Distributed Spatio-Temporal Similarity Search

Demetrios Zeinalipour-Yazti
Dept. of Computer Science
University of Cyprus
1678, Nicosia, Cyprus
dzeina@cs.ucy.ac.cy

Song Lin
Dept. of Comp. Sci. and Eng.
Univ. of California, Riverside
Riverside, CA 92521, USA
slin@cs.ucr.edu

Dimitrios Gunopulos
Dept. of Comp. Sci. and Eng.
Univ. of California, Riverside
Riverside, CA 92521, USA
dg@cs.ucr.edu

ABSTRACT

In this paper we introduce the *distributed spatio-temporal similarity search* problem: given a query trajectory Q, we want to find the trajectories that follow a motion similar to Q, when each of the target trajectories is segmented across a number of distributed nodes. We propose two novel algorithms, UB-K and UBLB-K, which combine local computations of lower and upper bounds on the matching between the distributed subsequences and Q. Such an operation generates the desired result without pulling together all the distributed subsequences over the fundamentally expensive communication medium. Our solutions find applications in a wide array of domains, such as cellular networks, wildlife monitoring and video surveillance. Our experimental evaluation using realistic data demonstrates that our framework is both efficient and robust to a variety of conditions.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]:

General Terms

Algorithms, Design, Performance, Experimentation

Keywords

Spatio-temporal Similarity Search, Top-K Query Processing

1. INTRODUCTION

The advances in networking technologies along with the wide availability of GPS technology in commodity devices, make spatiotemporal records nowadays ubiquitous in many different domains including cellular networks, wildlife monitoring and video surveillance. The enormous growth in spatiotemporal records in conjunction with the emerging in-network storage model, constitute centralized spatiotemporal query processing techniques obsolete in many respects.

CIKM'06, November 5–11, 2006, Arlington, Virginia, USA.

To stimulate our description consider the *Enhanced 911 (e911)*[1] service, which was recently enforced by the Federal Communications Commission (FCC) to all US cellular service providers. In *e911*, each provider must be able to locate wireless 911 callers within a 50 to 300 meters accuracy, when required. In order to satisfy the FCC requirements, carriers had the choice to either add GPS technology into their cell phones (the *handset solution*), or to estimate the position of a caller using the timing of signals emitted from the phone to the base station (the *network solution*). The bottom line of both approaches, is that base stations scattered around US neighborhoods must be able to provide the precise location of any cell phone user at any given moment. In the event of a 911 call, the accurate location information will be transmitted towards the local state and government agencies that can take further action. An important point is that the generated data remains *in-situ*, at the base station that generated the data, until some event of interest occurs.

The above example shows three important points : i) spatiotemporal data becomes available in an ever growing number of applications; ii) organizations realize that a distributed data storage and query processing model is in many occasions more practical than storing everything centrally. A category of applications for which this is particularly true, are sensor and RFID-related technologies that try to capture the physical world at a high fidelity; and iii) many of the generated spatiotemporal records might become outdated before they are ever utilized (for instance a cell phone user might never actually make a 911 call), which again shows that centralization might be a wasteful approach.

In this paper we propose techniques to overcome the inherent problems of the centralized scenario. Specifically, we formulate the *Distributed Spatio-Temporal Similarity Search* problem and devise techniques to solve this problem efficiently. To formalize our description, let A denote a spatiotemporal trajectory defined as a sequence of l multidimensional tuples $\{a_1, ..., a_l\}$. Each tuple is characterized by two spatial dimensions and one temporal dimension (i.e. $a_i(x_i, y_i, t_i)$, $\forall i \in [1..l]$). A *segment* or *subsequence* of a trajectory A, is defined as a collection of r consecutive tuples $[a_i..a_{i+r}]$ (i+r$\leq$ l). Note that the segments of each trajectory A, are located at different remote sites, depending on the site that collected the data. In real applications a trajectory will usually span many such sites, depending on the coverage provided by each access point.

[1]http://www.fcc.gov/911/enhanced/

We denote the natural fragmentation of each trajectory as *spatial fragmentation*, because a trajectory is sliced up into several disjoint subsequences which reside on spatially distributed sites. Our objective is to answer the query: *"Report the objects (i.e. trajectories) which follow a similar spatio-temporal motion with Q"*, where Q is some query trajectory. The notion of *similarity* captures the trajectories which differ only slightly, in the whole sequence, from the given search query Q. More formally, the tuples of each target trajectory A, are compared with the points of Q within some temporal and spatial window. Other queries, such as *pattern* queries [12], which look at the pattern of a trajectory rather than individual points, are similarly interesting but outside of the scope of this paper.

Research to this day, has focused on computing similarity queries assuming that the querying entity has access to all the trajectories in advance, or becomes aware of them in a streaming fashion (Section 2 provides an overview). While the centralized model serves well many scenarios where the transfer of data is inexpensive, it is not appropriate for environments with expensive communication mediums, such as wireless sensor networks [16], or environments where the distributed sites generate large quantities of spatio-temporal records (e.g. the e911 scenario).

Our approach is optimized for retrieving the K most similar trajectories to a query Q, for a user parameter K. Therefore the queries do not retrieve the whole universe of answers. Additionally, the techniques we propose employ trajectory matching techniques that have been shown to be accurate and tolerant to noise and outliers while featuring an extremely low computational overhead. In this paper we mainly use the *Longest Common Subsequence (LCSS) [9]* as a distance measure, but the techniques can easily be extended to work with *Dynamic Time Warping (DTW) [5]* as well. Our main contributions are summarized as following:

1. We introduce and formalize the problem of finding the most similar spatio-temporal trajectories in a distributed environment.
2. We propose the *UB-K* and *UBLB-K* algorithms, which are distributed query processing algorithms that find the K most similar trajectories to a query trajectory Q, by utilizing locally computed lower and upper bounds on the trajectory similarity function.
3. We propose *DUB_LCSS* and *DLB_LCSS*, which are distributed similarity approximation algorithms that can accurately upper and lower bound the Longest Common Subsequence (*LCSS*) similarity.

The remainder of the paper is organized as follows: Section 2 provides an overview of related research, Section 3 formulates the problem and our notation. Section 4 describes our distributed query processing algorithms, *UB-K* and *UBLB-K*, which utilize upper and lower bound scores on a variety of distance measures in order to compute the K most similar trajectories to a query trajectory Q. The exact mechanism of generating the upper bounds (*DUB_LCSS*) and lower bounds (*DLB_LCSS*) is described in Section 5. In Section 6, we present an experimental study of our algorithms using 25,000 car trajectories moving in the city of Oldenburg (Denmark) and Section 7 concludes the paper.

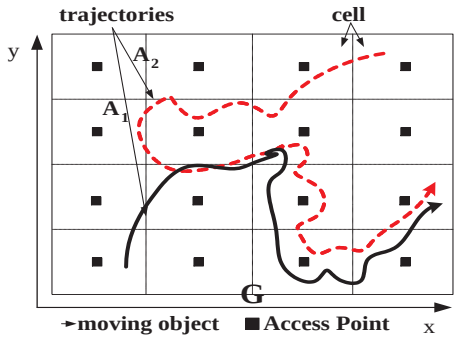

Figure 1: The trajectories A_1 and A_2 of two moving objects in space G. Each cell contains an access point that records subsequences of A_1 and A_2.

2. RELATED WORK

To the best of our knowledge the distributed spatiotemporal similarity search problem has not been addressed in the literature before. However spatio-temporal queries have been an intense area of research over the last years [1, 4, 13, 19, 20, 22, 24, 26]. This resulted in the development of efficient access methods [13, 15, 22, 24] and similarity measures [5, 9, 24] for predictive [23], historical [24] and complex spatio-temporal queries [12]. All these techniques, as well as the frameworks for spatio-temporal queries [18, 21, 25], work in a completely centralized setting. Our techniques on the other hand are decentralized and keep the data *in-situ*, which is more appropriate for environments with expensive communication mediums and for large scale applications that generate huge amounts of spatiotemporal records.

One problem with the in-situ storage of trajectories is that query processing now becomes significantly more complex. Finding similar trajectories in a distributed fashion might require sophisticated techniques and interactions to uncover the potentially very large number of answers. Note that a query of the type *"Find which other trajectories are similar to trajectory Q"* yields fuzzy answers, thus it is meaningful to limit the cardinality of the answer set to some user defined threshold K. Otherwise, the user would end up retrieving a large number of less relevant answers. Solutions to the above *Top-K query processing* problem, have traditionally been provided by the database community in a variety of contexts including middleware systems [10, 11], web accessible databases [7, 17, 27], stream processors [2], peer-to-peer systems [3] and other distributed systems [8, 28].

In general, a Top-K query returns the K highest ranked answers to a user defined similarity function. For instance the query by example: *"Find the K=5 images that are most similar to some query image Q"*, returns the five pictures that minimize the average distance for a set of given dimensions (e.g. using features such as color, texture, etc). A top-k query returns a subset of the complete answer set, in order to minimize some cost metric that is associated with the retrieval of the complete answer set. Such a cost is usually measured in terms of disk accesses or network transmissions, depending on where the data physically resides.

The TA [11] algorithm and its variants are well established and understood algorithms for computing top-k queries in a centralized setting. A fundamental assumption underlying

these algorithms is that the exact score is available for each dimension of the similarity function. For instance, given some image p_i and some query image Q, we have a similarity score associated with each of its dimensions (i.e. 0.7 similarity with respect to color, 0.94 similarity with respect to texture etc). The total similarity of p_i and Q, is then simply be the average of these scores (i.e. 0.82). Exact scores are also the underlying assumption of distributed top-K query processing algorithms proposed in recent literature, namely the *TPUT* [8], *TJA* [28] and *TPAT* [14].

Unfortunately such exact scores are not available in our setting and therefore none of the above top-k query processing solutions can be utilized in our case. To understand this first assume that we map, using a 1:1 correspondence, each query dimension to a distributed site. The most similar trajectory A is the one that maximizes the similarity to Q across all dimensions (i.e. all sites).

A naive solution would be to calculate some exact similarity score at each remote site and then combine these scores using any of the aforementioned top-k query processing algorithms. For instance, by utilizing the *Euclidean distance* *(L_2)*, given as $|Q-A| = \sqrt[2]{\sum_{i=1}^{l} |q_i - a_i|^2}$, one can produce a set of exact scores, which express the distance of Q to A. However the matching between Q and A, would only occur between points of identical time positions. As a result, it would neither be flexible to out-of-phase matches (e.g. if we have two identical trajectories but the first one moves earlier in time) nor tolerant to noisy data (e.g. we have two identical trajectories but the first has some slight deviation in its spatial movement). Disregarding these parameters might result in an extremely poor similarity matching.

Therefore we opt for a multidimensional similarity measure that takes into account out-of-phase matches and gracefully handles noisy data. In particular, we use the Longest Common Subsequence that will be described in Section 5.2. Using such a measure in a distributed environment limits us to a lower and upper bound on the similarity score between a query trajectory Q and A_i. For instance, we might only know that the similarity of Q to some subsequence A_1 is between 0.88 and 0.92 and that the similarity of Q to A_2 is between 0.85 and 0.90. Thus, we can not determine which of the two trajectories is more similar to Q. For instance if the real similarity is $FullM(Q, A_1) = 0.89$ and $FullM(Q, A_2) = 0.90$ then A_2 is the most similar one; on the contrary if $FullM(Q, A_1) = 0.89$ and $FullM(Q, A_2) = 0.87$ then A_1 is the most similar.

The above description shows that by having score bounds, rather than exact scores, is not enough to identify the most similar trajectories. This creates a challenging problem: How to calculate the top-K answers if we have score bounds rather than exact scores? We will provide two novel algorithms that solve this problem in an iterative fashion. Such algorithms might potentially have many other applications outside the spatiotemporal similarity search domain, although we will not explore these possibilities here.

3. PROBLEM FORMULATION

In this section we provide the notation used throughout the paper. Specifically, we formalize our data and query model. The main symbols and their respective definitions are summarized in Table 1.

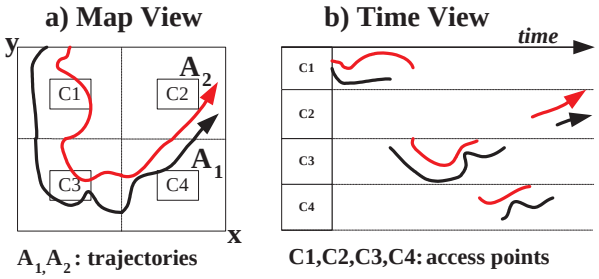

Figure 2: Two Spatially Fragmented Trajectories A_1, A_2 across four cells $C1, C2, C3, C4$.

Symbol	Definition
G	A Given Geographic Area
QN	The Querying Node
Q	A Spatio-Temporal Query
n	Number of Cells in G
m	Number of Trajectories (Objects)
l	Trajectory Length (discrete points)
K	Number of requested results
$LowerM(Q,A)$, $UpperM(Q,A)$, $FullM(Q,A)$	The Lower, Upper Bounding and Full Match between the query Q and the trajectory A. Larger Match means Smaller Distance

Table 1: Symbol Description.

3.1 The Data Model

Let G denote a 2-dimensional matrix of *points* in the xy-plane that represents the coordinate space of some geographic area. Without loss of generality, we assume that the points in G are logically organized into $x \cdot y$ *cells* as illustrated in Figure 2. Each cell contains an *access point* (AP) that is assumed to be in communication range from every point in its cell.[2]

Although the coordinate space is assumed to be partitioned in square cells, other geometric shapes such as variable size rectangles or Voronoi polygons are similarly applicable but outside the scope of this paper. This partitioning of the coordinate space simply denotes that in our setting, G is covered by a set of APs. Now let $\{A_1, A_2, ..., A_m\}$ denote a set of m objects moving in G. At each discrete time instance, object A_i ($\forall i \leq m$) generates a spatio-temporal record $r = \{A_i, t_i, x_i, y_i\}$, where t_i denotes the timestamp on which the record was generated, and (x_i, y_i) the coordinates of A_i at t_i. The record r is then stored locally at the closest AP for l discrete time moments after which it is discarded. Therefore at any given point every access point AP maintains locally the records of the last l time moments.

A trajectory can be conceptually thought of as a continuous sequence $A_i = ((a_{x:1,y:1}), ..., (a_{x:l,y:l}))$ ($i \leq m$), while physically it is spatially fragmented across several cells (see Figure 2). Similarly, the spatio-temporal query is also represented as: $Q = ((q_{x:1,y:1}), ..., (q_{x:l,y:l}))$ but this sequence is not spatially fragmented.

3.2 The Query Model

Our objective is to answer efficiently top-K queries of the type: *given a trajectory Q, retrieve the K trajectories which*

[2]The terms *access point* and *cell* are used interchangeably.

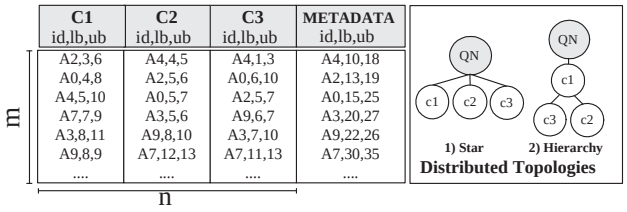

Figure 3: (a) *METADATA*: Lower and Upper bounds computed for m trajectories. (b) Distributed Topologies.

are the most similar to Q. First note that the similarity query Q is initiated by some querying node QN, which disseminates Q to all cells that intersect the query Q. We call the intersecting regions *candidate cells*. Upon receiving Q, each candidate cell executes locally a lower bounding matching function ($LowerM$) and an upper bounding function ($UpperM$) on all its local subsequences (these functions will be described in Section 5.2). This yields $2{\cdot}m$ local distance computations to Q by each cell (one for each bound). To speed up computations we could utilize spatiotemporal access methods similar to those proposed in [24]. The conceptual array of lower (**LB**) and upper bounds (**UB**) for an example scenario of three nodes (C_1, C_2, C_3) is illustrated in Figure 3a. We will refer to the sum of bounds from all cells as *METADATA* and to the actual subsequence trajectories stored locally by each cell as *DATA*. Obviously, *DATA* is orders of magnitudes more expensive than $METADATA$ to be transferred towards QN. Therefore we want to intelligently exploit $METADATA$ to identify the subset of $DATA$ that produces the K highest ranked answers. Figure 3b illustrates two typical topologies between cells: star and hierarchy. Our proposed algorithms are equivalently applicable to both of them although for the remainder of the paper we use a star topology to simplify our description.

In order to find the K trajectories that are most similar to a query trajectory Q, QN can fetch all the *DATA* and then perform a centralized similarity computation using the $FullM(Q, A_i)$ ($\forall i \leq m$) method, which is one of the LCSS, DTW or other L_p-Norm distance measures presented in Section 5. *Centralized* is extremely expensive in terms of data transfer and delay.

4. DISTRIBUTED QUERY PROCESSING

In this section we present two novel distributed query processing algorithms, *UB-K* and *UBLB-K*, which find the K most similar trajectories to a query trajectory Q. The UB-K algorithm uses an upper bound on the matching between Q and a target trajectory A_i, while UBLB-K uses both a lower and an upper bound on the matching. The description on how these bounds are acquired is delayed until Section 5.

4.1 The UB-K Algorithm

The *UB-K* algorithm is an iterative algorithm for retrieving the K most similar trajectories to a query Q. The algorithm minimizes the number of *DATA* entries transferred towards QN by exploiting the upper bounds from the *METADATA* table. Notice that *METADATA* contains the bounds of many objects that will not be in the final top-K result. In order to minimize the cost of uploading the complete *METADATA* table to QN we utilize a distributed top-K

Algorithm 1 : UB-K Algorithm

Input: Query Q, m Distributed Trajectories, Result Parameter K, Iteration Step λ.
Output: K trajectories with the largest match to Q.

1. Run any distributed top-K algorithm for Q and find the Λ ($\Lambda > K$) trajectories with the highest UBs.
2. Fetch the $(\Lambda - 1)$ trajectories from the cells and compute their full matching to Q using $FullM(Q, A_i)$.
3. If the Λth UB is smaller or equal to the Kth largest full match then stop; else goto step 1 with $\Lambda{=}\Lambda + \lambda$.

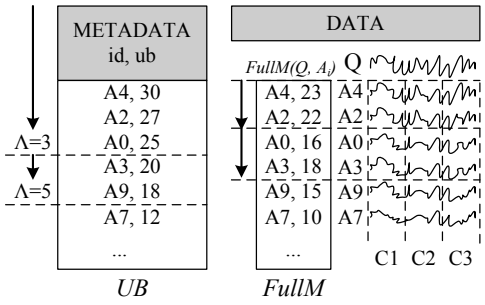

Figure 4: Example execution of *UB-K*.

query processing algorithm, such as *TPUT* [8], *TJA* [28] and *TPAT* [14], that transfers only the most necessary entries from the *METADATA* table towards QN. These algorithms do the following function: Given a set of n distributed scores for each of m objects, they return the K objects with the highest score across all n sites. Note that in our setting, these scores are the local upper bounds. In the experimental evaluation we utilize the TJA algorithm, although any other distributed top-k algorithm is applicable as well.

Description: Algorithm 1 presents *UB-K*. In the first step of the algorithm, QN retrieves the Λ highest UBs with the assistance of a distributed top-K algorithm. The parameter Λ expresses the user confidence in the *METADATA* bounds. For example when the user is confident that the *METADATA* table contains tight bounds, then Λ might be set to a small value.[3] In the second step, QN fetches the exact trajectory (i.e. *DATA*) for $\Lambda - 1$ trajectories A_i identified in the first step. It then performs a local computation of $FullM(Q, A_i)$ and determines their full matching to the query Q. If the Λth highest UB is smaller or equal to the Kth highest full matching value, we terminate with the final result. Otherwise, we perform another iteration by increasing the parameter Λ by λ.

Example: Consider the example scenario of Figure 4. Assume that the query is to find the top-2 trajectories ($K{=}2$) across $C1, C2$ and $C3$. In the first step, QN computes the UBs of the highest Λ *METADATA* entries (i.e. A_4, A_2, A_0) using a distributed top-k algorithm. These entries are the ones that have the highest average UB across the three cells. The entries below A_0 in the *METADATA* table are not available to the querying node QN at this point. In the

[3]In this paper we initialize Λ as $K + 1$.

second step QN will fetch the subsequences of the trajectories identified in the first step. Therefore QN has now the complete trajectories for A_4 and A_2 (right side of Figure 4). QN then computes the following full matching: $FullM(Q, A_4) = 23$, $FullM(Q, A_2) = 22$ using the Longest Common Subsequence described in Section 5. Since the Λth highest UB ($A_0 = 25$) is larger than the Kth highest full match ($A_2 = 22$), the termination condition is not satisfied in the third step. To explain this, consider a trajectory X with a UB of 24 and a full match of 23. Obviously X is not retrieved yet (because it has a smaller UB than 25). However, it is a stronger candidate for the top-2 result than $(A_2, 22)$, as X has a full match of 23 which is larger than 22. Therefore we initiate the second iteration of the UB-K algorithm in which we compute the next λ ($\lambda = 2$) *METADATA* entries and full values $FullM(Q, A_0) = 16$, $FullM(Q, A_3) = 18$. Now the termination has been satisfied because the Λth highest UB $(A_9, 18)$ is smaller than the Kth highest full match $(A_2, 22)$. Finally we return as the top-2 answer the trajectories with the highest full matches (i.e. $\{(A_4, 23), (A_2, 22)\}$).

Theorem 1. *The UB-K algorithm always returns the most similar objects to the query trajectory Q.*

Proof: Let A denote some arbitrary object returned as an answer by the UB-K algorithm ($A \in Result$), and B some arbitrary object that is not among the returned results ($B \notin Result$). We want to show that $FullM(Q, B) \leq FullM(Q, A)$ always holds.

Assume that $FullM(Q, B) > FullM(Q, A)$. We will show that such an assumption leads to a contradiction. Since $A \in Result$ and $B \notin Result$ it follows from the first step of the algorithm that $ub_B \leq ub_A$. In the second phase of the algorithm we fetch the trajectory A and calculate $FullM(Q, A)$. By using the assumption, we can now draw the following conclusion: $FullM(Q, A) < FullM(Q, B) \leq ub_B \leq ub_A$. When the algorithm terminates in the third step, with A among its answers, we know that ub_X, for some object X, was smaller or equal to the Kth largest full score (i.e. $ub_X \leq ... \leq FullM(Q, A)$). But it is also true that $ub_B \leq ub_X$ (as object B was not chosen in the first step of the algorithm), which yields $ub_B \leq ub_X \leq FullM(Q, A)$ and subsequently $FullM(Q, B) \leq FullM(Q, A)$ (by definition $FullM(Q, B) \leq ub_B$). This is a contradiction as we assumed that $FullM(Q, B) > FullM(Q, A)$ □

4.2 The UBLB-K Algorithm

The *UBLB-K* algorithm is, similarly to *UB-K*, an iterative algorithm for retrieving the K most similar trajectories. However it has two subtle differences: (i) It uses both an upper bound (UB) and a lower bound (LB) in order to determine whether the top K trajectories have been found and (ii) It transfers the candidate trajectories in a final bulk step rather than incrementally.

Description: Algorithm 2 presents *UBLB-K*. The first step of this algorithm is identical to *UB-K* with the difference that we also compute a distributed LB. This comes at a very small network and delay overhead as this is performed in parallel with the UB computation. In the second step, QN checks if the Λth highest UB is smaller or equal to the Kth highest LB. If that is the case then QN certainly

Algorithm 2 : UBLB-K Algorithm

Input: Query Q, m Distributed Trajectories, Result Parameter K, Iteration Step λ.
Output: K trajectories with the highest match to Q.

1. Run any distributed top-K algorithm for Q and find the Λ ($\Lambda > K$) trajectories with the highest UB. For each UB also retrieve the respective LB.
2. If the Λth highest UB is smaller or equal to the Kth highest LB then goto step 3; else goto step 1 with $\Lambda = \Lambda + \lambda$.
3. Fetch the trajectory for objects which have a UB bigger than the Kth highest LB.

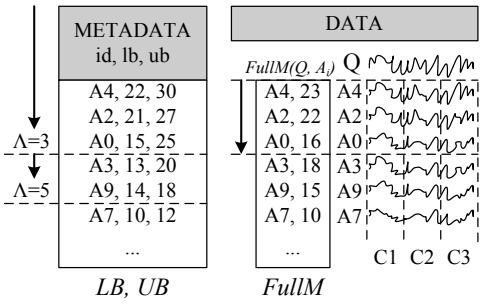

Figure 5: Example execution of *UBLB-K*.

knows that the top-K trajectories are in the candidate list constructed in the first step. Therefore QN proceeds to step 3 in which it fetches all trajectories with UB larger than the Kth highest LB. Notice that if the Λth highest UB is larger than the Kth highest LB, then we would proceed to step 1 for another iteration with a larger parameter Λ. The intuition behind *UBLB-K* is that the termination condition can be identified based on the LB rather than the full matching. Therefore we are not required to incrementally fetch the distributed trajectories, but we can fetch them all together in a final bulk step.

Example: Consider the example of Figure 5, which this time presents both bounds in *METADATA*. In the first step we compute the LB and UB for the first Λ *METADATA* entries (i.e. A_4, A_2, A_0). We observe that the Λth highest UB $(A_0, 25)$, is larger than the Kth highest LB $(A_2, 21)$. Therefore the termination condition of step 2 is not satisfied. The intuition behind this condition is in fact the same as for the UB-K case. Specifically, the existence of some trajectory X with a UB 24 and a full matching of 23 would create a stronger candidate for the top-2 result than the current candidate $(A_2, 21, 27)$. Therefore we initiate the second iteration in which we compute the next λ ($\lambda = 2$) *METADATA* entries A_3, A_9. Now the Λth highest UB (18) is smaller than the Kth highest LB (21). Therefore QN can safely proceed to the final bulk transfer step knowing that the result is among the candidates. Instead of fetching all candidates however, QN only fetches the trajectories with UB larger than the Kth highest LB which is 21. Therefore QN fetches $(A_4, 22, 30), (A_2, 21, 27)$ and $(A_0, 15, 25)$, since these are the only candidates with UB larger than Kth highest

LB $(A_2, 21)$. After QN performs the final bulk transferring step it calculates the full match of the retrieved candidates and simply returns the top-2 trajectories with the highest match to the query (i.e. $\{(A_4, 23), (A_2, 22)\}$).

Theorem 2. *The UBLB-K algorithm always returns the most similar objects to the query trajectory Q.*

Proof: Similar to Theorem 1 □

4.3 Discussion

UB-K vs. UBLB-K: Comparing the two algorithms we can observe that in many cases (like our example), *UBLB-K* might terminate and retrieve less *DATA* entries at the expense of an increased overhead of *METADATA* entries. Note that the *DATA* entries are orders of magnitudes more expensive to be transferred than *METADATA* entries. The savings increase when the LBs are tighter, which consequently allows QN to determine faster whether the top-K results have been found. The savings of *UBLB-K* are also increased for larger values of Λ. Note that *UB-K* has to always retrieve Λ full trajectories while *UBLB-K*, based on the LBs, can be more selective. These observations are validated in Section 6.

Incremental Deepening into Top-K Results: Since both our algorithms fetch the highest *METADATA* incrementally (e.g. they find the top K, $\Lambda+\lambda$, $\Lambda+2\lambda$, ... UBs at increasing iterations), QN can cache the *METADATA* and *DATA* it has received in the previous iterations and only request for the new *METADATA* and *DATA* in a new iteration. Consider for example Figure 4, where in the first iteration, QN fetches the trajectories of {A4, A2}. In the second iteration, QN only needs to fetch the trajectories of A0 and A3, since the top 2 trajectories have already been fetched in the previous iteration.

Global Clock Independence: It is important to mention that our algorithms operate correctly in the absence of a global clock. This is true because the various phases of our algorithms are not defined as a function of time. However, when nodes are not synchronized then this might result in the computation of incorrect answers to the respective queries. We emphasize that this is not attributed to the operation of our algorithms but rather to the out-of-order trajectories. In fact even the centralized algorithm would be affected by the same problems in this case.

5. SIMILARITY MEASURES FOR SPATIO-TEMPORAL TRAJECTORIES

In the previous section we have discussed how our proposed distributed query processing algorithms work by utilizing locally computed lower and upper bound scores on the matching between a query Q and the respective trajectories. In this section we describe how these bounds are calculated. We start out by providing an overview of distance measures that were proposed in a centralized setting, where the querying node has access to the complete trajectory of some moving object. We then provide extensions for computing these distances in a distributed setting. In particular, we will focus on a distributed version of the Longest Common Subsequence, which is utilized in this work.

5.1 Centralized Similarity Measures

Let $A((a_{x:1,y:1}), ..., (a_{x:l_1,y:l_1}))$ and $B((b_{x:1,y:1}), ..., (b_{x:l_2,y:l_2}))$ denote two 2-dimensional trajectories with sizes l_1 and l_2 respectively. The most straightforward way to compute the similarity between A and B is to use any of the L_p-*Norm* distances, such as the *Manhattan* (L_1), *Euclidean* (L_2) *or Chebyshev* (L_∞). Although this family of distances can be calculated very efficiently, it is not flexible to out-of-phase matches and not tolerant to noisy data because the points are only matched at identical time positions.

The *Dynamic Time Warping (DTW) [5]*, solves some of the matching inefficiencies associated with the L_p-*Norm* distances by allowing local stretching of the sequences to optimize the matching. However its performance might deteriorate in the presence of noisy data in which outliers distort the true distance between sequences.

The *Longest Common Sub-Sequence (LCSS)* similarity has been extensively used in many 1-dimensional sequence problems such as string matching. The 2-dimensional adaptation of LCSS using the L_∞[4] is defined as following:

Definition *Given integers δ and ϵ, the Longest Common Sub-Sequence similarity $LCSS_{\delta,\epsilon}(A, B)$ between two sequences A and B is defined as:*

$LCSS_{\delta,\epsilon}(A, B) =$

$$\begin{cases} \mathbf{0}, & \text{if A or B is empty} \\ \mathbf{1 + LCSS_{\delta,\epsilon}(Tail(A), Tail(B))} & \\ \quad \text{if } |a_{x:l_1} - b_{x:l_2}| < \epsilon \text{ and} & \\ \quad\quad |a_{y:l_1} - b_{y:l_2}| < \epsilon \text{ and } |l_1 - l_2| < \delta & \\ \mathbf{max(LCSS_{\delta,\epsilon}(Tail(A), B), LCSS_{\delta,\epsilon}(A, Tail(B)))} & \\ \quad \text{otherwise} & \end{cases}$$

where the δ and ϵ are user defined thresholds that allow flexible matching in the time and the space domain respectively. LCSS can deal more efficiently with outliers, because outliers are simply dropped from the matching and so large outliers do not skew the measure. Similar to DTW, LCSS can be computed by a dynamic programming algorithm with a time complexity of $O(\delta \cdot (l_1 + l_2))$[9].

Even though LCSS offers many desirable properties, its time complexity of $O(\delta \cdot (l_1 + l_2))$ might constitute it inefficient for large values of l_1, l_2 or δ, so it is desirable to give a technique to upper bound the $LCSS$ similarity. The idea of the technique proposed in [24] is to encapsulate the query trajectory Q within a bounding envelope and then find the intersection between the envelope and the trajectories. For simplicity consider the 1-dimensional case where Q= $(q_{x:1}, \ldots, q_{x:l_1})$ denotes a query and A= $(a_{x:1}, \ldots, a_{x:l_2})$ a trajectory. Suppose that we replicate each point Q_i for δ time instances before and after time i and that we also replicate each point Q_i for ϵ space instances above and below Q_i (see Figure 6). The area contained in the union of all these points defines the *Minimum Bounding Envelope (MBE)* of the query trajectory Q. The notion of the bounding envelope can be trivially extended to more dimensions.

The LCSS similarity between the envelope of Q and a sequence A is defined as:

$$LCSS(MBE_Q, A) = \sum_{i=1}^{n} \begin{cases} 1 & \text{if A[i] within envelope} \\ 0 & \text{otherwise} \end{cases}$$

[4]We could also use L_1 or L_2 for the recursion step

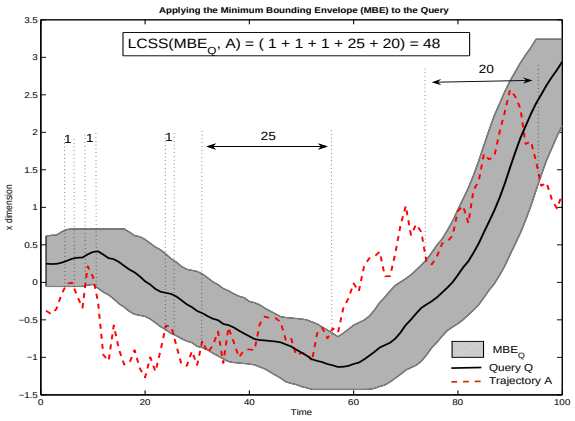

Figure 6: To upper bound $LCSS_{\delta,\epsilon}(A,Q)$, we use the envelope MBE of Q

Note that the LCSS similarity of an envelope and a sequence does not depend on ϵ and δ. These have been incorporated in the building of the envelope. It can be computed in $O(min(l_1,l_2))$. For example, in Figure 6 the $LCSS$ similarity between MBE_Q and sequence A is 48 because this represents the total intersection length. This value represents an *upper bound* on $LCSS_{\delta,\epsilon}(A,Q)$. For any two trajectories Q and A, $LCSS_{\delta,\epsilon}(A,Q) \leq LCSS(MBE_Q,A)$.

5.2 Distributed Similarity Measures

The similarity measures presented in the previous section are only applicable to the centralized scenario, where all the segments of a target trajectory are stored locally at the QN. In this section we show how to compute the Longest Common Subsequence $LCSS_{\delta,\epsilon}(Q,A_i)$ in a distributed setting. In particular we present techniques to upper bound ($UpperM$) and lower bound ($LowerM$) the LCSS matching. These bounds can then be exploited, using the UB-K and UBLB-K algorithms described in the previous section, in order to find the K most similar trajectories in a completely distributed fashion.

Recall that in a distributed setting each trajectory A_i ($i \leq m$) is spatially fragmented over n cells. We define as A_{ij}, the segment of the trajectory A_i ($i \leq m$) that lies inside cell c_j ($j \leq n$). We note that A_{ij} may not be continuous, however this does not present a problem since each point in the trajectory is uniquely identified. Therefore each local subsequence can still be matched over the query Q, which is assumed to be available in its entirety to each cell.

The basic idea of our approach is to perform local computations of partial lower and upper bounds at each cell and then combine these partial results to give upper and lower bounds for $LCSS_{\delta,\epsilon}(Q,A_i)$ ($\forall i \leq m$). This allows us to perform the computation in parallel and to minimize the amount of data transferred to QN.

Note that the only other alternatives are to either transfer all A_{ij} ($\forall i \leq m, \forall j \leq n$) to QN (the *Centralized* solution) or to perform the dynamic programming computation of $LCSS_{\delta,\epsilon}(Q,A_i)$ ($\forall i \leq m$) in a distributed setting. The latter approach is lengthy and expensive, as it requires the communication of the execution state between neighboring cells for each pair (Q,A_i) ($\forall i \leq m$).

Algorithm 3 : DUB_LCSS Algorithm

Input: n distributed cells, m trajectories per cell (each denoted as A_{ij} ($i \leq m$, $j \leq n$)), query trajectory Q.
Output: $DUB\_LCSS_{\delta,\epsilon}(MBE_Q,A_i)$, $\forall i \leq m$

1. Each cell c_j ($j \leq n$) uses $LCSS(MBE_Q,A_{ij})$ to calculate the similarity of each local subsequence trajectory A_{ij} to MBE_Q.
2. The upper bound $DUB\_LCSS_{\delta,\epsilon}(MBE_Q,A_i)$ ($i \leq m$), is constructed by adding the n local results (i.e. $\sum_{j=1}^{n} LCSS(MBE_Q,A_{ij})$).

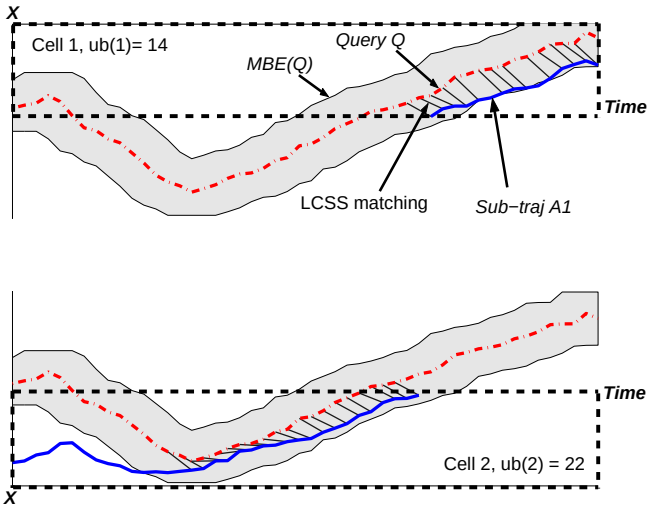

Figure 7: The local upper bound computations of the projections of cell 1 (top) and cell 2 (bottom) in the X dimension. The overlap of the sub-trajectory in the cell with the MBE_Q is taken as upper bound matching.

5.3 Distributed LCSS Upper Bound (DUB_LCSS)

Algorithm 3 presents our distributed upper bound algorithm on LCSS (DUB_LCSS). The idea is to have each cell c_j ($\forall j \leq n$) locally match its local subsequences A_{ij} ($\forall i \leq m$) to Q using the upper bounding method $LCSS(MBE_Q,A_{ij})$ presented in section 5.1. Note that this is a simple and cheap operation since each trajectory point in the local subsequence A_{ij} is associated with a timestamp. We then simply perform a parallel addition of these individual results which yields an upper bound on the LCSS matching. Figure 7 illustrates the operations of the algorithm. The correctness of DUB_LCSS is established by Theorem 3.

Theorem 3. For any query trajectory Q and any distributed trajectory A_i the following holds:
$LCSS_{\delta,\epsilon}(Q,A_i) \leq DUB\_LCSS(MBE_Q,A_i)$.

Proof: By construction, the aggregate similarity for a trajectory A_i is computed by adding the local similarity computation in each of the n cells: $\sum_{j=1}^{n} LCSS(MBE_Q,A_{ij})$. If a trajectory point (x,y) is in $LCSS_{\delta,\epsilon}(Q,A_i)$, then this point must be within δ and ϵ from the query Q. The trajectory points returned by $LCSS(MBE_Q,A_{ij})$ are all the points in

Algorithm 4 : DLB_LCSS Algorithm

Input: n distributed cells, m trajectories per cell (each denoted as A_{ij} $(i \leq m,\ j \leq n)$), query trajectory Q.
Output: $DLB\_LCSS_{\delta,\epsilon}(Q, A_i), \forall i \leq m$

1. For each trajectory A_i, cell c_j $(j \leq n)$ finds the time region $T_{ij} = \{ts(p) | p \in A_{ij}\}$ when A_i stays in cell c_j. Filter Q into Q'_{ij} such that Q'_{ij} is in the same time intervals as A_{ij}, $Q'_{ij} = \{p | p \in Q \ and \ ts(p) \in T_{ij}\}$.

2. Each cell c_j $(j \leq n)$ performs a local computation of $LCSS_{\delta,\epsilon}(Q'_{ij}, A_{ij})$ $(\forall i \leq m, \forall j \leq n)$.

3. The lower bound $DLB\_LCSS_{\delta,\epsilon}(Q, A_i)$ $(\forall i \leq m)$, is constructed by adding the n local results (i.e. $\sum_{j=1}^{n} LCSS_{\delta,\epsilon}(Q'_{ij}, A_{ij})$).

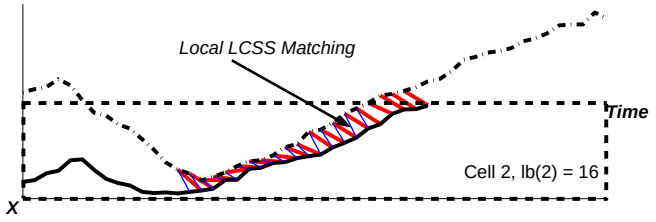

Figure 8: The local lower bound lb(Q,A) in the projection of cell 2 in the X dimension. The thick red line represents full LCSS matching, while the thin blue line represents the lower bound matching within the cell.

A_{ij} that are within δ and ϵ from the query Q. It follows that $\sum_{j=1}^{n} LCSS(MBE_Q, A_{ij}) \geq LCSS_{\delta,\epsilon}(Q, A_i)$ which reduces to $LCSS_{\delta,\epsilon}(Q, A_i) \leq DUB\_LCSS(MBE_Q, A_i)$ □

5.4 Distributed LCSS Lower Bound (DLB_LCSS)

Algorithm 4 presents our distributed lower bound algorithm on $LCSS$. The idea is again to perform n distributed computations with a local similarity function, and then perform a parallel addition of these individual results. Our lower bound is computed by having each cell c_j $(j \leq n)$, to perform a local computation of $LCSS_{\delta,\epsilon}(Q'_{ij}, A_{ij})$ $(\forall i \leq m, \forall j \leq n)$, without extending the warping window δ outside A_{ij}. The correctness of $DLB\_LCSS(Q, A_i)$ is established by Theorem 4. Figure 8 illustrates the computation of the local lower bound.

Theorem 4. For any query trajectory Q and any distributed trajectory A_i the following holds:
$DLB\_LCSS(Q, A_i) \leq LCSS_{\delta,\epsilon}(Q, A_i)$.

Proof: According to [9], a dynamic programming algorithm can be used to compute the LCSS similarity of two trajectories by constructing a matching matrix. Consider Figure 9, where we illustrate a matching matrix for two trajectories Q and A. Suppose we set the parameter δ as 2.5 and ϵ as 0.5, then the matching matrix M[i][j] is marked with X as local matching of Q[i] and A[j], and O as non-matching. Notice that the matching is only computed and considered within a δ-diagonal zone (the gray blocks) as we constraint the matching window size. The common subsequences of Q and A can be found by counting matches in the matrix.

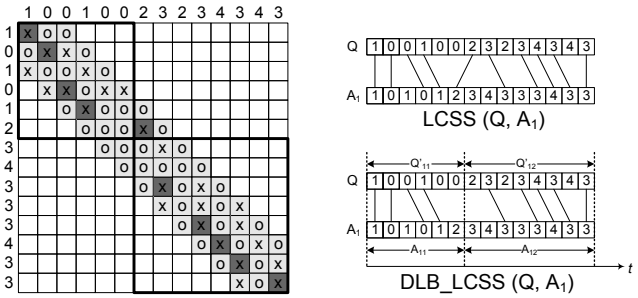

Figure 9: Lower Bounding LCSS: In LCSS, the matching is only possible within the matching window (the diagonal line which is light-gray shaded). The LCSS matching can be lower bounded by the sum of the dark-gray shaded cells in the two respective sub-matrix (6X6 and 8X8).

The algorithm will start from the right bottom block of the matrix $M[\ell][\ell]$ (ℓ is the length of the two trajectories), jump to the left-up neighbor when the current block is matching, and jump to the left or the up neighbor when the current block is not matching, until we reach the left-top block $M[1][1]$. The LCSS(Q, A) is then the largest number of matching blocks (dark-gray blocks) we get. Algorithm 4 divides the query Q into non-overlapping segments Q'_{ij} in the same manner as A_i is divided into A_{ij}. In the matching matrix, these segments can be represented by the sub-matrices in the diagonal line (as the sub-matrices in bold frame in Figure 9). As shown in Figure 9, any matching in the sub-matrices is also a valid matching in the global matrix. Since the sub-matrices do not overlap with each other, the LCSS computation of the $l \times l$ matrix can be lower bounded by the summation of LCSSs of non-overlapping sub-matrices at diagonal line.

Therefore $\sum_{j=1}^{n} LCSS(Q'_{ij}, A_{ij}) \leq LCSS_{\delta,\epsilon}(Q, A_i)$, thus $DLB\_LCSS\ (Q, A_i) \leq LCSS_{\delta,\epsilon}(Q, A_i)$□

6. EXPERIMENTAL EVALUATION

In this section we present an extensive experimental evaluation of the three similarity search methods: (i) *Centralized*, (ii) *UB-K* and (iii) *UBLB-K*. We have implemented a trace driven simulator in GNU C++ which takes as an input a spatio-temporal dataset and then splits it into n equi-width cells. Our dataset and methodology are described below.

Dataset: For our evaluation we use the *OLDENBURG* dataset, which includes 25,000 trajectories generated over the Oldenburg street map, using the *Network-based Generator of Moving Objects* [6]. The spatial universe of this dataset is 23,500 x 23,500 points and each trajectory has a length of 500 temporal points. The dataset has an overall size of 200MB (each spatio-temporal record in this paper is 16 bytes). Our queries for the above dataset are synthetically derived with the addition of interpolated peaks of Gaussian noise. This created variations in the pattern of these trajectories. Our results are averages over 10 queries.

Methodology: Our performance measures are: (i) *Bytes*, (ii) *Time* and (iii) *Messages* required for finding the K most similar trajectories to Q. Our communication protocol is

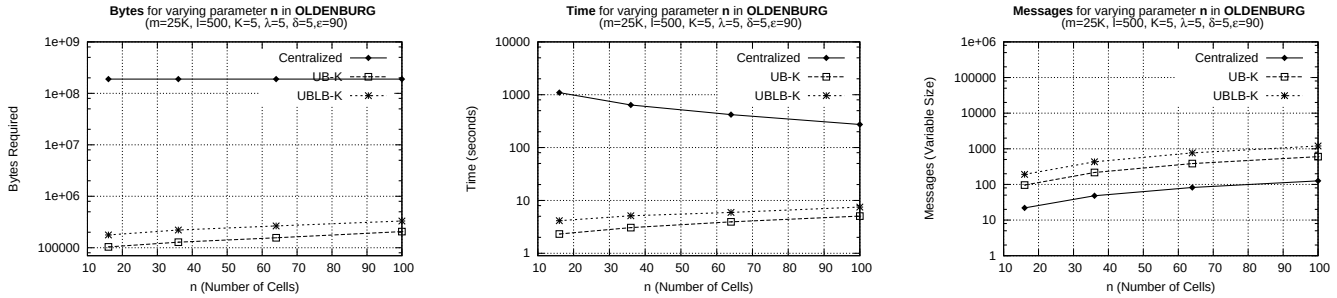

Figure 10: Performance Evaluation, varying n (# cells) for the OLDENBURG dataset.

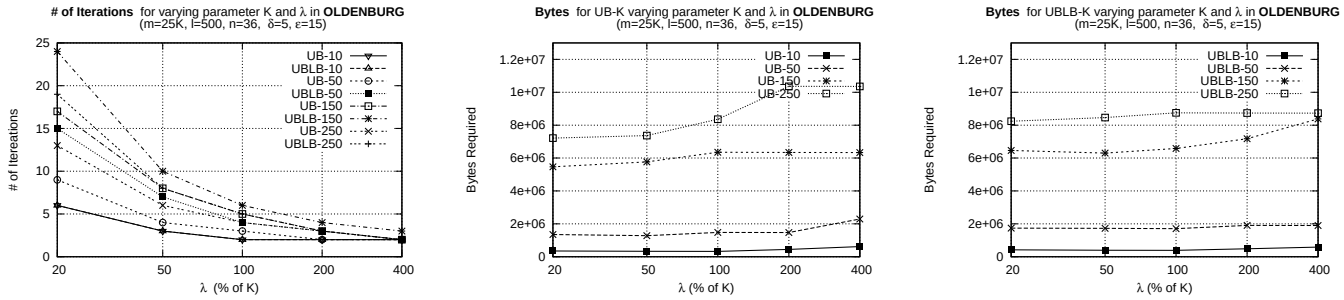

Figure 11: Varying the Parameter K and λ using the OLDENBURG dataset.

structured in the following way: Each message is associated with a 40 byte header. This is augmented with an additional 6 byte application layer header that includes: (i) The Cell identifier (1 byte), (ii) the Message size (4 bytes) and the depth of a cell from the querying node QN (1 byte). We measure the total communication cost *Bytes* as a summation of *message cost* (40 bytes header cost per message) and the *data cost* (i.e. the cost of communicating $DATA$ and $METADATA$ between QN and all the cells). In order to allow reproducible and comparable results we perform our experiments on a single host using a simulated dedicated bandwidth of 128KBps for the edges between QN and the cells. Our edges have a random latency between 0-100ms which is typical for Internet environments. Since the cells can transmit their results back to QN in parallel, we consider only the longest time of each round.

6.1 Performance Evaluation

Figure 10 illustrates the three performance parameters as a function of the cell numbers for the three alternatives (*'Centralized', 'UB-K' and 'UBLB-K'*) using the OLDENBURG dataset. In all cases *UB-K* and *UBLB-K* return the same result with *Centralized*. However, *UB-K* and *UBLB-K* require two to three orders of magnitudes less bytes and are two orders of magnitudes faster than *Centralized*.

We also noticed that *UB-K* performs in less iterations than *UBLB-K*. Specifically, we found that *UB-K* required 1-3 iterations while *UBLB-K* required 2-6 iterations. This was attributed to the conservative parameter λ (studied in Section 6.2).

For the Messages graph, we observe that *UB-K* and *UBLB-K* require more messages than *Centralized*. This is because the *UB-K* and *UBLB-K* algorithms include multiple rounds of $METADATA$ and partial $DATA$ communications between QN and each cell, while the centralized method just performs one round of full $DATA$ transmission. It is important to notice that these $METADATA$ messages are small in size and therefore are cheaper to be transferred, in terms of bytes, than transferring $DATA$.

6.2 Varying the Parameters K and λ

In our UB-K and UBLB-K algorithms, λ is the number of top-K $METADATA$ to fetch in each iteration. In this experimental series we study the impact of the parameter λ to the minimization of the bandwidth cost between the query node and cells. As shown in Figure 11a, when λ increases, the number of iterations in UB-K and UBLB-K decreases.

This is due to the fact that a larger step length λ, yields more candidates in each round and therefore the iteration stop condition is satisfied much earlier. A consequence of increasing λ is also the fact that the message cost is significantly reduced (although this is not directly shown in the figures). This is explained by the fact that a small λ yields many small-sized packet transmissions from the cells to QN, while a larger λ yields a few larger-sized packet transmissions.

The downside of increasing λ, is that the *Bytes* parameter (data and message cost), now increases for both UB-K and UBLB-K as shown in figures 11b and 11c respectively. This is attributed to the fact that a larger λ, will yield more $METADATA$ and $DATA$ candidate transfers towards QN. Figure 11 shows that, for a small K (e.g. ≤ 50), the *Bytes* parameter remains low with λ set to 50%K $\sim$ 100%K. If K is large (e.g. ≥ 150), the parameter λ should be kept small as the data cost is so dominant that the message cost is negligible.

7. CONCLUSION

This paper introduces and formalizes the *distributed trajectory similarity search* problem. We propose two novel distributed query processing algorithms that provide an efficient and exact solution to this problem. Our algorithms exploit a partial lower and upper bounds on the LCSS similarity, which is computed locally by each node. Comprehensive experiments with realistic data shows that *UB-K* and *UBLB-K* are orders of magnitudes more efficient in terms of network traffic and delay. Our approach can easily be extended to lower bound the DTW distance as well. Since DTW is a distance, our distributed retrieval techniques would have to find the K trajectories with the smallest distances to the query. To do that we can modify our *UB-K* algorithm to work with lower bounds instead.

Acknowledgements

We would like to thank Michalis Vlachos (IBM TJ Watson) for his help in developing the spatiotemporal similarity measures. We would also like to thank Eamonn Keogh (UCR) for the interesting discussions and ideas. This work was supported by grants from NSF ITR #0220148, #0330481.

8. REFERENCES

[1] Al-Taha K. , Snodgrass R., and Soo M., "Bibliography on Spatiotemporal Databases", In SIGMOD Record, 22(1):59–67, March 1993.

[2] Babcock B. and Olston C., "Distributed Top-K Monitoring", In *SIGMOD'03*, San Diego, CA, USA, pp. 28-39, 2003.

[3] Balke W.-T., Nejdl W., Siberski W., Thaden U., "Progressive Distributed Top-K Retrieval in Peer-to-Peer Networks", In *ICDE'05*, Tokyo, Japan, pp. 174-185, 2005.

[4] Basch J., Guibas L. and Hershberger J., "Data Structures for Mobile Data", In *SIAM Symposium on Discrete Algorithms*, New Orleans, Louisiana, pp. 747-756, 1997.

[5] Berndt D., Clifford J., "Using Dynamic Time Warping to Find Patterns in Time Series", In *KDD'94*, Menlo Park, CA, pp. 229-248, 1994.

[6] Brinkhoff T., "A Framework for Generating Network-Based Moving Objects". In *GeoInformatica*, 6(2), 2002.

[7] Bruno N., Gravano L. and Marian A., "Evaluating Top-K Queries Over Web Accessible Databases", In *ICDE'02*, San Jose, CA, Page 369, 2002.

[8] Cao P. and Wang Z., "Efficient Top-K Query Calculation in Distributed Networks", In *PODC'04*, St. John's, Newfoundland, Canada, pp. 206-215, 2004.

[9] Das G., Gunopulos D., Mannila H., "Finding Similar Time Series", In *PKDD'97*, Trondheim, Norway, pp. 88-100, LNCS 1263, 1997.

[10] Fagin R., "Combining Fuzzy Information from Multiple Systems", In *PODS'96*, Montreal, Quebec, pp. 216-226, 1996.

[11] Fagin R., Lotem A. and Naor M., "Optimal Aggregation Algorithms For Middleware", In *PODS'01*, Santa Barbara, CA, pp. 102-113, 2001.

[12] Hadjieleftheriou M., Kollios G., Bakalov P., Tsotras V.J., "Complex Spatio-Temporal Pattern Queries", In *VLDB'05*, Trondheim, Norway, pp. 877-888, 2005.

[13] Kollios G., Gunopulos D., Tsotras V.J., "On Indexing Mobile Objects", In *PODS'99*, Philadelphia, PA, pp. 261-272, 1999.

[14] Yu H., Li H., Wu P., Agrawal D., Abbadi A.E., "Efficient Processing of Distributed Top-k Queries", In *DEXA'05*, Krakow, Poland, pp. 65-74, 2005.

[15] Kollios G., Gunopulos D., Tsotras V.J., Delis A., Hadjieleftheriou M., "Indexing Animated Objects Using Spatiotemporal Access Methods", In *IEEE TKDE*, Vol. 13, Iss. 5, pp. 758-777, 2001.

[16] Madden S.R., Franklin M.J., Hellerstein J.M., Hong W., "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", In *OSDI'02*, Boston, MA, pp. 131-146, 2002.

[17] Bruno N., Gravano L. and Marian A., "Evaluating Top-K Queries Over Web Accessible Databases", In *ICDE'02*, San Jose, CA, USA, Page 369, 2002.

[18] Mokbel M., Xiong X. and Aref W.G., "SINA: Scalable Incremental Processing of Continuous Queries in Spatiotemporal Databases", In *SIGMOD'04*, Paris, France, pp. 623-634, 2004.

[19] Papadias D., Mamoulis N., Delis V., "Approximate Spatio-Temporal Retrieval", In *ACM TOIS*, Vol. 19, Iss. 1, pp. 53-96, 2001.

[20] Saltenis S., Jensen C.S., "Indexing of Moving objects for Location-Based Services", In *ICDE'02*, Washington, DC, Page 463, 2002.

[21] Saltenis S., Jensen C.S., Leutenegger S.T., Lopez M.A., "Indexing the positions of continuously moving objects", In *SIGMOD'00*, Dallas, Texas, pp. 331-342, 2000.

[22] Tao Y., Papadias D., "The MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries", In *VLDB'01*, pp. 431-440, 2001.

[23] Tao Y., Sun J. and Papadias D., "Analysis of Predictive Spatio-Temporal Queries" In *ACM TODS*, Vol. 28 , Iss. 4, pp. 295-336, 2003.

[24] Vlachos M., Hadjieleftheriou M., Gunopulos D., Keogh E., "Indexing multi-dimensional time-series with support for multiple distance measures" In *ACM SIGKDD'03*, Washington, D.C., pp. 216-225, 2003.

[25] Xiong X., Mokbel M.F., Aref W.G., "SEA-CNN: Scalable Processing of Continuous K-NN Queries in Spatio-temporal Databases", In *ICDE'05*, Tokyo, Japan, pp. 643-654, 2005.

[26] Xia T., Zhang D., Kanoulas E., Du Y., "On Computing Top-t Most Influential Spatial Sites", In *VLDB'05*, Trondheim, Norway, pp. 946-957, 2005.

[27] Xiong L. , Chitti S., Liu L., "Top-k Queries across Multiple Private Databases", In *ICDCS'05*, Columbus, Ohio, pp. 145-154, 2005

[28] Zeinalipour-Yazti D., Vagena Z., Gunopulos D., Kalogeraki V., Tsotras V., Vlachos M., Koudas N., Srivastava D., "The Threshold Join Algorithm for Top-K Queries in Distributed Sensor Networks", In *DMSN* (collocated with VLDB'05), Trondheim, Norway, 2005.