

Dimensionality Reduction for Long Duration and Complex Spatio-Temporal Queries

Ghazi Al-Naymat
University of Sydney

Sanjay Chawla
University of Sydney

Joachim Gudmundsson
National ICT Australia Ltd

Abstract

From tracking of moose in Sweden, to movement of traffic in a large metropolis, spatio-temporal data is continuously being collected and made available in the public domain. This provides an opportunity to mine and query spatio-temporal data with the purpose of finding substantial patterns and understand the underlying data generating process. An important class of queries is based on the flock pattern. A flock is a large subset of objects moving along paths close to each other for a certain pre-defined time. The standard approach to process a “flock query” is to map spatio-temporal data into a high dimensional space and reduce the query into a sequence of standard range queries which can be presented using a spatial indexing structure. However, as it is well known, the performance of spatial indexing structures drastically deteriorates in high dimensional space.

In this paper we propose a preprocessing strategy which consists of using a random projection to reduce the dimensionality of the transformed space. We prove an “ $\epsilon - \delta$ ” probabilistic approximation which results from the projection and present experimental results which show, for the first time, the possibility of breaking the curse of dimensionality in a spatio-temporal setting.

1. Introduction and Related Work

The most common type of spatio-temporal (ST) data consists of movement traces of point objects. The widespread availability of GPS enabled mobile devices and location-aware sensors have led to an explosion of generation and availability of this type of ST-data.

A unique example of a project which is continuously generating ST-data is related to the tracking of caribou (a type of reindeer) in Northern Canada. Since 1993 the movement of caribous is being tracked through the use of GPS collars with the underlying expectation that the data collected will help scientist understand the migration patterns of caribous and also help them locate their breeding and

calving locations [1]. While the number of caribous tagged at a given time is small the length of the temporal data associated with each caribou is long. As we will see one of the major challenges in ST query processing and pattern discovery is to efficiently handle “long duration” ST-data. Some very interesting ST-queries can be formulated on this particular data set. For example, *how are herds formed and does herd membership change over time? Are there specific regions in Northern Canada where caribou herds tend to rendezvous?*

A more sombre example project involving ST-data is related to the study of “pandemic preparedness”. How does a contagious disease get transmitted in a large city given the movement of people across the city? A recent workshop held on spatial data mining¹ exclusively focused on how to use spatial and spatio-temporal data mining techniques to help answer such questions. Some of the ST-queries that we will formulate and discuss can readily be applied to better understand this scenario.

To abstract the problem we assume that we are given an input which consists of a set P of n moving point objects p_1, \dots, p_n whose locations are known at τ consecutive time steps t_1, \dots, t_τ that is, the trajectory of each object is a polygonal line that can self-intersect. For brevity, we will call moving point objects *entities* from now on. Since the focus of the paper is to efficiently process known, albeit complex, query patterns, rather than mining for new patterns we will assume that the velocity of an entity along a line segment of the trajectory is constant.

The analysis and querying of ST data is an active research area within the database, computational geometry and data mining community. Much of the database research has focused on the querying and indexing ST-data [3, 11, 19]. A common theme in database research is to take an existing spatial query type and then study its generalizations to ST-data. A canonical example of this theme is the recent work on *continuous-kNN* querying over mobile data [18, 22]. The focus of our work is to study those queries which naturally arise in a ST setting and may not

¹<http://www.cs.dartmouth.edu/cbk/sdm06/>

have an equivalent spatial (or non-spatial) counterpart.

The focus within data mining research is to design techniques to discover new patterns in large repositories of ST-data. For example, Mamoulis et. al. [17] mine periodic patterns moving between objects, Ishikawa et. al [13] mine ST patterns in the form of markov transition probabilities. More recently Verhein et. al. [21] have proposed efficient association mining type algorithms for ST patterns such as, sinks, sources, stationary region, and thoroughfares. The emphasis in what we are proposing is not to mine new unknown patterns but efficiently query existing complex ST-patterns.

Design and analysis of algorithms for ST data is an active area of research within computational geometry. Laube and Imfeld [15] proposed the REMO framework (Relative MOtion) which defines similar behavior within groups of entities. They define a collection of spatio-temporal patterns based on similar direction of motion or change of direction. Laube et al. [16] extended the framework by not only including direction of motion, but also location itself. They defined several spatio-temporal patterns, including *flock*, *leadership*, *convergence*, and *encounter*, and gave algorithms to compute them efficiently. Later Gudmundsson et al. [10] extended the algorithmic results.

However, the above algorithms only consider each time step separately, that is, given $m \in \mathbb{N}$ and $r > 0$ a flock is defined by at least m entities within a circular region of radius r and moving in the same direction at some point in time. Benkert et al. [7] argued that this is not enough for many practical applications, e.g., a group of animals may need to stay together for days or even weeks before it is defined as a flock. They proposed the following definition of a flock:

Definition 1 ((m, k, r) -flock_A) - Given a set of n trajectories where each trajectory consists of τ line segments, a flock in a time interval $I = [t_i, t_j]$, where $j - i + 1 \geq k$, consists of at least m entities such that for every point in time within I there is a disk of radius r that contains all the m entities. Note that $m, k \in \mathbb{N}$ and $r > 0$ are given constants.

Assume that the movement of an entity from its position at time t_j to its position at time t_{j+1} is described by the straight-line segment between the two coordinates, and that the entity moves along the segment with constant velocity. Benkert et al. [7] proved that there is an alternative, and algorithmically simpler, definition of a flock that is equivalent.

Definition 2 ((m, k, r) -flock_B) - Given a set of n trajectories where each trajectory consists of τ line segments a flock in a time interval $[t_i, t_j]$, where $j - i + 1 \geq k$ consists of at least m entities such that for every discrete time step t_ℓ , $i \leq \ell \leq j$, there is a disk of radius r that contains all the m entities.

In the remainder of this paper we refer to Definition 2 whenever we discuss flocks. Using this model, Gudmundsson and van Kreveld [8] recently showed that computing the longest duration flock and the largest subset flock is NP-hard to approximate within a factor of $\tau^{1-\epsilon}$ and $n^{1-\epsilon}$ respectively.

Benkert et al. [7] described an efficient approximation algorithms for reporting and detecting flocks, where they let the size of the region deviate slightly from what is specified. Approximating the size of the circular region with a factor of $\Delta > 1$ means that a disk with radius between r and Δr that contains at least m objects may or may not be reported as a flock while a region with a radius of at most r that contains at least m entities will always be reported. Their main approach is a $(2 + \epsilon)$ -approximation with running time $T(n) = O(\tau n k^2 (\log n + 1/\epsilon^{2k-1}))$. Note that even though the dependency on the number of entities is small the dependency on the duration of the flock pattern is exponential. Thus, the main remaining open problem in [7] is to develop a good algorithm with a smaller dependency on k and this is exactly the focus of this paper.

1.1. Main Contribution and Outline

Current state-of-the-art algorithms for querying complex spatio-temporal patterns have an exponential dependency on the duration of the spatio-temporal data and pattern. In this paper we will show that we can use random projection(s) to manage the exponential dependency while provably retaining a high-quality solution. We will also present experimental results which will confirm that using random projection, as a preprocessing strategy, can effectively help overcome the ‘‘curse of dimensionality’’ for spatio-temporal pattern processing.

The rest of the paper is organized as follows. In Section 2 we will introduce and review the complex ST patterns for which we want to design efficient querying solutions. An overview of the relevant approximation algorithms will be provided in Section 3. In Section 4 our proposed solution of combining random projections with approximation algorithms to efficiently process long duration ST queries will be outlined. The experiment design and results will be discussed in Section 5. We will conclude in Section 6 with a summary and directions for future work.

2. Spatio-Temporal Queries

We focus on a class of spatio-temporal patterns proposed by Laube and Imfeld [15]. The class consists of four patterns namely Flock, Leadership, Encounter and Convergence. Other interesting patterns would be Recurrence and Concurrency Recurrence. Our focus will primarily be on

the *Flock* query but for completeness we enumerate all of them.

1. **Flock:** Parameters: $(m > 1, r > 0$ and $s)$. At least m entities are within a circular region of radius r and they move in the same direction during a time interval of length at least s . In Figure 1(a) objects o_1, o_2 and o_3 are participating in a flock pattern.
2. **Leadership:** Parameters: $(m > 1, r > 0$ and $\tau > 0)$. At least m entities are within a circular region of radius r , they move in the same direction, and at least one of the entities was already heading in that direction for at least τ time steps. In Figure 1(a), again the three objects o_1, o_2 and o_3 form a leadership pattern with o_3 as the leader.
3. **Encounter:** Parameters: $(m > 1, r > 0$, and $D)$. At least m entities will be concurrently inside the same circular region of radius r , assuming they move with the same speed and direction. In Figure 1(a), object o_1, o_2, o_3 and o_4 are heading towards an encounter in the shaded region.
4. **Convergence:** Parameters: $(m > 1, r > 0)$. At least m entities pass through the same circular region of radius r (not necessarily at the same time). In Figure 1(b) objects o_1, o_2, o_3, o_4 and o_5 are set to participate in a Convergence pattern.
5. **Recurrence:** Parameters: $(m > 1, k > 1, r > 1)$. At least m entities are visiting a circular region of radius r at least k times. Figure 1(c) shows an example of Concurrent Recurrence pattern where flock $f1$ repeatedly passes through the shaded circle.
6. **Concurrent Recurrence:** Parameters: $(m > 1, k > 1, r > 1)$. At least m entities are concurrently visiting a circular region of radius r at least k times. Figure 1(d) shows two flocks $f1$ and $f2$ are participating in a Concurrent Recurrence pattern.

It is clear from the above definitions that the *Flock* pattern is the most important and basic amongst all of them. The other pattern queries can be readily answered once an efficient way of querying the flock pattern is made available.

3. Approximate flocks

The nature of the ST patterns is such that they involve “sufficiently large” groups of entities passing or ready to pass through a “small” area. This is formalized by the parameters m and r in the definitions that represent the number of entities and radius of the region respectively. An exact value of m and r has no special significance - 20 caribou

meeting in a circle of radius 50 is as interesting as 19 caribou meeting in a circle of radius 51. Therefore the use of approximation algorithms is ideally suited for these scenarios [8].

In this section we generalise the approach suggested by Benkert et al. [7]. The input is a set P of n trajectories p_1, \dots, p_n , where each trajectory p_i is a sequence of τ coordinates in the plane $(x_1^i, y_1^i), (x_2^i, y_2^i), \dots, (x_\tau^i, y_\tau^i)$, where (x_j^i, y_j^i) is the position of entity p_i at time t_j .

3.1. Previous approach

The basic idea builds upon the fact that a polygonal line with d vertices in the plane can be modeled as a point in $2d$ dimensions. The trajectory of an entity p in the time interval $[t_i, t_j]$ is described by the polygonal line $p(i, j) = ((x_i, y_i), (x_{i+1}, y_{i+1}), \dots, (x_j, y_j))$, which corresponds to a point $p'(i, j) = (x_i, y_i, x_{i+1}, y_{i+1}, \dots, x_j, y_j)$ in $2(j - i + 1)$ -dimensional space.

The first step when checking whether there is a flock in the time interval $[t_i, t_{i+k-1}]$ is to map the polygonal lines of all entities to \mathbb{R}^{2k} .

The second, and key step, is to characterize the image of the flock in the higher dimensional space. It turns out that a flock can be expressed in terms of a high-dimensional *pipe* which we now define.

Definition 3 An (x, y, i, r) -pipe in \mathbb{R}^{2k} is the following region:

$$\{(x_1, \dots, x_{2k}) \in \mathbb{R}^{2k} \mid (x_i - x)^2 + (x_{i+1} - y)^2 \leq r^2\}.$$

An (x, y, i, r) -pipe is an unbounded region in \mathbb{R}^{2k} and contains all the points that are only restricted in two of the $2k$ dimensions (namely in dimensions i and $i + 1$) and when projected on those two dimensions lie in a circle of radius r around the point (x, y) . Equivalence 1 gives the key characterization of flocks.

Equivalence 1 Let $F = \{p_1, \dots, p_m\}$ be a set of entities and let $I = [t_1, t_k]$ be a time interval. Let $\{p'_1, \dots, p'_m\}$ be the mappings of F to \mathbb{R}^{2k} w.r.t. I . It holds that:

$$F \text{ is a } (m, k, r)\text{-flock} \iff \exists x_1, y_1, \dots, x_k, y_k : \forall p \in F : p' \in \bigcap_{i=1}^k (x_i, y_i, 2i-1, r)\text{-pipe}.$$

Equivalence 1 has been used by Benkert et al. [7] to design a Δ -approximation algorithm to find flocks ($\Delta > 1$) by performing a set of n range counting queries in the transformed space.

Definition 4 A Δ -approximation algorithm will report every (m, k, r) -flock, may or may not report an $(m, k, \Delta r)$ -flock and will not report a (m, k, r') -flock where r' exceeds Δr

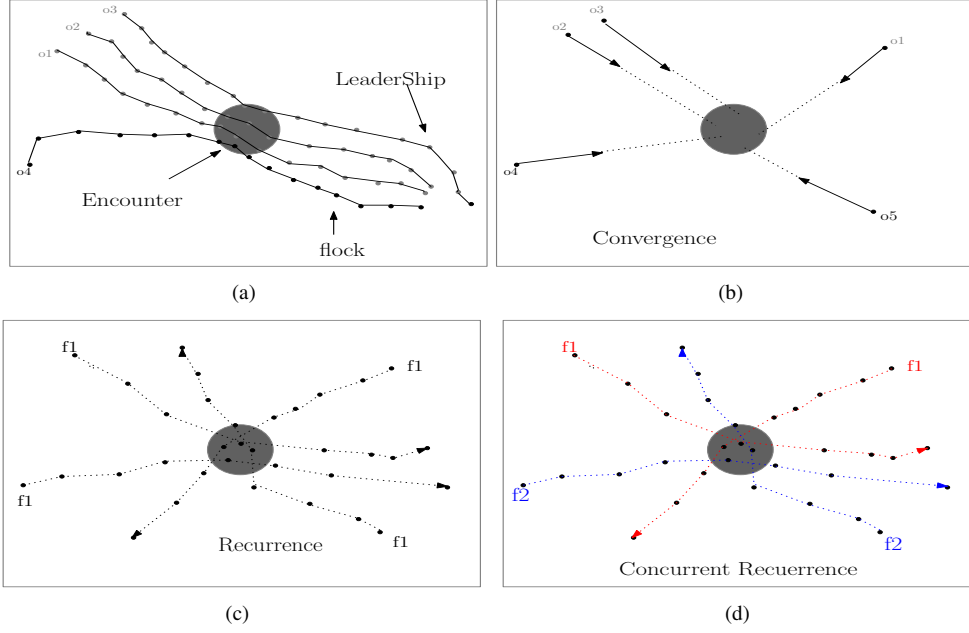


Figure 1. Spatio-Temporal Patterns

For each time interval $I = [t_i, t_{i+k-1}]$, where $1 \leq i \leq \tau - k + 1$, do the following computations. For each entity p let p' denote the mapping of p to \mathbb{R}^{2k} with respect to I . For each point $p' \in P$ perform a range counting query where the query range $Q(p')$ is the intersection of the k pipes $(x_i, y_i, 2i - 1, 2r)$ where (x_i, y_i) is the position of entity p at time step t_i . Every counting query containing at least m entities corresponds to an $(m, k, 2r)$ -flock according to [7]. Note that the same flock may be reported several times.

Since the theoretical bounds for answering range counting queries in high dimensions are close to linear Benkert et al. instead used $(1+\delta)$ -approximate range counting queries, to obtain the following result.

Fact 1 *The algorithm is a $(2+\delta)$ -approximation algorithm and requires $O(\tau n k^2 (\log n + 1/\delta^{2k-1}))$ time and $O(\tau n)$ space.*

We use the quadtree [20] structure as the building block for implementing the range query in the algorithm. Let $S = p_1, p_2, \dots, p_n$ be a set of n points in the plane contained in a square C of length l . A quadtree QT for S is recursively constructed as follows: The root of QT corresponds to the square C . The root has four children corresponding to the four squares of C of length $\frac{l}{2}$. The leaves of QT are the nodes whose corresponding square contains exactly one point. Using a compressed quadtree [4] for QT reduces its size to $O(n)$ by removing nodes not containing any points of S and eliminating nodes having only one

child. A compressed quadtree for a set n points in the plane can be constructed in $O(n \log n)$ time.

In [7] it was shown that this approach is very effective for small values of k , they performed experiments with k in the range of 4 to 16. However, since the running time has an exponential dependency on k it is obvious that it cannot handle long duration flocks.

3.2. Using random projection

To generalise the above approach to handle long duration flocks we extend the algorithm by adding a preprocessing step. The added step is intended to reduce the number of dimensions so that we can apply the algorithm in [7]. However, a flock is defined as a group of entities that at all times lie within a disk in the plane of radius r . If two entities lie within the same query range that means that the distance between them is at most $2r$ in every dimension. However, the distance between them in $2k$ -dimensions may be roughly $r \cdot \sqrt{2k}$. This observation makes it clear that we cannot perform a dimensional reduction and then use the same approach as above since the error will be too large.

However, if we modify the definition slightly the generalization can still be performed.

Definition 5 (m, k, r) -flock $_C$ - Given a set of n trajectories where each trajectory consists of τ line segments a flock f in a time interval $[t_i, t_j]$, where $j - i + 1 \geq k$ consists of at least m entities such that for every pair of entities $p, q \in f$ it holds that $\sum_{\ell=i}^j |p_\ell q_\ell| \leq r \cdot \sqrt{2k}$.

Instead of using a maximum of r in each time step we bound the sum of the differences. Intuitively this means that two entities that are very close to each other in all but one time step may still belong to the same flock in the new definition while this would not be possible in the definition by Benkert et al. .

Recall that in a step we consider a set of n points in \mathbb{R}^{2k} . Our first approach reduces the dimensions by using random projections by Johnson and Lindenstrauss [14], see also Achlioptas [2] and Indyk and Motwani [12]. The following theorem summarizes the tool:

Theorem 1 [2] *Let P be an arbitrary set of n points in \mathbb{R}^d , represented as an $n \times d$ matrix A . Given $\beta, \epsilon > 0$ let*

$$k_0 = \frac{4 + 2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n$$

For integer $k \geq k_0$, let R be a $d \times k$ random matrix with $R(i, j) = r_{ij}$, where r_{ij} are independent random variables from the following probability distributions:

$$r_{ij} = \sqrt{3} \times \begin{cases} +1 & \text{with probability } 1/6 \\ 0 & \text{.. } 2/3 \\ -1 & \text{.. } 1/6. \end{cases}$$

Let

$$E = \frac{1}{\sqrt{k}} AR.$$

Let $f : \mathbb{R}^d \rightarrow \mathbb{R}^k$ map the i^{th} row of A to the i^{th} row of E . With probability at least $1 - n^{-\beta}$, $\forall u, v \in P$

$$(1 - \epsilon)\|u - v\|^2 \leq \|f(u) - f(v)\|^2 \leq (1 + \epsilon)\|u - v\|^2$$

Recall that all the trajectories with k vertices in the plane can be modeled as a point in $2k$ dimensions. Thus after the transformation we have a set P of n points in $2k$ dimensions. Next, instead of performing n range counting queries we first apply the random projection on P to obtain a set P' of n points in $k' = \frac{4+2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n$ dimensions. Then, for each entity p perform a $(1 + \delta)$ -approximate range counting query where the query range $Q(p')$ is the k' -dimensional ball of radius $\sqrt{k'}$ and centre at p' which is the mapping of p in P' .

3.3. A theoretical analysis

Here we briefly give a probabilistic analysis on how the dimensional reduction affects the algorithm. We will say that an algorithm is an α -probabilistic $(m, k, \Delta r)$ -approximation algorithm if every (m, k, r) -flock is reported with probability α , an $(m, k, \Delta r)$ -flock may or may not be reported, while no $(m, k, \Delta' r)$ -flock will be reported with probability α .

Lemma 1 *The modified algorithm is a $(1 - n^{1-\beta})$ -probabilistic $(m, k, \Delta r)$ -approximation algorithm with running time $O(\tau n k^2 (\log n + 1/\delta^{2k'}))$ time, where $k' = \frac{4+2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n$ and $\Delta = 2(1 + \delta) \cdot (1 + \epsilon)^2$ for any constants $\epsilon, \delta > 0$ and $\beta > 1$.*

Proof. Consider the trajectories and let $P = \{p_1, \dots, p_n\}$ be the corresponding points in $2k$ -dimensions. Furthermore, let $f(p_i)$ be the point in P' corresponding to $p_i \in P$.

First we show that each (m, k, r) -flock f is reported by the algorithm with high probability. Let p_f be a point in P corresponding to an arbitrary entity of f and assume that f is a flock in the time interval $I = [t_i, t_{i+k-1}]$. We will prove that the approximation algorithm returns an $(m, k, \Delta r)$ -flock g such that $f \subseteq g$ with high probability and $\Delta = (1 + \delta) \cdot (1 + \epsilon)^2$.

According to Definition 5 there exists a $2k$ -dimensional ball with radius $r \cdot \sqrt{2k}$ that contains the points in P corresponding to the entities in f . According to Theorem 1 the corresponding points in P' will lie within a k' -dimensional ball of radius $(1 + \epsilon)r \cdot \sqrt{2k}$ with probability at least $(1 - n^{1-\beta})$, which is also the probability that a flock g containing the entities in f is reported.

For the stated bound to hold we also have to prove that the probability that an $(m, k, \Delta' r)$ -flock e is reported is small for $\Delta' > \Delta$. Let g be a flock reported by the algorithm in the interval I . The points in P' corresponding to the entities in g must lie within a k' -dimensional ball of radius $(1 + \delta) \cdot ((1 + \epsilon)r \cdot \sqrt{2k})$ and center at a point $f(p') \in P'$. Recall that the algorithm performs $(1 + \delta)$ -approximate range counting queries. Using Theorem 1 it then follows that the maximum distance between p' and every point corresponding to an entity in g is at most $(1 + \epsilon)((1 + \delta) \cdot ((1 + \epsilon)r \cdot \sqrt{2k})) = (1 + \delta) \cdot (1 + \epsilon)^2 r \cdot \sqrt{2k}$ with probability $(1 - n^{1-\beta})$. From the triangle inequality we get that the distance between any two points in g is bounded by $2(1 + \delta) \cdot (1 + \epsilon)^2 r \cdot \sqrt{2k}$ which completes the proof of the lemma. \square

3.4 Random projection in a DBMS

In 2003 [2], Achlioptas showed how a random projection can be carried out using “database friendly” operations, i.e., computing a random projection without actually generating the matrix with random entries. We clarify the process with the help of an example shown in Figure 2:

Given an integer k and a table A of dimension $size(A) = n \times d$, where n is the number of rows, d is the number of columns and k is the reduced dimension. We want to generate a new table E of size $n \times k$ without

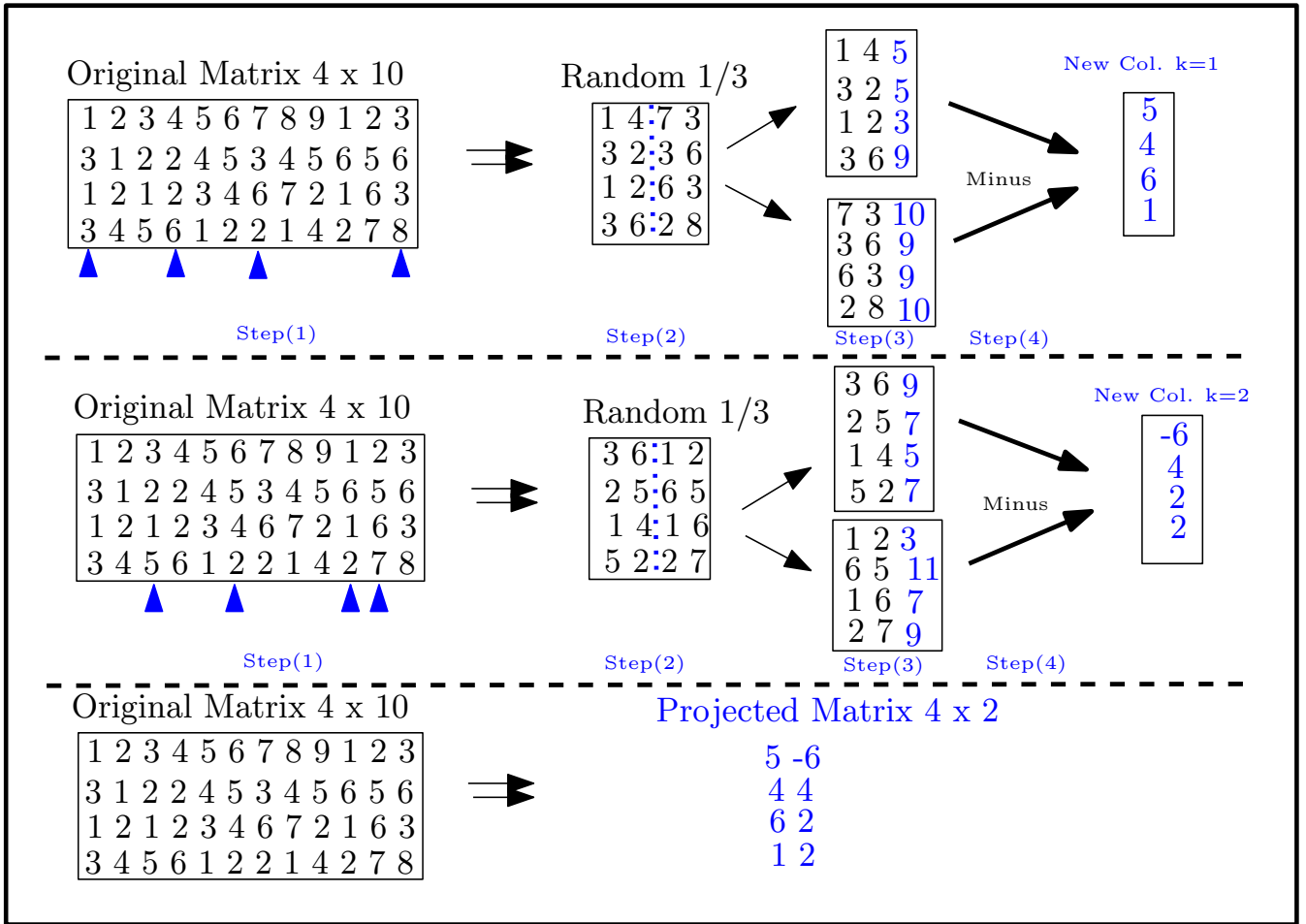


Figure 2. Random Projection Using "DB Friendly" Operations

actually multiplying A with a randomly generated matrix.

$$C = \prod_{e/2+1}^e A'$$

For each i , $1 \leq i \leq k$ we produce a new field in the new table and then at the end we merge them together. Effectively, we repeat for each i the following steps:

1. Uniformly at Random select 1/3 of A 's attributes. Call the obtained table A' . In other words, we project A to A' .

$$A' = \prod A_{d/3}$$

the size of A' is $n \times e$, where e is $d/3$ of the columns in the original table.

2. Uniformly at random vertically partition A' into two equal column-sized tables B and C .

$$B = \prod_1^{e/2} A'$$

3. We combine B and C to generate a new column as follows. We first create a $NewCol_1$ from B and $NewCol_2$ from C .

$$NewCol_1 = \left\{ \begin{array}{c} \sum_{j=1}^{e/2} B_{1,j} \\ \vdots \\ \sum_{j=1}^{e/2} B_{n,j} \end{array} \right\}$$

$$NewCol_2 = \left\{ \begin{array}{c} \sum_{j=1}^{e/2} C_{1,j} \\ \vdots \\ \sum_{j=1}^{e/2} C_{n,j} \end{array} \right\}$$

4. The i^{th} column of table E is defined as $NewCol_2 - NewCol_1$.

4. Experiments, Results and Discussion

We now report on the experiments that we have carried out to determine how the use of random projections will affect the accuracy of detecting flocks in long duration spatio-temporal data sets. We also report on the overheads caused because of preprocessing the data with a random projection.

4.1 Experimental Setup

All experiments were carried out on a Linux operated PC with a Pentium-4.3.6 GHz processor and 2 GB main memory. The data structures and algorithms were implemented in C++ and compiled with the Gnu compiler.

4.2 Data Set and Parameters

We used a synthetic data set as it allowed us to control the number of flocks present in the data. This helped in determining the correctness of our approach.

Twelve data sets with varying number of points, number of flocks and duration of flocks were created. In particular, three data sets each of size 16K, 20K and 32K were seeded with 32, 40 and 64 flocks respectively of duration (time step) 8, 16, 500 and 1000.

The size of each flock was set to 50 entities and the radius was fixed to 50 units. In the original data (before the random projection), each point coordinate was selected from the interval $[0, \dots, 2^{16}]$.

In our experiments we always looked for flocks of at least 50 entities in a circle with radius 50 and full time duration (8, 16, 500 or 1000).

4.3 Random Projection Parameters

The theoretical lower bound for random projection is of the order $O(\frac{\ln(n)}{\epsilon^2})$. Notice that the bound is independent of the dimensionality of the original space. For a data set of size 32K and a distortion of $\epsilon = 0.5$, the dimensionality of the projected space will be at least 165. This is too large for the quadtree indexing structure to handle. However, several research studies[5, 6] have noted that the theoretical bound is quite conservative. For example, Bingham and Manilla [5] have noted that for *their* image data the theoretical bound required was $k \approx 1600$ but a “ $k \approx 50$ was enough.”

In order to test the difference between the theoretically derived and experimentally acceptable bound (k) we carried out several experiments for different values of the size of the

data set (n), dimensionality(d), the error tolerance (ϵ) and and the confidence (β). We report on two sets of parameters as shown in Figure 3. Notice that while the theoretical bound depends upon n (as expected) the experimentally derived bound quickly stabilizes for different values of n . This confirms that the theoretical bound is too conservative and that for practical applications we can use a much lower k .

In the end we settled for projecting the data to a 32 dimension space. The primary reason being that the experiments reported in [7] only go up to sixteen time steps (32 dimension) after which the authors note that a range query search using quadtree effectively reduces to a linear scan. Since random projection is being used as a pre-processing step, it is reasonable to project to a space where the use of quadtree is still effective.

We may also use a different technique to reduce the number of dimensions. This step is performed on the coordinates obtained from the random projection. Given a positive constant $\gamma < 1$ randomly choose $k'' = \gamma k'$ coordinate pairs, thus the point set P' in $\mathbb{R}^{2k'}$ can be reduced to a set P'' of n points in $\mathbb{R}^{2k''}$. However, this option is not pursued here.

4.4 Results

The results of the experiments are shown in Table 1. The trends are more easily noticeable in Figure 4.

The two basic methods we used are *Brute-Force (BF)* and *Pipe*. The BF method does not use any index. It consists of two nested loops, the outer one specifying a potential flock center and inner one computing the distance between a point and the potential flock center. If there are at least m points within a ball of radius $2r$ centered at the potential flock center then a flock is reported. In that respect, the BF method is a 2-approximation. Notice that the complexity of the BF method is quadratic in the number of entities and does not depend upon the number of time stamps. This explains the small increase in time (Table 1, rows 3 and 4) for 16K points from 50 seconds to 59 seconds as the number of time steps (TS) increase from 500 to 1000.

The Pipe method is based on Equivalence 1 (Section 3.1) and uses a compressed quadtree as the underlying indexing structure. As expected, the Pipe method beats BF for low dimensions. For example, for data of size 32K, BF takes 2 and 8 seconds to find the 64 flocks for time duration 8 and 16 respectively (rows 9 and 10). For high dimensions (500 and 1000) the quad tree provides no extra advantage because of the “curse of high-dimensionality”: the internal nodes of a quadtree has 2^d children where d is the number of dimensions. Using 16 timestamp means 32 dimensions which translates to more than 4,000 million quadrant. It is very unlikely that the 32K randomly distributed points (not in flocks) fall into the same quadrant. This results in a vary

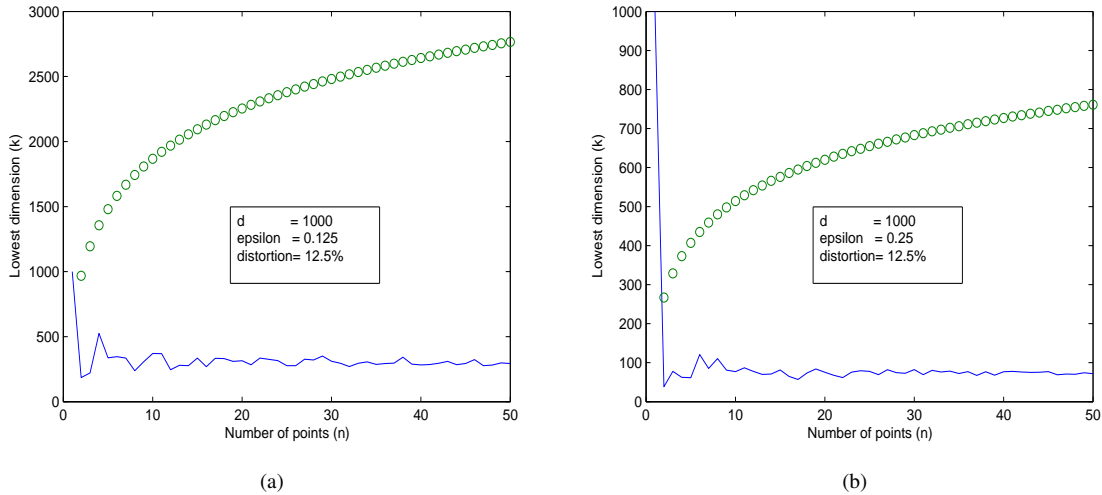


Figure 3. Difference between the theoretical and experimental dimensionality reduction bounds. The top line is the theoretical bound and the bottom line is derived experimentally using a brute force procedure.

flat tree and high running time.

We did not apply the random projection pre-processing step for time steps 8 and 16. The overall cost of random projection is $O(ndk)$ which is around 1 second for the largest data set (32K and 1000 time steps).

The **most important** result with the random projection is that we retrieve exactly the same number of flocks as retrieved without the random projection. This shows that the distortion induced by the random projection is within acceptable bounds and does not violate the overall correctness.

5. Summary and Future Work

Given a large spatio-temporal data set, an important query is to retrieve objects which move along paths which are close to each other for a long duration of time. This is called the *Flock* pattern query and is the basis of many important spatio-temporal pattern queries. Till date, the best known approximation algorithm for such queries have an exponential dependency on the the duration parameter of the query pattern. We have proposed the use of random projection as a practical solution to manage the exponential dependency. We have proved that the random projection will return the “correct” answer with high probability and follow it up with an experimental confirmation on several synthetic data sets.

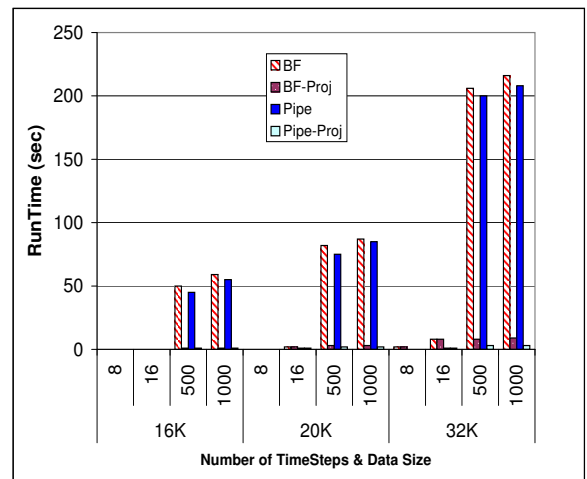


Figure 4. Results- With and Without Random Projection

References

- [1] Porcupine caribou herd satellite collar project. <http://www.taiga.net/satellite/>.
- [2] D. Achlioptas. Database-friendly random projections. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003.
- [3] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *Journal of Computer and System Science*, 66(1):207–243, 2003.

| Data | | | Brute Force(BF) | | Brute Force-With Proj | | Pipe | | Pipe-With Proj | |
|------|-----------|------|-----------------|-----|-----------------------|------|----------|-----------|----------------|------|
| | Data Size | # TS | # Flocks | Sec | # Flocks | Sec. | # Flocks | Sec. | # Flocks | Sec. |
| 1 | 16K | 8 | 32 | < 1 | N.A | N.A | 32 | < 1 | N.A | N.A |
| 2 | 16K | 16 | 32 | < 1 | N.A | N.A | 32 | < 1 | N.A | N.A |
| 3 | 16K | 500 | 32 | 50 | 32 | < 2 | 32 | (BF) 50+ | 32 | < 2 |
| 4 | 16K | 1000 | 32 | 59 | 32 | < 2 | 32 | (BF) 59+ | 32 | < 2 |
| 5 | 20K | 8 | 40 | < 1 | N.A | N.A | 40 | < 1 | N.A | N.A |
| 6 | 20K | 16 | 40 | 2 | N.A | N.A | 40 | 1 | N.A | N.A |
| 7 | 20K | 500 | 40 | 82 | 40 | 3 | 40 | (BF) 82+ | 40 | 2 |
| 8 | 20K | 1000 | 40 | 87 | 40 | 3 | 40 | (BF) 87+ | 40 | 2 |
| 9 | 32K | 8 | 64 | 2 | N.A | N.A | 64 | < 1 | N.A | N.A |
| 10 | 32K | 16 | 64 | 8 | N.A | N.A | 64 | 1 | N.A | N.A |
| 11 | 32K | 500 | 64 | 206 | 64 | 8 | 64 | (BF) 206+ | 64 | 3 |
| 12 | 32K | 1000 | 64 | 216 | 64 | 9 | 64 | (BF) 216+ | 64 | 3 |

Table 1. Results: With and Without Random Projection

- [4] S. Arya, D. Mount, N. Netanyahu, and R. Silverman. An optimal algorithm for approximate nearest searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.
- [5] E. Bingham and H. Manilla.
- [6] D. Fradkin and D. Madigan. Experiments with random projections for machine learning. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 517–522, New York, NY, USA, 2003. ACM Press.
- [7] J. Gudmundsson, M. Benkert, F. Huebner, and T. Wollé. Reporting flock patterns. 2006.
- [8] J. Gudmundsson and M. van Kreveld. Computing longest duration flocks in spatio-temporal data. Manuscript, June 2006.
- [9] J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of motion patterns in spatio-temporal data sets. In *GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 250–257, New York, NY, USA, 2004. ACM Press.
- [10] J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of motion patterns in spatio-temporal sets. *GeoInformatica*, 2006. To appear.
- [11] R. Gutting, M. Bohlen, M. Erwig, C. Jensen, N. Lorentzos, E. Nardelli, M. Schneider, and J. R. Viqueira. Spatio-temporal models and languages: An approach based on data types. In *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, pages 117–176. Springer-Verlag, 2003.
- [12] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *30th ACM Symposium on Theory of Computing*, pages 604–613, 1998.
- [13] Y. Ishikawa, Y. Tsukamoto, and H. Kitagawa. Extracting mobility statistics from indexed spatio-temporal datasets. In *STDBM*, pages 9–16, 2004.
- [14] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. In *In Conference in modern analysis and probability*, pages 189–206. Amer. Math. Soc, 1982.
- [15] P. Laube and S. Imfeld. Analyzing relative motion within groups of trackable moving point objects. In *GIScience '02: Proceedings of the Second International Conference on Geographic Information Science*, pages 132–144, London, UK, 2002. Springer-Verlag.
- [16] P. Laube, M. van Kreveld, and S. Imfeld. Finding REMO – detecting relative motion patterns in geospatial lifelines. In *11th International Symposium on Spatial Data Handling*, pages 201–214, 2004.
- [17] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 236–245, New York, NY, USA, 2004. ACM Press.
- [18] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou. Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 634–645, New York, NY, USA, 2005. ACM Press.
- [19] S. C. O. Wolfson, B. Xu and L. Jiang. Moving objects databases: Issues and solutions. In *10th International Conference on Scientific and Statistical Database Management*, pages 111–122, 1998.
- [20] H. Samet. *The design and analysis of spatial data structures*. Addison-Wesley, 1990.
- [21] F. Verhein and S. Chawla. Mining spatio-temporal association rules, sources, sinks, stationary regions and thoroughfares in object mobility databases. In *Database Systems for Advanced Applications: 11th International Conference, DASFAA*, pages 187 – 201, Singapore, 2006. Springer Berlin-Heidelberg.
- [22] X. Xiong, M. F. Mokbel, and W. G. Aref. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *ICDE*, pages 643–654, 2005.