1C4-OS-13a-6

データストリーム手法による行動軌跡パターン検出と時空間情報の可視化

Detection of Trajectory Patterns and Visualization of Spatio-Temporal Information Based on a Data Stream Approach

王 一驄[*1] Yicong Wang
関 和広[*1] Kazuhiro Seki
上原 邦昭[*1] Kuniaki Uehara

[*1] 神戸大学大学院システム情報学研究科計算科学専攻
Department of Computational Science, Graduate School of System Informatics, Kobe University

As the rapid increase of mobile GPS devices such as smartphones, it is meaningful to mine or analyze the massive trajectory data streams from those devices. Though there are many algorithms that can find patterns from massive static trajectory data or a dynamic data stream by a batch process, what we need now is a new algorithm that can deal with a massive data stream with limited resources. Our proposed algorithm is aimed at discovering the place at which people always stop when they moves or the place which is becoming crowded from massive trajectory data stream by online process.

1. 研究背景

1.1 人の流れプロジェクト

近年、防災や防犯、マーケティング、交通・都市計画などにおいて、時々刻々とダイナミックに変動する人々の動きを面的に把握する必要性が出てきている [PF-project 2011]。このため、東京大学空間情報科学研究センターでは「人の流れプロジェクト」をが行われている。このプロジェクトは、大量の人々の流れに関するデータの品質の確保と、その処理のための共通処理基盤やデータ処理技術のあり方について概観し、研究者や実務者などを対象とした時空間サービスの実現を目指したものである。本研究では、このプロジェクトから提供される、人の移動軌跡データを実験用データとして利用している。このデータは、2008 年 10 月 1 日一日の人の移動軌跡データであり、一日当たりのデータ量として 70GByte、内訳はユーザ ID、時刻、経緯度、性別、年齢、職業、交通手段などから構成されている。

1.2 既存アルゴリズム

近年、DenStream [Cao 2006] など DBscan に基づくアルゴリズムが登場している。DenStream は、検索半径を十分小さく設定して、移動軌跡データから人の密度が高い場所を見つけるアルゴリズムである。DenStream は、ストリームデータを取扱うことができるが、得られるパターンは、ある地点の密度が高いあるいは低いなど、ある時点での静的な情報に過ぎない。人が単に集まって動かない場合は、人々の動きに関する頻出パターンが発見できない。本研究で求めようとするパターンは、動的な情報、もしくは人の密度が増えている場所、人が集まってくる場所である。例えば、短い時間間隔で、ある場所に大量の人が流入すると、この場所で大きな交通問題が生じることになる。このような状況を発見することが本研究の目的である。

一方、人の移動軌跡から頻出パターンを見つけるために、T-pattern miner [T-pattern miner] と呼ばれるソフトウェアがある。T-pattern miner のアルゴリズムは文献 [Trajectory Pattern Mining 2007] に基づいている。しかしながら、T-pattern miner はバッチ処理を前提としており、ストリームデータには適用できない。本研究で用いる人の移動軌跡のデータは、一日間で 70GByte になるため、単純にバッチ処理をすると、少なくとも 70GByte のメモリーが必要になる。このため、メモリー消費を抑えてデータストリームを取扱うことできるオンライン型のパターン頻度計算アルゴリズムが必要となる。

1.3 頻出パターンの検出

文献 [Anand 2010] では、ストリームデータから頻出パターンを検出する手法が提案されている。この手法の目的は、ストリームデータに現れる、各エレメントの頻度を計算することである。このアルゴリズムは、新しいデータの重みを大きくすることに特徴がある。具体的には、まずストリームデータを各エレメントに関するいくつかのサブストリームに分ける。例えば、ストリームデータのコンテンツが”a”, ”b”, ”c”からなるとき、ストリームデータから”a”のみを表現したサブストリームを作ることを考える。具体的には、ストリームデータの i 番目コンテンツが”a”なら、”a”のサブストリームの i 番目を 1、そうでなければ 0 にする。したがって、以下の Stream は、

Stream: c b a b a ……
time ⟶

以下のサブストリームのように変換される。

$$\text{Substream}_{"a"} = 0\,0\,1\,0\,1\cdots\cdots$$
$$\text{Substream}_{"b"} = 0\,1\,0\,1\,0\cdots\cdots$$
$$\text{Substream}_{"c"} = 1\,0\,0\,0\,0\cdots\cdots$$

ここで各エレメントの頻度を Popularity とすると、

$$\text{Popularity}_{"a"} = 0\cdot c^4 + 0\cdot c^3 + 1\cdot c^2 + 0\cdot c^1 + 1\cdot c^0 + \cdots\cdots$$
$$\text{Popularity}_{"b"} = 0\cdot c^4 + 1\cdot c^3 + 0\cdot c^2 + 1\cdot c^1 + 0\cdot c^0 + \cdots\cdots$$
$$\text{Popularity}_{"c"} = 1\cdot c^4 + 0\cdot c^3 + 0\cdot c^2 + 0\cdot c^1 + 0\cdot c^0 + \cdots\cdots$$

となる。もしこのストリームの先頭から 5 個のコンテンツからなる列の各エレメントの Popularity を考えると、

$$\text{Popularity}_{"a"} = c^2 + 1$$
$$\text{Popularity}_{"b"} = c^3 + c$$
$$\text{Popularity}_{"c"} = c^4$$
$$\text{Popularity}_{"a"} > \text{Popularity}_{"b"} > \text{Popularity}_{"c"}$$

連絡先:王 一驄, 神戸大学大学院システム情報学研究科, 神戸市灘区六甲台町 1-1, wang@ai.cs.kobe-u.ac.jp

となる。これは、"c"の出現回数が 1 回であることから、"a"や"b"より少ないこと、"a"と"b"は両方とも 2 回出現しているが、"a"は最近出現したので popularity がより高いという事実に一致している。さらに Popularity の計算を数式で表すと

$$\mathrm{Popularity}_a = \sum_{i=0}^{t} a_i c^{t-i}$$

となる。ここで c は1に近い定数、例えば $1\text{-}10^{-7}$とする。

$$a_{t-i} = \begin{cases} 1 & \text{stream の先頭から i 番目のコンテンツが a の時} \\ 0 & \text{a ではない時} \end{cases}$$

この手法では、出現回数が多く、かつ最近出現したコンテンツが高い Popularity となる。実際にストリームを入力として計算すると、この式は極めて簡単な形で実現できる。すなわち、ある時点tのエレメント a の Popularity を $\mathrm{Popularity}_{a@t}$とすると

$$\mathrm{Popularity}_{a@t+1} = c \cdot \mathrm{Popularity}_{a@t} + \delta$$

となる。ここで、$\mathrm{Popularity}_{a@0} = 0$ である。また δ は以下のように定義される。

$$\delta = \begin{cases} 1 & \text{Stream から次のコンテンツが a の時} \\ 0 & \text{Stream から次のコンテンツが a ではない時} \end{cases}$$

以上の式をアルゴリズムに導入すると、過去のデータを保存せずに、Popularity が計算できるようになる。以下では、この式に基づいて本アルゴリズムを提案する。

2. 提案手法

大量の移動軌跡データを入力として、一つストリームから

$$\cdots\cdots(x_t, y_t), (x_{t+1}, y_{t+1}) \cdots\cdots$$

のようなデータが流れてくるものとする。これは人の動きをタイムスタンプごとに表したものである。まず、このデータから人の軌跡がどこで止まったか判断することを検討する。データは時間順に流れてくるので、

$$(x_{t-2}, y_{t-2}) \neq (x_{t-1}, y_{t-1}) = (x_t, y_t) \qquad ①$$

のとき、(x_t, y_t)で止まったと判断できる。プログラム実装する時には、一つのストリームを三つの変数で構成されるリングバッファで監視して、データストリームから新しい(x_t, y_t)が来れば、式①を用いて人が止まったかどうかを判断する。

次に、地図でデータが出現するエリアを、適当に m·n 個セルに分ける。本稿では、セルの辺長を 50mと設定している。ここで、セルを新しいデータストリームのコンテンツと考えると、新しいデータストリームのエレメントは全部で m·n 種類になる。1.3 節では 1 次元のストリームデータを考えたが、ここでは x, y からなる 2 次元に拡張し、各セルの Popularity を計算している。

プログラム実装では、まずリングバッファで(x_t, y_t)に相当するセルに移動軌跡が止まったことを検出し、1.3 節の手順で各セルの Popularity を計算する。具体的には、(x_t, y_t)に相当するセルの Popularity に定数 c をかけて 1 を足す。それと同時に、ほかのセルの Popularity にも c をかける。このようにすれば、人が立ち止まったセルの Popularity が、他のセルと比べて大きくなる。

例として、図 1 のように、ある移動軌跡が右上のセルに止まった場合、右上セルは $\mathrm{Popularity}_{UR} = \mathrm{Popularity}_{UR}*c+1$ となる。他のセルは $\mathrm{Popularity}_{else} = \mathrm{Popularity}_{else}*c$ となる。

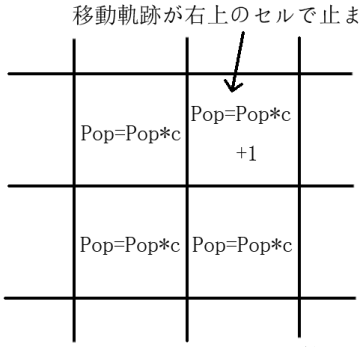

図 1 Popularity の計算

以上の動作を表1に擬似コードで示す。

表 1　Popularity を計算するプロトタイプ

```
stop-point = the cell which (x_t, y_t) fall at;
for obj in cells[m][n]{
   if(obj==stop-point) obj.popularity=
      cell.popularity*constant+1;
   else obj.popularity*=constant;
}
```

表1のアルゴリズムはプロトタイプだが、2つの欠点がある。まず第一に、一つのセルに移動軌跡が止まったと判断した後、すべてのセルの popularity をアップデートするために、計算時間は O(m*n)を要する。したがって、地図が大きい場合、計算時間を削減する必要がある。

第二に、Popularity は、出現回数と出現時刻の2つの要素に影響を受けるが、出現回数の影響の方が大きい。例えば、これまで A 駅の Popularity は何時間かずっと高かったが、最近の 10 分間で、A 駅から遠くない場所 B で事件が起こったとする。この場合、B に人が集まる可能性が高いが、本アルゴリズムでは、A 駅の Popularity は依然として B より遥かに高いことになる。したがって、動きの頻出パターンが発見されることはない。

以上の問題を解決するために、まず、地図をサブエリアに分割して、セルの Popularity をアップデートする計算時間を減少させる。さらに、過去データからの影響が大きすぎるという問題に対しては、最近のデータのみ計算する方法を考える。たとえば、過去 10 分間のデータを計算する場合、まず 10 分間のデータにタイムスタンプをつけて、Array List の後ろから保存する。Array List の先頭は、現在時刻の 10 分前に相当するため、先頭のデータに当たるセルの Popularity から c^{10} を引く。以上の操作により、先頭のデータを削除すれば、先頭データ(10 分間前のデータ)が各セルに及ぼす影響も除去できることになる。擬似コードを表2(ページ3)に示す。

この手法では、バッチ処理せずに、10 分間前のデータが各セルに及ぼす影響を除去できるので、表1のプロトタイプに比べて、計算量をほとんど増やさず、前 10 分間のデータを保存することができるという特徴がある。

最後に本アルゴリズムの計算時間とメモリコストについて分析する。データストリームの数を n、セルの数を m とする。まず、空間的に考えると、データストリームを監視するリングバッファにつ

いて、コンテンツ保存用変数 3 つとバッファヘッドを示す変数 1 つが必要になる。このためリングバッファのメモリコストはO(n)になる。

表２　最近データを考慮したプロトタイプ

```
initialize  Array List databuffer;
for (x_t, y_t) in data stream{
    top-point = the cell which (x_t, y_t) fall at;
    add (stop-point, timestamp) to databuffer;
    while(timestamp of the 1st element of databuffer == current
time - 10    minutes){
          (the cell which the 1st element of databuffer point
to).popularity -= c^10;
          remove first element of databuffer;
          }
    for obj in subarea{
       if   (obj==stop-point)   obj.popularity=
          obj.popularity*constant+1;
       else obj.popularity*=constant;
}}
```

時間的に考えると、一つコンテンツごとに式①で判断するので、一つのコンテンツに対する時間コストはO(1)になる。ここで、一つのコンテンツに対する時間コストを考える理由は、ストリームデータを取り扱うアルゴリズムは、通常のアルゴリズムと異なることによる。すなわち、ストリームデータを取り扱うアルゴリズムでは、次のコンテンツが来る前に、直前のコンテンツに関する計算を終わらせられることが重要である。したがって、ストリームデータを取り扱うアルゴリズムには、一つのコンテンツに対する時間コストが評価指標になる。

次に、セルの Popularity を保存する行列について空間的に考えると、m 個変数があるので、メモリコストはO(m)になる。時間的に考えると、サブエリアすべてのセルの popularity をアップデートするため、時間コストはO(サブエリアのセル数)になる。したがって、全体的に考えると、メモリコストはO(m+n)、一つのコンテンツに対する時間コストはO(サブエリアのセル数)である。

例えば、東京都全域、入力ストリームが 50 万の場合を考えると、東京都全域の面積は 2,188km2 なので、セルの辺長を 50m とすると、メモリコストは最低 2188*106/502*4Byte(Float)+5*105*4*28Byte(Float*2*3+Int)=56MByte となる。10 分間前のデータを保存する Array List を考えると、「人の流れプロジェクト」プロジェクトから提供される東京データは一日当たり 70GByte なので、10 分間当たりのデータ量は 0.486GByte となり、タイムスタンプをつけて全部メモリで保存しても負荷は高くない。

3. 実験結果

今回の実験では、東京大学空間情報科学研究センターの「人の流れプロジェクト」の人の移動軌跡データを使って、特に JR 東京駅を中心として半径 3km以内の一日分のデータを入力とした。サンプリングユーザ数は 76,685 人である。Google Maps に基づいて計算結果を可視的に出力したものを図 2 に示す。

図2は、昼 12 時頃の新橋駅付近の様子である。10 分間に新橋駅で降りた乗客は予想通り多かったので、深い黒セルになっている。同時に、みずほ銀行新橋中央支店の所には、新橋駅より薄いセルがある。これは、同じ時間帯で銀行に来客が多かったことを表している。人の密度でいえば、新橋駅が銀行よりもはるかに高いので、DBScan 系のアルゴリズムによってクラスタリングすれば、新橋駅ではもちろんクラスタが発見されるが、銀行ではおそらくクラスタは発見されないと予想される。

図 2　本アルゴリズムによる動きパターンの検出

実際に DenStream アルゴリズムを用いて、昼 12 時頃の新橋駅付近のデータを計算した結果を図3に示す。みずほ銀行新橋中央支店の所では、予想通り、クラスタは発見されていない。

図 3　DenStream よる動きパターンの検出

4. 今後について

本稿では、人の移動軌跡から人がよく止まる場所を見つけたが、今後は、見つかった場所に基づいて人の動きパターンを検出する研究を行う予定である。

参考文献

[Anand 2010] Anand Rajaraman and Jeffrey David Ullman: Mining of Massive Datasets, Cambridge University Press, 2010.

[PF-project 2011] http://pflow.csis.u-tokyo.ac.jp/index-j.html

[T-pattern miner] http://sourceforge.net/projects/t-patterns/

[Trajectory Pattern Mining 2007] F. Giannotti, M. Nanni, D. Pedreschi and F. Pinelli: Trajectory Pattern Mining, Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 332-339, 2007.

[Cao 2006] Feng Cao, Martin Ester, Weining Qian and Aoying Zhou: Density-Based Clustering over an Evolving Data Stream with Noise, Proc. of 2006 SIAM Conference on Data Mining, pp. 326-337, 2006.