Translational and Rotational Invariant Mining of Frequent Trajectories and Related Optimizations

Alexander Andreopoulos

Bill Andreopoulos

Aijun An

Xiaogang Wang

Technical Report CS-2006-02

January 2006

Department of Computer Science and Engineering

4700 Keele Street North York, Ontario M3J 1P3 Canada

Translational and Rotational Invariant Mining of Frequent Trajectories and Related Optimizations

Alexander Andreopoulos
York University
Centre for Vision Research
Department of Computer Science and Engineering
Toronto, Ontario, Canada
alekos@cs.yorku.ca

Bill Andreopoulos, Aijun An
York University
Department of Computer Science and Engineering
Toronto, Ontario, Canada
{billa, aan}@cs.yorku.ca

Xiaogang Wang
York University
Department of Mathematics and Statistics
Toronto, Ontario, Canada
stevenw@mathstat.yorku.ca

ABSTRACT
We present a framework for mining frequent trajectories from a database of trajectories and propose various novel optimization techniques for efficiently mining such trajectories. We begin by presenting a methodology for discovering frequent trajectories, frequent trajectories which are translated with respect to each other and frequent trajectories which are both translated and rotated with respect to each other. We perform experiments demonstrating the effectiveness and accuracy of our approach. We then proceed to demonstrate a multiresolution methodology based on the wavelet transformation for speeding up the discovery of frequent trajectories and present some related experiments. We conclude the paper by proposing a methodology for tackling the so called curse of dimensionality problem of higher dimensional trajectories, by presenting an algorithm which scales linearly instead of exponentially as we increase the dimension of the trajectories we are interested in mining. Such optimizations are a necessity for mining higher dimensional trajectories. We conclude by pointing out some issues where more work needs to be done and argue that they can provide interesting topics for future research.

1. INTRODUCTION
There exist many situations where we are confronted with trajectories describing the movement of various objects. We are often interested in mining the *frequent trajectories* that groups of such objects go through. Trajectory datasets arise in many real world situations, such as in mobility experiments, for the discovery of biological patterns and in surveillance [12, 11, 5]. In surveillance, for example, we might have a camera observing a large shopping mall over a period of months, extracting the trajectories that customers in the mall follow. We could use such trajectories to mine for frequent paths that people like to follow and potentially make various improvements in the layout of the shopping mall. This could be very difficult to do manually, due to the high number of trajectories we would have to mine. Another example of where we might be interested in finding frequent trajectories is for observing the surroundings of a high security area such as an embassy. We might be interested in finding whether there are repeated patterns over a long period of time indicating that someone is surveying the location. In sports situations, a computer vision system could extract from images of the field the trajectories that players follow over periods of time and we could extract from these trajectories the frequent trajectories that players follow, in order to better understand the opponent's strategy. In a situation where we have lots of trajectories, it could be very difficult if not impossible to distinguish such interesting trajectories from the deluge of irrelevant trajectories. The contributions of this paper are as follows:

- We present a framework for finding frequent trajectories whose sampling interval is small enough to allow extracting their first and second order derivatives.
- We propose a fuzzy framework for dealing with errors and uncertainties in the trajectories.
- We propose a robust framework for mining trajectories that are translated and/or rotated with respect to each other.
- We present a multiresolution based framework for speeding up frequent trajectory mining.

This paper is organized as follows. Section 2 presents some related work. Section 3 introduces our notation and the general framework we use for mining trajectories. Section 4 describes a method for mining translated and rotated trajectories. Section 5 offers an approach for optimizing the mining speed of frequent trajectories. Section 6 proposes some possible approaches for extending this method to higher dimensions. Section 7 concludes the paper.

2. RELATED WORK
Trajectory mining is a domain where various data mining techniques could be applied. Naturally, sequential pattern mining techniques [2, 17, 14] can provide useful insights. In sequential pattern mining we are typically given a database containing sequences of transactions and we are interested in extracting the *frequent sequences*, where a sequence is frequent if the number of times it occurs in the database satisfies a minimum support threshold. Popular methods for mining such data sets include the GSP algorithm [2] - which is an Apriori [1] based algorithm - and the PrefixSpan [15]

algorithm. GSP can suffer from a high number of generated candidates and multiple database scans. Pattern growth methods such as PrefixSpan are more recent approaches for dealing with sequential pattern mining problems. They avoid the candidate generation step, and focus the search on a restricted portion of the initial database making them more efficient than GSP [2, 15].

The problem that is most related to frequent trajectory mining is sometimes referred to as *frequent spatio-temporal sequential pattern mining* in the literature [12, 11, 5, 6]. The main difference between our work and previous work [12, 11, 19, 21, 20, 5, 6] is that our method assumes that we are dealing with *densely sampled* trajectories - trajectories whose sampling interval is small enough to allow us to extract from a trajectory its first and second derivative. Furthermore, this allows us to define a neighborhood relation between the cells making up our trajectories, allowing us to perform various optimizations. In general, previous work often assumes that we have sparsely sampled trajectories in our database.

The fact that the space of frequent trajectories forms a metric space where we can define a neighborhood relationship amongst the points through which a trajectory passes, can be exploited to provide various optimization techniques for efficient mining of trajectories. Similar work has been done in [3] for determining typical user navigation patterns of web users. However, the main difference of our work is what we have just indicated, namely, that in general they do not assume a neighborhood relationship amongst the websites visited, since from any website a user might directly jump to any other website. There has been a significant amount of research on defining similarity measures for detecting whether two trajectories are similar [18, 4]. However, this research has not focused on mining frequent trajectories.

3. APRIORI BASED MINING OF FREQUENT TRAJECTORIES

In this section we present an extension of the GSP algorithm [1] for mining frequent trajectories. We begin by introducing a so called *cell representation* of trajectories and then present an Apriori based algorithm for mining frequent trajectories. Then we introduce an improvement to the frequent trajectory mining algorithm that is more robust in mining trajectories that are similar but not identical.

3.1 Cell representation of trajectories

We define a trajectory c as a continuous function $c(s) = [x(s), y(s)]$ in the 2D case and as $c(s) = [x(s), y(s), z(s)]$ in the 3D case. Similar extensions follow for higher dimensional trajectories. The function $c(s)$ is an arc-length parameterization of a curve/trajectory. In other words, the parameter s denotes the length along the trajectory and $c(s)$ denotes the position of the trajectory after traversing distance s. In other words our trajectories do not depend on time, or the speed with which the object/person traverses the trajectory. We assume independence from time and speed for mining the frequent trajectories and subtrajectories.

We define a trajectory c as frequent, if the number of the trajectories $\{c_1, c_2, ..., c_o\}$ in a database that pass through the path described by c satisfies a minimum support count threshold (*minsup*). This definition requires only that there exist *minsup* subtrajectories of all trajectories in $\{c_1, c_2, ..., c_o\}$ that are identical to c; but it does *not* require that $c = c_i$ for a sufficient number of c_i's. More formally, we say that trajectory c over interval $[0, \tau]$ is frequent with respect

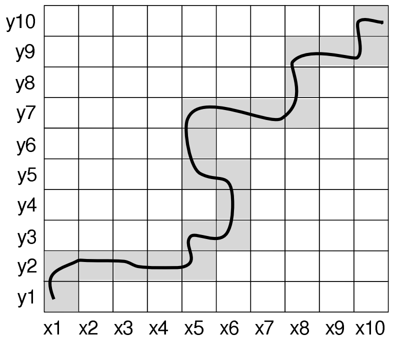

Figure 1: A cell sequence representation of a dense trajectory. The cell sequence representation of the dense trajectory consists of the gray cells (in order) that are intersected by the dense trajectory.

to a dataset of trajectories if there exist a *minsup* number of compact intervals $[\alpha_1, \alpha_1 + \tau], \cdots, [\alpha_{minsup}, \alpha_{minsup} + \tau]$ such that for all $i \in \{1, \cdots, minsup\}$ and for all $0 \leq s \leq \tau$ we have $c(s) = c_{\pi(i)}(\alpha_i + s)$ (where π is a permutation function of $\{1, \cdots, o\}$).

The frequent trajectory mining problem can be formulated as a sequential pattern mining problem in the following way. Here we only discuss the 2D case as the 3D case is similar. Assume that we are observing a square $region$ of size $N \times N$ over which all the trajectories occur. Although this region can be of arbitrary size $N \times M$, for notational convenience we assume it is an $N \times N$ square. By splitting the region into a $grid$ of square $cells$, as shown in Figure 1, we denote by (x_i, y_j) the cell located at the i^{th} column and j^{th} row. A potential way of discretizing a region into a grid is by uniformly sampling along the two dimensions. In this paper we create the grid by uniform sampling, even though square cells are not necessary for our approach to work. Then we define:

(i) A trajectory $c(s)$ is referred to as a *dense trajectory* if it is represented by a densely sampled set of points. The sampling interval depends on the problem at hand and should be small enough to obtain accurate first and second derivatives.

(ii) A dense trajectory's *cell sequence* refers to the sequence of cells $((x_{\pi_x(1)}, y_{\pi_y(1)}), \cdots, (x_{\pi_x(n)}, y_{\pi_y(n)}))$ intersected by the dense trajectory (where π_x is a permutation function). The following conditions must hold: $a.$ $\pi_x(i) \neq \pi_x(i+1)$ or $\pi_y(i) \neq \pi_y(i+1)$, and $b.$ $|\pi_x(i) - \pi_x(i+1)| \leq 1$ and $|\pi_y(i) - \pi_y(i+1)| \leq 1$. Thus, we encode the order in which the dense trajectory intersects the cells. As we discuss below, in some situations it is preferable to also associate with each cell (x_i, y_i) from the sequence the arclength/distance over which the trajectory falls in this cell.

(iii) The number of cells in a trajectory's cell sequence is its *length.* For example, the cell sequence
$((x_4, y_3), (x_3, y_2), (x_3, y_3), (x_3, y_4), (x_2, y_5))$ has length 5.

(iv) A *continuous subsequence* ω of a trajectory c's cell sequence $((x_{\pi_x(1)}, y_{\pi_y(1)}), \cdots, (x_{\pi_x(n)}, y_{\pi_y(n)}))$ must satisfy $\omega = ((x_{\pi_x(i)}, y_{\pi_y(i)}), (x_{\pi_x(i+1)}, y_{\pi_y(i+1)}), \cdots, (x_{\pi_x(j)}, y_{\pi_y(j)}))$ where $1 \leq i \leq j \leq n.$

In practice, for various reasons, a trajectory $c(s)$ might be represented by a small number of sample points. We can interpolate those points and subsample the interpolated trajectory, to obtain the dense representation of those trajectories.

Another possible way of representing a trajectory is by uniformly sampling the trajectory along its arclength and using the sequence of cells in which sampled points fall. If the sampling interval is sufficiently small this still enforces the constraint that for every cell there is a finite neighborhood of cells that the next point could fall in. However, this representation does not guarantee that the neighborhood relationship holds when mining translationally and/or rotationally invariant trajectories, so we decided to use the method described in the previous paragraphs for representing trajectories.

3.2 Standard Apriori based Mining

Using the cell representation method to represent trajectories, the problem of mining frequent trajectories is defined as finding all the contiguous subsequences of the cell sequences in a database that satisfy a support threshold. We first point out that frequent trajectories satisfy the Apriori property. Namely, any continuous subsequence of a frequent trajectory's cell sequence is frequent. We exploit this property to implement efficient algorithms for mining frequent cell sequences.

If (x_i, y_j) is our current cell position, the next allowable cell position (x_k, y_l) *must* be one of its 8 neighboring cells, such that $|i - k| \leq 1$ and $|j - l| \leq 1$. We use this constraint to modify the GSP algorithm to generate a much lower number of candidates than using the GSP algorithm if it does not include this constraint during candidate generation.

```
Inputs: $minsup$: Minimum support count.
        $D$: Data set of cell sequences of trajectories.

Output: All the frequent cell sequences.

(1) for each cell sequence $t \in D$
(2)     for each cell $g \in t$
(3)         $g.count++$.

(4) $L_1 = \{cell\ g | g.count \geq minsup\}$.
        //$L_1$ is the frequent length-1 cell sequences
        //(consisting of a single cell)

(5) for (k=2;$L_{k-1} \neq \emptyset$;k++) {

(6)     $C_k$ =trajectory($L_{k-1}$). //candidate
                                      //length-$k$ cell sequences

(7)     for each cell sequence representation $t \in D$ {

(8)         $C_t$= the set of contiguous subsequences of $t$
                   that are contained in $C_k$

(9)         for each candidate cell sequence $c \in C_t$
(10)            c.count++. //increment the support count
                           //of this candidate cell sequence

(11)    }

(12)    $L_k = \{c \in C_k | c.count \geq minsup\}$.

(13) }

(14) return $L = \cup_k L_k$.
```

Figure 2: Apriori based mining of frequent trajectories.

Figure 2 shows the pseudocode for the Apriori based mining of frequent trajectories. L_k is the set of frequent length-k cell sequences

```
Input: $L_k$: The length-$k$ frequent cell sequences.

Output: $C_{k+1}$: The candidate length-$(k+1)$ cell sequences.

(1) if ($k = 1$) {

(2)     for all pairs of single cells $(a_1) \in L_1$,$(b_1) \in L_1$
            such that $(a_1) \neq (b_1)$

(3)         if $(a_1)$ and $(b_1)$ are neighbors, then

(4)             $c_2 = (a_1, b_1)$. //join $a_1$ and $b_1$
(5)             $C_2 = C_2 \cup \{c_2\}$.

(6) } else {

(7)     for all pairs of length-$k$ cell sequences
            $(a_1, \cdots, a_k) \in L_k$, $(b_1, \cdots, b_k) \in L_k$
            such that $(a_1, \cdots, a_k) \neq (b_1, \cdots, b_k)$

(8)         if $(a_2, \cdots, a_k) = (b_1, \cdots, b_{k-1})$, then

(9)             $c_{k+1} = (a_1, \cdots, a_k, b_k)$.
(10)            $C_{k+1} = C_{k+1} \cup \{c_{k+1}\}$.

(11) }

(12) return $C_{k+1}$.
```

Figure 3: The `trajectory()` function.

found in the grid. C_k is the set of candidate length-k cell sequences. The algorithm first scans the database to find L_1, i.e., the length-1 frequent cell sequences, and then generates C_2, the length-2 candidate cell sequences using the `trajectory()` function. It then computes the support count of each candidate in C_2 by scanning the database, and keeps the frequent candidates as the length-2 frequent cell sequences. It then generates the next level (i.e. length-3) candidate using the `trajectory()` function. The process goes on until a frequent set or a candidate set is empty. The main difference between this algorithm and GSP lies in the `trajectory()` function for generating candidates of length k from frequent cell sequences of length $k - 1$. Figure 3 describes this function. Figure 4 shows step by step how this algorithm initially works to find frequent length-2 cell sequences from length-1 cell sequences (i.e. single cells). When finding the candidate length-2 cell sequences, it suffices to only join two length-1 cell sequences (i.e. single cells) if the cells are neighboring/adjacent to each other, resulting in a much smaller number of candidates than if we had used GSP to accomplish this without using this neighborhood constraint. For $k \geq 2$, when joining length-k cell sequences to find length-$(k+1)$ candidate cell sequences it suffices to only join cell sequences a with b if the last $k - 1$ cells of a and first $k - 1$ cells of b are identical. Figure 5 shows an example of this. We notice that by joining two continuous paths, the resulting path is also continuous. Also, notice that there is no pruning step in the candidate generation process of our algorithm. This is because pruning may cause the loss of good candidates since we are mining for frequent *contiguous* subsequences. This is another major difference between the stardard GSP algorithm and our algorithm.

Assume there are r cells into which our $N \times N$ region has been split and there are b neighboring cells for each non-boundary cell. In our

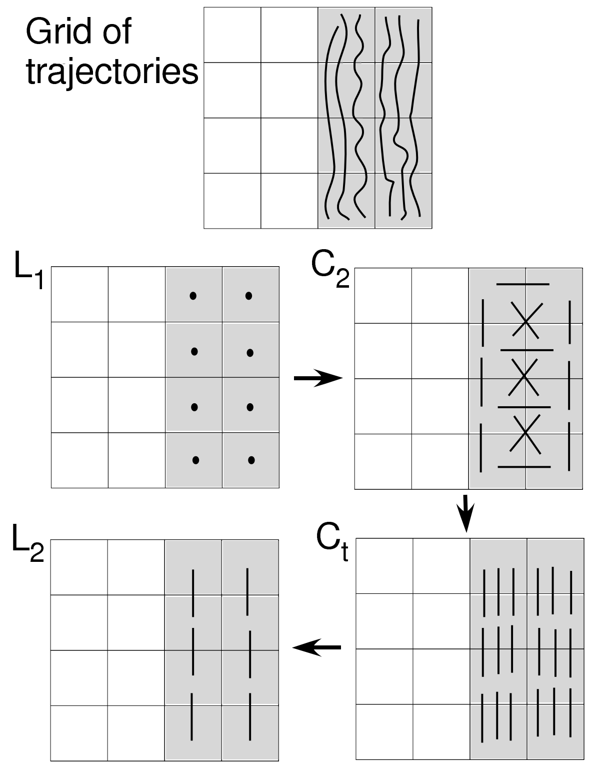

Figure 4: Assuming $minsup = 3$, the *grid* contains 6 frequent trajectories and 2 different cell sequence representations (these are the 2 gray columns). A cell sequence is represented as a line intersecting one or more cells. L_1 : The frequent length-1 cell sequences are the single cells that have at least $minsup$ trajectories intersecting them. C_2 : The candidate length-2 cell sequences include any set of 2 neighboring frequent single cells. C_t : Overlaps between C_2 and cell sequence representations in the initial grid. L_2: The frequent length-2 cell sequences include any cell sequence in C_2 that occurs at least $minsup$ times in the initial grid.

case $b = 8$, since each cell is surrounded by at most 8 other cells. Then, the upper bound on the number of length-$(k + 1)$ candidate cell sequences generated is $|L_k| \times b$. This is lower than the upper bound of $|L_k| \times r$ that GSP might generate if we were dealing with a sequential pattern mining problem where we could not apply this neighborhood constraint, since typically $b << r$. Of course, other algorithms for sequential pattern mining, such as PrefixSpan [15], could potentially be modified to solve this problem. An interesting topic for future research is to investigate their performance in this domain.

3.3 Robust Representation of Trajectories: Fuzzy Apriori

A problem with the previously discussed cell representation is that it is not too robust when trajectories pass close to a border of the grid, but on opposite sides of the border. This problem has been recognized in previous work on spatiotemporal pattern mining and various solutions have been proposed [11]. Consider for example two linear 2D trajectories. Assume for illustration purposes that they are given by the equations $y = 0.01$ and $y = -0.01$. If $y = 0$ is a border between cells (assuming we have split the region uniformly into square cells) then $y = 0.01$ and $y = -0.01$ will

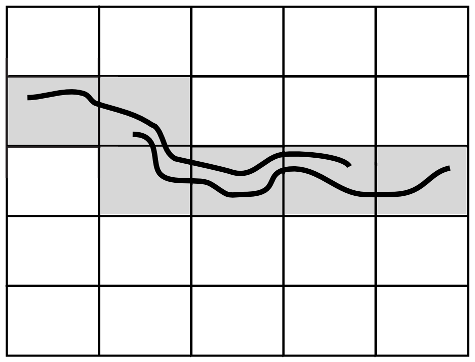

Figure 5: An example of joining two length-5 cell sequences to find a length-6 candidate cell sequence.

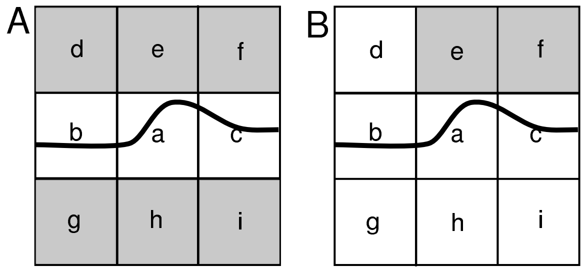

Figure 6: (A) Potential neighbors of cell a in its *fuzzy* cell representation are indicated with gray. Notice that cells b and c are excluded because they are entrance and exit points of the trajectory. (B) The *fuzzy* cell representation of cell a consists of cells a, e and f because the trajectory passes sufficiently close to those cells.

have completely different cell representations, even though they are extremely close to each other. We need to make our algorithm more robust in such situations.

To accomplish this, we define a *fuzzy cell representation* in the following way. Considering the 2D case shown in Figure 6A, assume a trajectory enters cell a from cell b and exits cell a to cell c. If we remove cells b and c, there remain 6 neighbors of cell a, indicated with gray in Figure 6A. If the trajectory passes sufficiently close to a subset S of these 6 neighboring cells, then we define the *fuzzy cell* through which the trajectory passes by $\{a\} \cup S$. Figure 6B shows the fuzzy cell as consisting of cells a, e and f. A cell x of a candidate trajectory matches this *fuzzy* cell if x is equal to any of the items/cells in the fuzzy cell.

When we form L_1 (the set of frequent single cells) we do not use the fuzzy cell representation. If we used fuzzy cells in L_1 the algorithm would be intractable. Let us consider 3 identical trajectories' cell sequence representations of length n and assume each fuzzy cell has size 4. If our minimum support count is less than or equal to 3, then the number of frequent cell sequences of length n would be at most 4^n. Obviously the algorithm would be intractable for large data sets. We tested this and indeed the algorithm is extremely slow if we do not use the above mentioned simplification.

When calculating the support count of a candidate cell sequence, it is infeasible to check if it agrees with each cell sequence implied by a fuzzy cell sequence. Instead, the support counting for a candidate cell sequence should be done by checking if each of its cells belongs to the corresponding fuzzy cell. If each cell of the candidate belongs to its corresponding fuzzy cell, then the candidate's support count is incremented. This takes linear time in the number of cells in the candidate cell sequence.

A potential future improvement could be to associate a number in the range 0 to 1 with each item/cell in a fuzzy cell. This number would indicate the importance of that particular item/cell in the fuzzy cell. When counting the support of a candidate cell sequence, we could use all the fuzzy values to better adjust the support count.

3.4 Experiments

We used MATLAB 7.01 running on an Intel Xeon 3Ghz with 3GB RAM to run our experiments. We chose to use MATLAB due to its suitability for rapid prototyping. The drawback that we encountered was that since MATLAB is an interpreted programming language it can be quite slow compared to other languages like C. To compensate for this drawback we adjusted the size of our testing set accordingly, so that our MATLAB programs could execute the code within a reasonable amount of time. We leave it for future research to implement these algorithms in a language such as C and use larger test sets.

If a dense trajectory passed within a Euclidean distance of 0.15 from one of the 6 neighboring cells we discussed above, we added the corresponding neighboring cell to the fuzzy representation of the cell (each cell had a dimension of 1×1 in our experiments). We used artificially created datasets as our trajectories. They were created by sampling the sinusoidal functions (sin, cos) and by placing various rotated lines in our trajectory space. The minimum support count we used was 2.

We first compared the standard Apriori mining method of section 3.2 by its own on a test set with 70 trajectories to verify the correctness of our algorithm. The dataset consisted of 10 identical sinusoidal trajectories, 10 randomly translated versions of these trajectories and 50 trajectories rotated by varying angles. Figures $10A$ and $10B$ show examples of translated and rotated trajectories, respectively. We discretized our trajectory space into 32×32 cells. The algorithm correctly mined all the frequent trajectories. It took about 4 minutes to mine these trajectories.

Then we created another artificial dataset in order to compare the two methods given in sections 3.2 and 3.3 and find which method is more robust when the trajectories are slightly different from one another. In this case, our test set contained three identical sinusoidal functions which had been randomly translated with respect to each other by a random distance between 0 and 1. The sinusoidals consisted of at most 67 cells/fuzzy cells. We discretized our trajectory space into 32×32 cells. The fuzzy Apriori algorithm managed to locate a frequent trajectory of length 67; however, the longest frequent trajectory that standard Apriori managed to find had length 31, worse than Fuzzy Apriori (Figure 7). In the presence of noise, standard Apriori tends to *fragment* the frequent trajectories, i.e., break them up into different smaller trajectories. This demonstrates the superior capabilities of Fuzzy Apriori for mining noisy trajectories. Fuzzy Apriori is proven to be slower than standard Apriori, but very robust and is able to handle trajectories for which standard Apriori is not satisfactory. The runtime of the algorithms is 8 seconds for standard Apriori and 9 minutes for Fuzzy Apriori for the data set. Fuzzy Apriori is slower mainly due to the greater size of its candidate data sets (Figure 8).

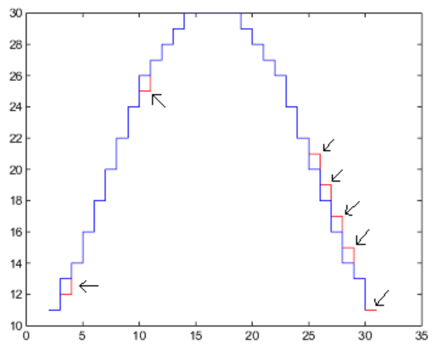

Figure 7: An original trajectory from our database (with color blue) and one of the two longest mined trajectories mined from the database using Fuzzy Apriori (with color red) superimposed on each other. Notice that the longest mined trajectory using Fuzzy Apriori is extremely similar to all the trajectories from the data set. Moreover, it does not have the problem of breaking up into different smaller trajectories (fragmentation) like it does when using standard Apriori. The arrows indicate the positions where the two trajectories do not overlap.

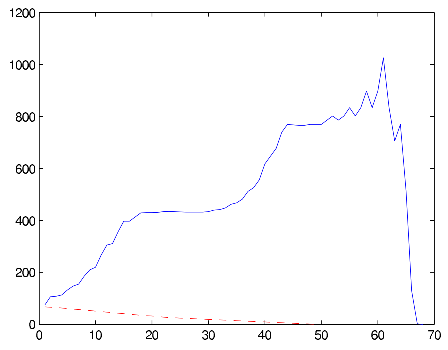

Figure 8: The y axis indicates the number of length-k candidates for the corresponding k-value on the x-axis. The red dashed line corresponds to standard Apriori and the solid blue line corresponds to Fuzzy Apriori.

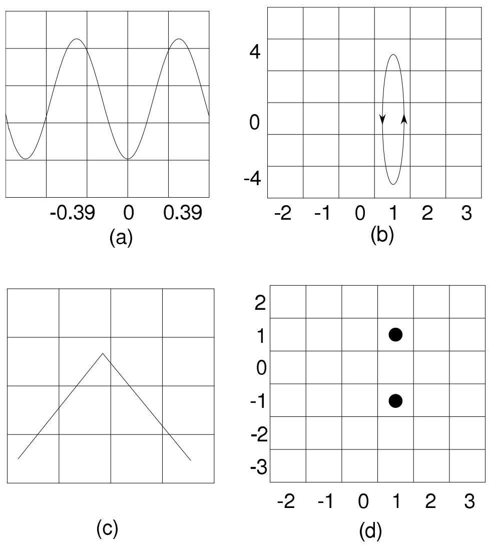

Figure 9: (a) **A trajectory given by function** $c(s) = (s, sin(4s))$. (b) **The derivative space** $c'(s) = (1, 4cos(4s))$. **We mine for such patterns to find translated patterns.** (c) **A trajectory that is not differentiable everywhere.** (d) **The derivative space of this function consists of two isolated and non-neighboring cells. We either have to smooth the function in** (c) **to have a continuous derivative space, or we are forced to use standard Apriori to do the mining, which is much slower.**

We believe that there is a wealth of research waiting to be done for creating more efficient methods for mining fuzzy trajectories, using a smaller number of candidate trajectories. Algorithms similar to PrefixSpan [15] could potentially lead to significant improvements in the mining speed. Furthermore, by adjusting the threshold by which we create the fuzzy trajectories and by adjusting the maximum size a fuzzy cell can have, we believe we could get significant improvements in mining speed.

4. TRANSLATIONAL AND ROTATIONAL INVARIANT MINING

In this section we present two trajectory mining techniques. The first is a method for mining frequent trajectories that are translated with respect to each other. The second method is for mining frequent trajectories that are both translated and rotated with respect to each other. Figures $10A$ and $10B$ show examples of translated and rotated trajectories, respectively. Such algorithms are useful in situations where we are interested in detecting more complex motion patterns. For example in surveillance situations, the camera which extracts the motion patterns might be rotated and translated by an unknown amount over the course of acquiring the motion patterns. In such situations the best we can hope to accomplish, in terms of frequent trajectory mining, is to make the frequent trajectory extraction invariant to the unknown amount by which the camera and subsequently the trajectories were translated and rotated.

4.1 The Derivative and Curvature

Assume $c_1(t) = [x(t), y(t)]$ and $c_2(t) = [x(t)+5, y(t)+3]$. In other words c_2 is a translated version of path c_1. If we take the derivatives $c_1'(t)$, $c_2'(t)$ of these two paths then we notice that $c_1'(t) = c_2'(t)$ for all values of t. We use this fact to mine for frequent trajectories that are translated with respect to each other.

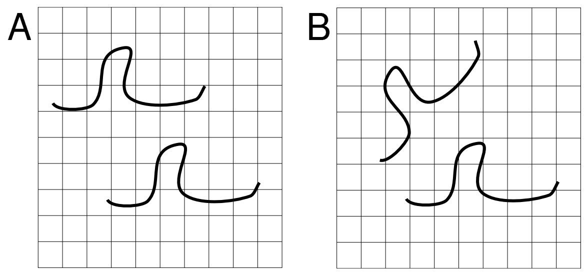

Figure 10: (A) **Trajectories that are translated with respect to each other.** (B) **Trajectories that are both translated and rotated with respect to each other.**

An issue to keep in mind is that derivatives tend to magnify noise. In other words, two trajectories that are slightly different due to noise would have an even more different derivative. We discuss methods for dealing with this later. We mine for frequent translated trajectories in the following way. For every dense trajectory $c_i(t) = [x_i(t), y_i(t)]$ in our database of trajectories we use finite differences to find its derivative $c_i'(t) = [x_i'(t), y_i'(t)]$. Then we use equiwidth binning [7] to discretize the space of derivatives for the x-coordinates and y-coordinates into a number of bins (cells). Then we represent every $c_i'(t)$ by a sequence of tuples (dx_i, dy_i), each of which denotes the current *derivative cell* in which the trajectory is located. A new tuple (dx_i', dy_i') is added to the sequence of tuples whenever the trajectory's derivative changes significantly enough to be part of a new derivative cell (Figure 9). For example, a linear trajectory is encoded by a sequence of length 1 (a single derivative cell) since its slope is constant. With each such tuple we could also associate a number denoting the arclength/distance over which the cell occurs. We use this measure, as described below, to detect translated trajectories. We can apply on this new trajectory representation the mining algorithms described in section 3 to find frequent trajectories that are translated with respect to each other.

Note than in our test cases we make the assumption that we are dealing with differentiable functions that do not change 'abruptly'. Figures 9c and 9d show problems that might arise otherwise. If we wish to mine such trajectories, we could either apply some sort of smoothing such as the wavelet transformation described in the next section to make the function better behaved, or we could apply some sort of standard Apriori/GSP/PrefixSpan mining which does not make the neighborhood assumption for adjacent cells in our cell sequence representation. This would likely be detrimental to our trajectory mining speed. We do not deal with this issue in this paper and leave it as a topic for future research.

The *curvature* of an arclength parameterized path $(f(s), g(s))$ at position s is given by the derivative with respect to s of the angle θ the path makes with the x axis.

$$\kappa = \frac{d\theta}{ds} \qquad (1)$$

Intuitively the curvature gives us a measure of the rate with which a curve is changing direction. It can be shown that an equivalent way of expressing the curvature is by the following determinant:

$$\kappa = \begin{vmatrix} f' & g' \\ f'' & g'' \end{vmatrix} = f'g'' - f''g' \qquad (2)$$

It is straightforward to show that any rotation and translation of $(f(s), g(s))$ results in the same curvature measure. In other words *curvature for 2D trajectories is rotation and translation invariant*. We can, therefore, use this measure to detect rotationally and translationally invariant patterns in a similar way as we did with translated trajectories.

To encode each trajectory $c_i(t)$ we follow the same procedure that we followed for the translationally invariant mining. We can use equiwidth or equidepth binning [7] to discretize the space of curvature measures and encode each trajectory using a sequence of *curvature cells*. We could also associate a number with each curvature cell, denoting the arclength/distance over which the trajectory belongs in this cell before it changes significantly to warrant using another cell in the sequence to encode it.

4.2 Experiments

First we mined the translated trajectories. We used the dense trajectories that were also used in section 3. However, in this case we *randomly translated* all the sinusoidal trajectories by much greater distances than we did for some of the trajectories in section 3. This removes the possibility that two trajectories match because they are physically near each other. Then, we found the derivatives of the dense trajectories and discretized the trajectories into derivative cell representations. The derivatives were split into a 32×32 grid. Then, we applied our mining method to mine these derivative cell representations. It took about 7 seconds to mine the dataset and the method was proven to be succesful in finding the translated datasets.

Then, we mined the translated and rotated trajectories. This time we also *randomly rotated* the sinusoidal patterns. We applied the previously described curvature detection method and then used equidepth binning to split the space of curvature values from our dataset into bins (cells). Each dense trajectory was then split into a curvature cell representation and we again applied our mining method to mine for the frequent trajectories. The algorithm ran on 12 seconds and was able to find all the rotated trajectories, demonstrating the validity of the algorithm.

We wish to point out a potential problem when we are dealing with real world applications which, however, has not been a problem in our dataset. The problem is that in the above cell sequence representation of derivatives and curvatures we have not associated the distance over which each dense trajectory belongs in a particular cell. It is possible for example to have two trajectories whose cell derivative representation consists of the same two cells. If the distance over which each dense trajectory belongs in each cell is very different, the two trajectories might be very different and should not lead to a match. The solution to this problem is the one indicated at the beginning of this section, namely to associate with each cell the distance over which the dense trajectory belongs to the cell. There are two simple ways we could use these distances to solve this problem. In one solution the cell distances could become an extra dimension in the trajectory. So for example in a 2D trajectory, we would add a third dimension containing the distance over which each derivative/curvature cell has the value it has and then perform mining of 3D trajectories. A simpler and probably faster approach would be for each frequent mined derivative/curvature cell sequence, to investigate the trajectories supporting the sequence and split the supporting trajectories into groups based on the cell distances of each trajectory - using for example some sort of a clustering technique.

Figure 11: Multiresolution decomposition of an image, for 3 different resolutions.

5. WAVELET BASED OPTIMIZATION OF MINING SPEED

So called multiscale and multiresolution techniques are well known in the signal processing community and are of great use for solving difficult problems such as image denoising and image compression[10, 8]. More recently, the applicability of such methods has been demonstrated for various data mining problems. For example WaveCluster is a multiresolution clustering algorithm that uses the Wavelet transform to transform the original data and find dense regions in the transformed space [16]. The Wavelet transform is useful because it has the ability to supress weaker information, effectively being a good method for noise removal. Since we are dealing with less information, we can speed up the mining process.

In our previous experiments we demonstrated the accuracy of the proposed mining methods. However, the methods may generate a high number of candidates. In this section we investigate the use of multiresolution techniques for speeding up our methods. We point out that for various types of datasets it can lead to significant optimizations in discovering frequent trajectories.

5.1 The Basics of Wavelets and Multiresolution Analysis

Wavelets $\psi_{a,b}$, $(a, b \in \mathbf{Z})$ are $\mathbf{L^2}(\Re)$ functions (functions such that $(\int_{-\infty}^{\infty} |\psi_{a,b}(t)|^2 dt)^{1/2} < +\infty$) that are generated by dilations and translations of a so called mother wavelet ψ [9]. One can construct wavelets ψ such that the dilated and translated family

$$\{\psi_{a,b} = \frac{1}{\sqrt{2^a}} \psi(\frac{t - 2^a b}{2^a})\} \qquad (3)$$

is an orthonormal basis of $\mathbf{L^2}(\Re)$. We can use such orthogonal wavelet bases to compute the approximation of a signal $f \in \mathbf{L^2}(\Re)$ at various resolutions, by the orthogonal projection of f on different subspaces of $\mathbf{L^2}(\Re)$. The details of how such *multiresolution* approximations of functions are built is beyond the scope of this paper and the reader is referred to [9] for more details. The point is that it is possible to use this orthogonal wavelet basis to construct spaces $\mathbf{V}$ and $\mathbf{W}$ such that $\mathbf{V} \oplus \mathbf{W} = \mathbf{L^2}(\Re)$, where $\mathbf{V}$ is a space corresponding to the low frequency content of $\mathbf{L^2}(\Re)$ and $\mathbf{W}$ corresponds to the high frequency content of $\mathbf{L^2}(\Re)$. The projection of a signal on these two spaces allows us to decompose a signal into its low and high frequency content, thus, effectively splitting the frequency axis of the signal into high and lowpass components. Repeated applications of this process allow us to obtain a wavelet

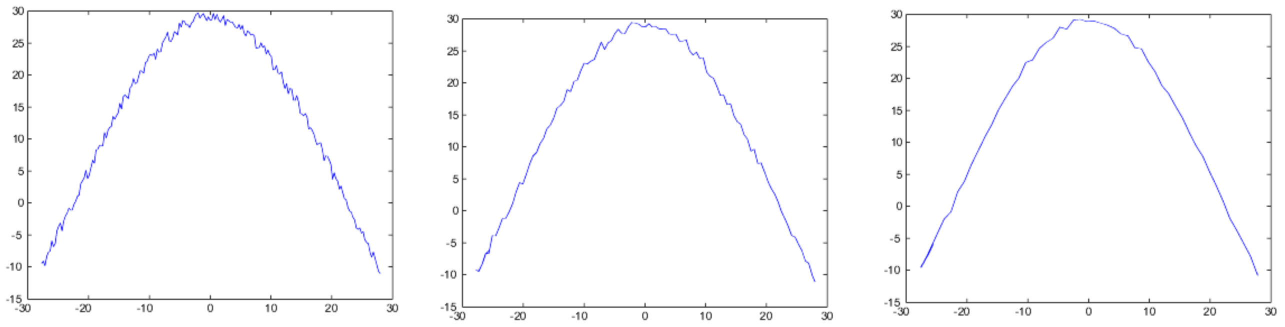

Figure 12: Multiresolution representation of a trajectory. From left to right, three progressively denoised trajectories using the wavelet transform, progressively using a smaller number of samples for each image (the plotting program we used automatically joins adjacent points with a straight line).

decomposition of the signal f. Wavelets have been applied with success in problems such as multiresolution image pyramids, texture classification, denoising and signal compression to name just a few. Figures 11 and 12 show a multiresolution representation of an image and a trajectory that is progressively denoised.

5.2 Approximately Correct Trajectories

We now propose a method for speeding up the frequent trajectory mining phase. We discuss its applicability only for standard trajectory based mining using the standard Apriori algorithm described in section 3.2, as the extensions for the translational and rotational invariant mining are similar. Let us assume **dwt** is a function denoting the one-dimensional discrete wavelet transform. For example, given as input a vector x of length 1024, **dwt**(x) returns a vector x' of length 512, denoting the lower resolution version of vector x. The idea is that at the moment when the original *dense* trajectories are processed in order to convert them to their cell sequence representation, we apply **dwt** to each dense trajectory in order to get its lower resolution version. Then, we convert the lower resolution trajectory to a lower resolution cell sequence representation, where the cells now have *twice the width and height* they previously had. The effect of this is that the lower resolution cell sequence representation has approximately *half the length* of the original cell sequence representation. We refer to these new cell sequence representations as *scaled down* and we refer to their larger cells as *scaled down cells*. We can use these scaled down cell sequence representations to mine the frequent trajectories in our database at a coarser scale. This significantly decreases the length of our trajectories and the number of cells used to represent our region. As we discuss below, this can lead to significant improvements in the mining speed.

A potential objection to this procedure involves questioning the reason why we need to apply the wavelet transform in the first place. Some might argue that in order to get a smaller cell representation it suffices to simply double the cell size used in our equi-width/equidepth discretization of the trajectories. This question is similar to asking why we cannot simply decrease the resolution of a signal by subsampling the signal, i.e., selecting every second sample. This is a fundamental issue in the signal processing literature and the answer is that we need to do this for noise removal and for handling the aliasing problem [13]. The reason why we would want to remove noise and high frequency components is straightforward. We run into the risk that for various signals whose dense representation has a localized high frequency component, this might lead to incorrectly adding cells to a trajectory's cell sequence representation, especially if the noisy trajectory passes near a cell's border. By 'smoothing' a function we decrease the risk of dealing with functions which are not differentiable everywhere (Figures 9c and 9d). As the discrete wavelet transform is a linear algorithm, there is no significant pre-processing overhead involved. As we show in our experiments, this can significantly improve the mining speed, depending of course on the distribution of the trajectories in the data set we are currently working with. The drawback of this method is that we lose precision on the localization of the trajectory, since the cells are much larger. We deal with this in the next subsection.

5.3 Refining the Trajectories

Since the scaled down cell sequence representations consist of larger cells, they might give a worse localization of the trajectories. We now propose a method for refining the accuracy of the frequent trajectories that were mined using the above described method - in other words obtaining a better localization of the coordinates through which frequent trajectories pass. When mining for frequent trajectories we are often only interested in finding trajectories that have a minimum non-scaled down length of m. Let S denote the set of all scaled down cells through which a frequent scaled down trajectory of length at least $\lfloor \frac{m}{2} \rfloor$ passes. Then, we are guaranteed that the frequent trajectories with a non-scaled down cell sequence representation of length over m pass through the scaled down cells in S. This means that to refine the accuracy of mining frequent scaled down cell sequence representations, it suffices to consider only the non-scaled down cell sequence representations that pass through some cells in S. For datasets that have very spread out trajectories with a few paths through which frequent trajectories pass, this can also result in significant improvements in mining speed. In the next subsection we perform experiments on artificial data that we created and demonstrate the significant gains in speed that this approach can offer.

5.4 Experiments

We performed experiments on mining frequent trajectories of length over 30, to investigate the improvements in mining speed offered by the approach described in this section. We employed the dense trajectories used to create the first dataset from section 3.

First, we utilized the dense trajectories to create their cell sequences on a 64×64 grid, instead of the 32×32 grid used in section 3. All cell sequences on the 64×64 grid had length over 30. We applied the standard Apriori based method of section 3.2 to estimate the time it took to find the frequent trajectories on the 64×64 grid. In our experiments with MATLAB it took 26 minutes.

Then, we employed the standard Apriori based method of section 5.2 to mine the frequent trajectories on the scaled down 32×32 cell grid. It took 4 minutes for standard Apriori to terminate, a significant improvement in the mining speed, underscoring what a major effect the number of cells and the trajectory's cell sequence representation's length can have on the mining speed.

Then, we used the method described in section 5.3 to find the trajectories on the non-scaled down 64×64 grid. We removed from our dataset the non-scaled down cell sequence representations whose scaled down version in the 32×32 grid did *not* contribute to a frequent sequence of length over $\frac{30}{2} = 15$. The resulting mining speed was 13 minutes, for a total time to find the frequent trajectories of length over 30 of $4 + 13 = 17$ minutes, significantly better than the 26 minutes it took with the standard Apriori based method. The frequent cell sequence representations of length over 30 that were found are the same as those found using the algorithm in section 3, so no loss of frequent trajectories was observed.

There exist various types of datasets where this approach can lead to significant improvements in mining speed. This method improves the mining speed if there are clusters through which frequent trajectories of the desired length m pass, since it provides a quick method of finding which trajectories should be ignored from further processing. However, this method should not be considered a panacea. If our trajectories are evenly spread out around the region we do not expect to gain a lot in terms of mining speed.

It is possible that this process be repeated iteratively. In other words, we could further scale down the scaled down cell sequence representations and repeat this iteratively. For various datasets it is possible that this would lead to significant improvements in mining speed. Storing the scaled down cell sequence representations, in addition to the original non-scaled down ones, will at most double the database space requirements. This is so because on average each scaled down cell sequence representation has half the length of its previous representation and $1 + \frac{1}{2} + \frac{1}{4} + \cdots = 2$.

6. EXTENSION TO HIGHER DIMENSIONS

A problem with mining high dimensional trajectories (trajectories of dimension greater than 2) is that the number of cells grows by an entire factor for every increase in the dimension. For example, if we use equiwidth binning to split every dimension into n cells, then we are dealing with n distinct cells for 1D trajectories, n^2 distinct cells for 2D trajectories, n^3 distinct cells for 3D trajectories and so on. This can become computationally expensive, making the method we presented above infeasible in practice for dimensions greater than 2. We propose a method for handling this problem. We discuss the problem of mining standard trajectories, leaving the problem of mining frequent trajectories that are translated and rotated with respect to each other for future research.

Let us consider the 3D case. In 3D, any cell has $3^3 - 1 = 26$ neighbors. In other words if we are trying to generate the set C_k of length-k candidates, from set L_{k-1} of frequent length-$k-1$ cell sequences, an upper bound on the cardinality of C_k is $|C_k| \leq |L_{k-1}| \times 26$. On the other hand, assume that we first mined the x-coordinates of the trajectories, then mined the y-coordinates of the trajectories in our database and then mined the z-coordinates of the trajectories. Notice that in 1D, any cell has 3 neighbors, since it is possible that a sequence has two identical successive cells. In that case $|C_k| \leq |L_{k-1}| \times 3$ for all k, when mining the frequent cell sequences of each dimension. By applying the Apriori algorithm 3 times to discover the frequent cell sequences for each dimension and then recombining the sequences to form the frequent trajectories, we could potentially end up with significant improvements in the mining of trajectories, at least in terms of memory requirements. Consider the pseudocode shown in Figure 13.

```
Inputs: $minsup$: Minimum support count.
        $D$: Data set of cell sequences of trajectories.

Output: $L^x$,$L^y$,$L^z$, the frequent 1D trajectories of the
        $x$,$y$ and $z$ coordinates.

(1) Let $L_1^x$,$L_1^y$,$L_1^z$ be the frequent length-1 trajectories
    of the $x$,$y$,$z$ coordinates respectively.

(2) for (k=2;$L_{k-1} \neq \emptyset$;k++){

(3)     $C_k^x$ =trajectory($L_{k-1}^x$);
(4)     $C_k^y$ =trajectory($L_{k-1}^y$);
(5)     $C_k^z$ =trajectory($L_{k-1}^z$);

(6)     for each cell sequence $t \in D$ {

(7)         $C_t^x$ =subset($C_k^x, t$);
(8)         $C_t^y$ =subset($C_k^y, t$);
(9)         $C_t^z$ =subset($C_k^z, t$);

(10)            for each candidate $c \in C_t^x$,$c \in C_t^y$,$c \in C_t^z$
(11)                c.count++;

(12)    }

(13)    $L_k^x = \{c \in C_k^x | c.count \geq minsup\}$
(14)    $L_k^y = \{c \in C_k^y | c.count \geq minsup\}$
(15)    $L_k^z = \{c \in C_k^z | c.count \geq minsup\}$

(16) }

(17) return $L^x = \cup_k L_k^x$,$L^y = \cup_k L_k^y$,$L^z = \cup_k L_k^z$.
```

Figure 13: Pseudocode for extracting the three frequent 1D trajectories from a 3D trajectory.

We can assume without loss of generality that in the above piece of code we only retain the maximally frequent trajectories, *i.e.*, trajectories that do not have a frequent supersequence. The difficulty lies in recombining the frequent 1D trajectories into frequent 3-dimensional trajectories. A potential solution would involve searching each trajectory one by one. Assume each trajectory cell has a pointer to the maximally frequent x, y and z 1D trajectories it supports. Then by scanning each trajectory one by one, we can join the frequent x, y and z 1D trajectories it points to into a frequent 3D trajectory.

Memorywise it is obvious that this method is significantly less demanding. Based on the experiments discussed so far, we believe it will most likely also lead to significant improvements in mining speed. However, we do not test this in this paper and leave this problem for future research.

A direction worth pursuing as future work is to investigate using cells of varying sizes for splitting the original trajectory into its cell sequence representation. This could have a significant effect in terms of mining speed and could be useful in various applications, such as monitoring the paths followed by vehicles. For example, we could encode the entire length of a certain road by a single cell and use another cell only when the road is intersected by another road. If it is a long road, this could provide a significant optimization. Other optimizations could involve using an unsupervised clustering algorithm such as k-Means to split the range into intervals based on the density of trajectories passing through a certain region, or to investigate the use of equidepth binning for discretizing the trajectories. Depending on the application, we believe that this could prove to be very important in improving the speed of the algorithm.

Further research could focus on methods for mining translationally and rotationally invariant trajectories in 3D. The definition of curvature we have given in this paper applies only to 2D paths. More work is needed to come up with efficient ways of doing this for 3D paths $[x(s), y(s), z(s)]$.

7. CONCLUSIONS

We have presented various methods for mining frequent trajectories. We have also presented various methods for optimizing the mining of such trajectories. We believe these are appropriate steps in the right direction and we have pointed out issues that remain to be taken care of. More research needs to be done in using intelligent methods for discretizing the continuous range of values that the trajectories can assume, as this could potentially decrease the number of cells used to encode trajectories. Finally, a direction worth pursuing as future work is to find methods for mining the frequent trajectories without candidate generation.

Acknowledgements

The authors are grateful for the financial support of the National Science and Engineering Research Council (NSERC) and Ontario Graduate Scholarship (OGS).

8. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases*, 1994.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of ICDE'95*, 1995.

[3] J. Borges and M. Levene. Data mining of user navigation patterns. *Web Usage Analysis and User Profiling*, 2000.

[4] Y. Cai and R. Ng. Indexing spatio temporal trajectories with chebyshev polynomials. In *SIGMOD 2004*, 2004.

[5] H. Cao, N. Mamoulis, and D. Cheung. Mining frequent spatio-temporal sequential patterns. In *Proceeding of the ICDM*, 2005.

[6] M. Dimitrijevic. Mining for co-occuring motion trajectories - sport analysis. Master's thesis, University of British Columbia, Department of Computer Science, 2001.

[7] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[8] T. Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, 1994.

[9] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 2nd edition, 1999.

[10] S. G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7), July 1989.

[11] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. L. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *Conference on Knowledge Discovery in Data*, 2004.

[12] Y. Morimoto. Mining frequent neighboring class sets in spatial databases. In *Conference on Knowledge Discovery in Data*, 2001.

[13] A. Oppenheim. *Signals and Systems*. Barnes and Noble, 1996.

[14] J. Pei. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of ICDE 2001*, 2001.

[15] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. Int. Conf. on Data Engineering (ICDE'01)*, 2001.

[16] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *Proceedings of the 24th International Conference on Very Large Databases*, 1998.

[17] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference Extending Database Technology*, 1996.

[18] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *SIGKDD'03*, 2003.

[19] J. Wang, W. Hsu, and M. L. Lee. A framework for mining topological patterns in spatio-temporal databases. In *Conference on Information and Knowledge Management*, 2005.

[20] X. Xiong, M. F. Mokbel, and W. G. Aref. Processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *Proceedings of the International Conference of Data Engineering*, 2005.

[21] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *Proc. of ICDE*, 2005.