という入力に対しては，

$$input(L) = R_L - \sigma_{loc \in l} R_L$$

となる．つまり，全体集合を表す位置情報を保持したリレーションから領域 l の内部に存在した時点のタプルを選択したリレーションの差を取る処理となる．

関数 $input$ の動作について，基本的な 4 つの入力に対しての例を説明した．関数 $input$ は，上記以外の入力の場合にも，入力に対応した同様な操作を行うことで，入力に対応するリレーションを取得する．

5.3.2 パターンへの追加条件に対する処理

遷移関数の入力の処理が終わったあと，パターンへの追加条件に対する処理を行う．パターンへの追加条件に対する処理を行う関数を $condition(R, C)$ とする．ただし，遷移関数の処理

を終えて得られた状態のリレーション R，パターンへの追加条件 C である．関数 *condition* には条件処理を必要としない状態のリレーションの入力も許す．その場合は処理されずに，入力された R が関数の出力となる．

［処理 1］（時刻条件） 入力 C に，$t_{interval} = (S_s, S_e, [t_s, t_e])$ が時刻条件として含まれているのならば，S_s に対応するリレーションを R_s，S_e に対応するリレーションを R_e とすると，状態 S_s の処理の際に，

$$condition(R_s, C) = \sigma_{time \geqq t_s} R_s$$

とし，状態 S_e の処理の際に，

$$condition(R_e, C) = \sigma_{time \leqq t_e} R_e$$

となる．

［処理 2］（時間条件） 入力 C に，$t_{duration} = (S_s, S_e, t)$ が時間条件として含まれているのならば，S_s に対応するリレーションを R_s，S_s に対応するリレーションを R_e とすると，状態 S_s の処理の際に，

$$condition(R_s, C) = R_s$$

とし，状態 S_e の処理の際に，

$$condition(R_e, C) = \sigma_{time - R_s.time \leqq t} R_e$$

となる．

S_s に対応する処理は，開始時点の時刻を取得だけで，リレーションに対しての処理はない．

［処理 3］（属性条件） 入力 C に，$attr = (V, c)$ が属性条件として含まれているのならば，移動パターンの開始状態 S_1 に対応するリレーションを R_1 とすると，状態 S_1 の処理の際に，

$$condition(R_1, C) = R_1 \bowtie_{R_1.oid = R'_A.oid} R'_A$$

ただし，$R'_A = \sigma_{R_A.V=c} R_A$ である

となる．

5.3.3 移動パターンに対応するリレーションの生成

5.1.3 で述べたオートマトンに対応するリレーションを生成するアルゴリズムに示す．

入力はオートマトンとパターンへの追加条件であり，入力の移動パターンに対応するリレーションを表現したリレーショナル代数である．まず，3 行目にあるように，現在の状態から遷移できる次の状態の数をチェックする．遷移先の状態が 2 つ以上であれば，4 行目〜10 行目の処理へと移る．今までに得られた結果を一時テーブル R_t に保持し，それぞれの分岐先から始まるオートマトンの処理を再帰的に行う．その後，分岐先の各オートマトンの結果と一時テーブルを結合し，さらにそれらの和をとり，変換処理の結果として返す．

次の状態数が 1 つである場合は，11 行目からの処理に移る．ここからが 1 つの状態に対しての具体的な処理内容になる．まず，12 行目〜17 行目が，次状態へ遷移する遷移関数の処理であり，18 行目が次状態に関する追加条件の処理である．次状態へ遷移する遷移関数の処理では 2 つの処理に分かれる．次状態において滞在する遷移がある場合 (12，13 行目) とそうでない場合 (15 行目) である．どちらの場合においても，まず関数 *input* を利用し，遷移に対するリレーションを取得する．滞在しない場合は，関数 *input* を利用して得られたリレーションが，状態に対応したリレーションとなる．同じ状態に滞在する場合は，$\Phi'(R_s, R_r)$ 演算子を利用することで，状態に対応したリレーションを得る．

17 行目において，処理中の状態に必要な追加条件の処理を関数 *condition* を利用することで，一つの状態に対応するリレーションが生成される．18 行目において，前までに処理済みの状態との結合処理を行い，次の状態の処理へと移る．

アルゴリズム 移動パターンに対応するリレーションの生成

```
1: procedure MPQ2SQL(A, C)
2:     for i = 0 to n do
3:         if ∃j,k.(δ(S_i, L) = S_j, δ(S_i, L') = S_k, j ≠ k) then   ▷ 次の状態が 2 つ以上の時
4:             一時テーブル R_t に今までの結果を保持
5:             for l = 1 to n_b do                                  ▷ n_b(> 2) は分岐数
6:                 R'_j = MPQ2SQL(A_j, C_{A_j})                     ▷ A_j は j 番目の分岐から始まるオートマトン，C_{A_j} はオートマトン A_j への追加条件
7:             end for
8:             Q := R_t ⋈_{R_t.next=R'_1.tid} R'∪···∪R_t ⋈_{R_t.next=R'_m.tid} R'_m
9:             return Q
10:        else                                                     ▷ 次状態への遷移が 1 つの場合
11:            if ∃i.(δ(S_i, L) = S_j, δ(S_j, L') = S_j) then      ▷ 次状態に滞在する遷移がある場合
12:                R_s = input(L), R_r = input(L')                  ▷ 遷移関数の入力に対する処理
13:                R_j = Φ'(R_s, R_r)
14:            else                                                 ▷ 次状態に滞在する遷移がない場合
15:                R_j = input(L)
16:            end if
17:            R_j := condition(R_j, C)                             ▷ 追加条件に対する処理
18:            Q := ··· R_i ⋈_{R_i.next=R_j.tid} R_j                ▷ 次状態と前状態との結合
19:        end if
20:    end for
21:    return Q
22: end procedure
```

6. 評価実験

今回の実験では，本フレームワークによって生成される移動パターン問合せが実用的に利用できることを，変換処理を経て生成される SQL を用いて評価する．

6.1 実験環境

生成される SQL を実行するために，RDBMS に PostgreSQL を使用する．PostgreSQL は再帰 SQL を記述する構文 *with recursive* を含む標準 SQL の大部分をサポートしている．また，幾何データ型など豊富な型が用意されているため，本実験を行うのには都合がよい．

今回使用するデータは，2007 年 4 月から 2009 年 8 月にかけ

て Microsoft Research Asia が行った GeoLife Project で収集した GPS データ [8], [9] を利用する．全ユーザ数は 155 人であり，ファイルサイズは 1.44GB，レコード数は 23,794,008 件となった．使用されている GPS デバイスは様々であり，またデバイスに応じてログの取得も異なっている．データの 95%が 2〜5 秒間隔，または 5〜10 メートル間隔で取得されており，残りの 5%はこれよりも荒くデータを取得したものである．データは，ユーザ毎にフォルダに保存されており，さらにデータ取得した日毎にファイル分けされ，CSV ファイルと同様の形式で保存されている．今回はそのファイルを一つのストリームとみなし，データベースに入れた．実験は， OS が windows 7，CPU 2.67GHz，メモリ 8GB，PostgreSQL 9.0 をインストールしたマシン上で行った．今回は，シンプルな問合せにおいて，移動パターン問合せが実現できるかを検証する．そのため，属性情報についてはなしとした．

6.2 基本的な移動パターン問合せの処理時間の測定

まず，図 6 に示した 4 種類のオートマトンを用いた基本的な移動パターン問合せの処理時間を測定した．問合せ 1 は，3 つの領域を連続して移動するパターンを表している．問合せ 2 は，ある領域への滞在を含む移動パターンである．問合せ 3 は，任意の位置への移動を含む移動パターンである．図の問合せ 3 における＊は，領域の全体集合を表している．問合せ 4 は，分岐を含む移動パターンである．

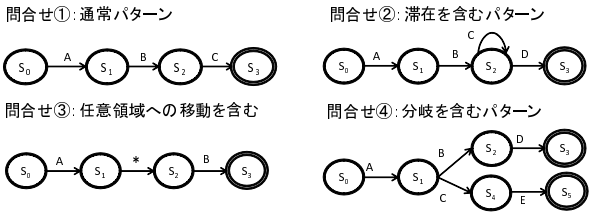

図 6　実験に利用する移動パターン

今回の実験は，各 15 個の移動パターンを用意して，それらの処理時間の平均を調べることとした．処理時間は，移動パターンの領域内に含まれるレコード数，キャッシュの有無，問合せプランの 3 つに大きく依存すると考えられる．そこで今回の実験では，以下に述べる状況で実験を行った．移動パターンの領域内に含まれるレコード数は 10000 程度とした．10000 程度のレコード数とは，位置情報が疎，または密に分布した領域ではなく，その中間程度に位置情報が分布した領域である．同じ移動パターン問合せでは，全て同じ問合せプランを使用するものとした．キャッシュについては，cold 状態，hot 状態の 2 つの状態において問合せを実行した．cold 状態での実行は，ディスクのキャッシュや PostgreSQL が保持しているキャッシュが全くない状態の時に問合せを実行することであり，hot 状態での実行は，問合せをすでに行なっており，キャッシュがある状態から問合せを実行することである．さらに，位置情報には GiST(Generalized Search Tree) を用いて，タプル ID には B-tree を用いて，索引を作成している．

この結果として，hot 状態の平均処理時間，cold 状態の平均処理時間，問合せの平均処理時間を図 7 に示す．

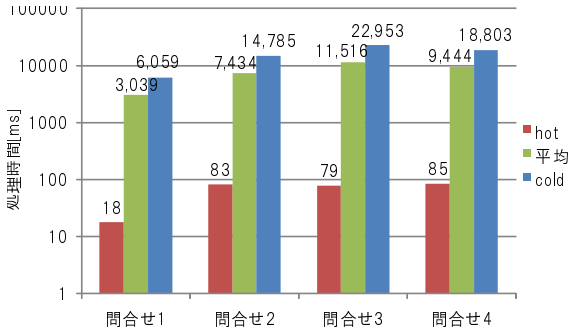

図 7　基本的な移動パターン問合せの処理時間

問合せ 1 は平均 3 秒であり，その他の問合せは平均処理時間は約 10 秒付近であった．若干処理に時間がかかるが，平均 10 秒前後で処理が実現できれば，実用的に利用できる範囲内であると考える．ただ，任意領域への移動を含む移動パターンの処理では，ワーストケースを示す cold 状態時に，最大 30 秒ほど処理にかかったケースもあった．この結果はキャッシュヒット率を向上させるチューニングによって，改善することがのぞまれる．また，今回の問合せは重いタスクである．応答が重要であれば，パターンの領域の範囲を限定した軽いタスクにすることも考えられる．

6.3 結合処理の違いについて

移動パターン問合せは，各状態のリレーションを生成し，それらを結合することで実現している．そのため，その結合処理方法や結合順番等が問合せ処理時間に大きく影響を与えると考えられる．そこで，結合処理方法による問合せ処理時間の違いを調べた．結合処理方法は，入れ子ループ結合，マージソート結合，ハッシュ結合の 3 種類がある．4 つの領域を連続して移動するパターンを 10 個用意し，それらの移動パターン問合せを行い，その処理時間を測定した．プランによる処理時間の違いを比較しやすいように，hot の状態で実行，かつ，位置情報が密に分布している領域を対象とした移動パターン問合せを実行した．移動パターンの領域の範囲内にあるレコード件数は約 85000 件である．

図 8 に入れ子結合時の問合せプラン，図 9 にソートマージ結合時の問合せプラン, 図 10 にハッシュ結合の問合せプランを示す．入れ子ループ結合の場合だけが最初のリレーションから順に結合していたが，その他の二つのプランは 1 つ目，2 つ目の状態をリレーション，3 つ目，4 つ目の状態のリレーションを結合し，その後それらの結果を結合していた．

それぞれの結合処理で行った問合せの最大処理時間，最小処理時間，平均処理時間を図 11 に示す．この結果からわかるように，場合によっては，入れ子ループ結合を利用した問合せは処理が最も速くなることもあるが，最も遅くなることもある．入れ子ループ結合は，リレーションが小さい場合に他の結合方法よりも処理時間が短くなる．入れ子ループ結合の処理が速かった要因としては，空間索引で位置情報を取得する段階で移動パターンの候補が削減でき，その後結合するリレーションが小さくなったために，入れ子ループ結合の処理が速くなったと考えられる．ソートマージ結合は，入れ子ループ結合のワース

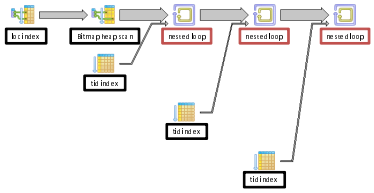

図 8　入れ子ループ結合を使用した場合

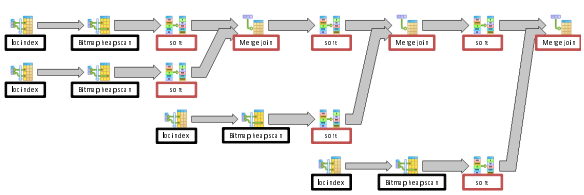

図 9　ソートマージ結合時を使用した場合

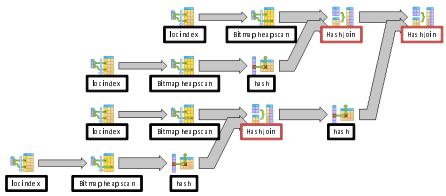

図 10　ハッシュ結合時を使用した場合

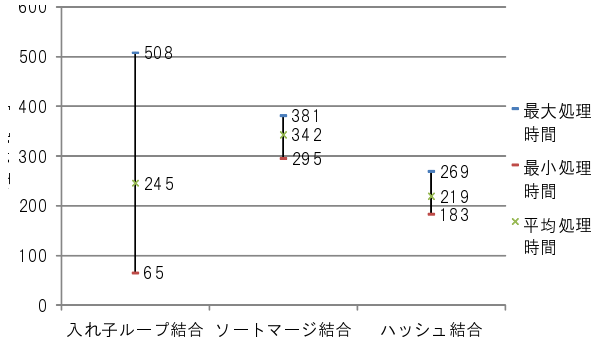

図 11　結合処理の違いによる処理時間の変化

トケースよりも良い結果となるが，最小処理時間，平均処理時間共に最低であった．ハッシュ結合を利用した問合せは処理が最も安定していた．

6.4 考　　察

今回の実験により，処理時間は問合せした移動パターンと問合せプランに大きく依存していることがわかった．効率的な問合せプランが作成されるかは，RDBMS 内のオプティマイザの性能次第であり，RDBMS のチューニング等で問合せプランを変更することは難しい．しかしながら，実行する SQL によって効率的な問合せプランを作成するように仕向けることはある程度可能である．そこで，フレームワークが生成する SQL を改善する方法が 2 つ考えられる．1 つ目は，意図的に問合せプランを変更する記述を SQL に追加することである．Oracle などではヒントを SQL に記述し，オプティマイザに指令を出すことができるため，問合せプランを変更することができる．ただ，今回使用した PostgreSQL ではそのような記述がないために使用できない．もう一方の方法は，特定の一つのリレーション，または，複数のリレーションに対して冗長な条件を追加することである．その具体例として，冗長な状態遷移の条件を追加することである．位置情報のテーブルには，タプル ID，次の時刻の位置を保持したタプル ID を示した next と前の時刻の位置を保持したタプル ID を示した prev があるが，今回提案したアルゴリズムでは，状態遷移の条件として，next と次のタプルのタプル ID が一致するという条件だけを利用した．そこで，次のタプル ID が持つ prev と現在のタプルのタプル ID が一致するという冗長な条件を追加する．この条件を追加することで，オプティマイザがより柔軟に問合せプランを作成でき，その結果問合せ処理が短縮される可能性がある．

7. ま と め

本研究では，センサから得られた位置情報を RDBMS に蓄積し，その履歴情報を用いて移動パターン問合せを行うための問合せ手法を提案した．移動パターン問合せを実現するためのフレームワークを検討し，そのフレームワーク上で必要となる SQL 生成アルゴリズムについて述べた．生成アルゴリズムは，移動パターン問合せを表す決定性有限オートマトンから基本的な移動パターン問合せを実現する SQL へ変換する．また，実データを用いて，本手法によって生成された SQL を RDBMS 上で実行し，本手法の有効性を確認した．今後，SQL 生成アルゴリズムの改善，問合せの拡張を行っていきたい．

謝　　辞

本研究の研究経費は，内閣府最先端研究開発プロジェクト (FIRST) による．

文　　献

[1] A. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. White. Cayuga: A general purpose event monitoring system. In *Proc. CIDR*, 2007.

[2] Y. Diao, N. Immerman, and D. Gyllstrom. SASE+: An agile language for kleene closure over event streams, 2007.

[3] W. Fan, X. J. Yu, H. Lu, J. Lu, and R. Rastogi. Query translation from XPath to SQL in the presence of recursive DTDs. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pp. 337–348, 2005.

[4] J. Letchner, C. Re, M. Balazinska, and M. Philipose. Access methods for markovian streams. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, pp. 246–257, 2009.

[5] Y. Mei and S. Madden. ZStream: a cost-based query processor for adaptively detecting composite events. In *Proceedings of the 35th SIGMOD International Conference on Management of Data*, pp. 193–206, 2009.

[6] M. Stonebraker and U. Cetintemel. One size fits all.

[7] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pp. 407–418, 2006.

[8] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma. Understanding mobility based on GPS data. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, UbiComp '08, pp. 312–321, 2008.

[9] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pp. 791–800, 2009.

[10] 川島, 寺島, 北川. ストリーム処理エンジンにおける効率的な来歴管理. 日本データベース学会論文誌, 8(1):101–106, 2009 年 6 月.

[11] 樫山, 花井, 田中, 今木, 西澤. ストリームデータ処理におけるデータベースアーカイブ処理高速化の提案と評価. 電子情報通信学会技術研究報告 (データ工学), No. 131 in DE2007-77, pp. 333–338, 2007 年 7 月.

[12] 山田, 渡辺, 北川, 天笠. ストリーム管理システムにおける永続化要求の妥当性評価. 電子情報通信学会技術研究報告 (データ工学), 106(149):209–214, 2006 年 07 月.