BACR: Set Similarities with Lower Bounds and Application to Spatial Trajectories

Martin Werner
Mobile and Distributed Systems Group
Ludwig-Maximilians University Munich, Munich, Germany
martin.werner@ifi.lmu.de

ABSTRACT

This paper proposes a length-independent feature representation of sets of strings based on Bloom filters called BACR for similarity search in databases. Further, we show how a Z-curve-based discretization of geospatial trajectories can be used in order to search for similar trajectories in large databases. Additionally to the already-known estimation of the size of the union and the intersection of sets from Bloom filters, we propose a way to calculate an upper bound for the intersection and a lower bound for the union of sets. Consequently, we show that the Jaccard distance and many other similarity measures allow for a lower bound. This makes exact similarity search on large databases of this type feasible. Finally, we show that the Jaccard distance is incompatible with the union of sets and replace the Jaccard distance appropriately in a way such that even collections of sets of strings can be represented with a single BACR feature vector at least for similarity search applications. The algorithms are thoroughly evaluated and motivated by real-world examples.

Categories and Subject Descriptors

H.2.8 [**Database**]: Database Applications—*Spatial databases and GIS*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing

General Terms

Algorithm, Spatial Computing, Time Series, Trajectory Analysis

Keywords

Trajectory, Moving Objects, Multi-modal Trajectory, Big Data

1. INTRODUCTION

Multi-dimensional and multi-modal time series experience a more and more central role in the context of "big data".

SIGSPATIAL '15, November 03-06, 2015, Bellevue, WA, USA
Copyright 2015 ACM . ISBN 978-1-4503-3967-4/15/11
DOI: http://dx.doi.org/10.1145/2820783.2820802 ...$15.00.

Driven by the wide distribution of mobile devices equipped with GPS sensors and by infrastructures providing similar location information inside buildings, large collections of multi-modal time series data are collected and generated. These datasets typically include GPS readings, phone call data (e.g., call detail records), measurements in industry, multimedia and social media data, web page data, and much more.

While the treatment of pure spatial trajectories has been studied well in the last decades [13, 5, 7, 16], the multi-modal nature of trajectories – namely that they consist of both spatial and non-spatial data in many applications – generates a two-stage system, which runs into problems in general. There exist proposals to index the keyword space (e.g., everything except spatial coordinates) as a first level index and index the spatial domain relative to that keyword effectively building an inverted index mapping keywords to spatial indexing trees, see [15] and references therein. Alternatively, one could try to generate a spatial index structure in the spatial domain and use another technique in a second step to also evaluate a query with respect to the keyword space. Still, it is difficult for these indices (or even impossible) to compress data in order to allow for querying very large datasets.

This paper, however, proposes to relax the spatial domain into a suitable discrete representation as sets leading to a situation in which spatial similarity as well as similarity in other modalities can be evaluated at the same time.

A widely used approach for solving this class of problems relies on random projections and locality-sensitive hashing (LSH) as introduced by Gionis et al. in 1999 [6]. Locality-sensitive hashing schemes are known to exist in many situations, however, often only for approximate results. For example, Broder et al. introduced Min-Wise Independent Permutations for the Jaccard similarity of sets [3]. In this context, a set $h \in H$ of hash functions is selected and it can be shown that

$$\mathrm{Pr}\left(\mathrm{h(A)} = \mathrm{h(B)}\right) \approx \mathrm{sim}_{\mathrm{Jaccard}}(\mathrm{A}, \mathrm{B})$$

In order to decrease the random effects from hashing, this scheme can be iterated with different hash functions leading to feature vectors $(h_1(A), h_2(A), ...)$ and the Jaccard similarity can be measured by counting coincidences between those vectors.

Interestingly, many different set similarity coefficients have been used in practice, [12, 11], however, not all of them support LSH scheme. This is due to a lemma of Charikar, which proves that the existence of LSH for a similarity measure sim(A, B) implies the triangle inequality for $1 - \text{sim}(A, B)$ [4].

For this paper, a *trajectory* shall be any time series consisting of samples which are vectors containing any mixture of spatial coordinates, points in some metric space, or even discrete values. Note that this definition is quite general. It is intended to treat multimodal trajectories containing labels in the same way as pure geometric trajectories.

This paper proposes a scheme in which general trajectories are suitably represented as sets and similarity is given by one of the various set similarity measures, usually based on the intersection, union, and size of sets. Concretely, this paper proposes to represent general trajectories as sets of strings and provides a data structure of constant size based on Bloom filters in order to summarize each trajectory. Possibly the most important novel results of this paper are the following: (1) A derivation of a lower bound for the Jaccard distance of sets represented using Bloom filters, (2) a method for quickly calculating the approximate Jaccard distance of sets represented in this way, (3) a method for performing exact nearest neighbor search on these sets represented as Bloom filters with respect to the Jaccard distance, and (4) a lower bounding method for nearest neighbor search with respect to the Jaccard distance, which is compatible with collections of sets of strings in which collections of sets can be represented by a single feature vector.

While the last aspect is a central idea in many spatial indexing structures for example based on trees, it is unavailable for the Jaccard distance, as there cannot exist a straightforward *mindist* function for unions of sets under the Jaccard distance as will be shown.

The remainder of the paper is structured as follows: In Section 2, related work is shortly reviewed, in Section 3, the Bloom Filter Aggregated Cell Representation (BACR) is introduced, in Section 4, the subset relation is discussed, in Section 5, methods to estimate the Jaccard distance are developed and a lower bound is introduced for this. Section 6 shows how the lower bound can be adapted to the case in which a BACR feature vector models several trajectories at least for similarity search by using the Dice index. Section 7 shows results on synthetic and real datasets and, finally, Section 8 concludes the paper.

2. BACKGROUND AND RELATED WORK

In this section, we first review approaches for indexing geometric trajectories. Then, we explain the Bloom filter data structure. Further, we explain Z-curve codings, a technique based on a space-filling curve with which geometric points can be transformed into strings and with which we discretize geometric or spatial components of the given trajectories.

2.1 Trajectory Indexing

Many different approaches have been discussed in order to index spatial trajectories. For example, spatial index structures for points have been augmented in order to also cover the special time domain of spatial trajectories. Additionally, spatial indices for various timestamps are maintained and the relation between consecutive indices in time is modelled.Thirdly, a regular grid is being used in order to reduce the complexities induced by good indexing structures for points. A good introduction to this area has been presented by Deng et al.[5].

All these approaches lack direct support for non-spatial and multimodal data. However, these are becoming more and more important in reasoning applications, where data analysis shall be applied to the additional, non-spatial attributes of trajectories or trajectory segments.

2.2 Bloom Filter

The Bloom filter is a probabilistic data structure modelling small sets of objects via hashing [1, 2]. Basically, a Bloom filter consists of a hash field F of m binary slots. First, a set $\{h_1 \dots h_k\}$ of k pairwise independent hash functions mapping objects (e.g., strings) to $1 \dots m$ is fixed. An empty filter is represented by an all-zero hash field, inserting an element e is done by setting all slots addressed by at least one of the hash values to one (e.g., $\forall_{i=1\dots k} F[h_i(e)] := 1$). In order to find out whether an element has been inserted into a Bloom filter, one looks at the same slots and returns true, if all slots contain a one (e.g., "$e \in F$", if $\forall_{i=1\dots k} F[h_i(e)] == 1$). The Bloom filter does not have false negatives, however, it can have false positives, if the bits addressed by the hash functions have been set to one by the insertion of other elements. Still, the probability of false-positives is equivalent to the probability of one of the k random hash function addressing a zero slot for a given element, which depends only on the number of elements, the number of hash functions, and the filter size. It is possible to optimally determine the number k of hash functions out of a given number of elements and a given size of a filter and – conversely – it is possible to calculate the probability of a false-positive given the number of elements in the filter, the number of hash functions and the size of the hash field in closed form [1].

2.3 Z-curve codings and the Geohash

The process of mapping coordinates in some d-dimensional space into strings amounts to a mapping of the d-dimensional space onto a one-dimensional discrete space. Space-filling curves such as the Hilbert curve or the Z-curve can be used for this projecting a space cell onto the time in the curve, the Z-curve being very easy to calculate. For the two-dimensional Z-curve, it is known that nearby spatial points tend to be near after Z-curve coding, however, with several locations where the opposite is the case. From an implementation perspective, the Z-curve coding can be generated by splitting the space into square (cubic, ...) cells and bit-interleaving the associated cell indices written in binary. The resulting bit vector can be coded into a string using, e.g., BASE32, which is then widely known as the Geohash [9, 14]. This construction has several properties which are not explicitly used throughout this paper, but which can be used by applications: It is, for example, possible to calculate the neighboring cells given a geohash by some lookups in tables of letters, it is – in general – very fast to encode and decode, and it supports generalization by removing trailing characters: The geohash cell of a prefix of a string is a rectangle containing the geohash cell as well as all other geohash cells of the same prefix. These form a rectangle at anytime.

2.4 Lower Bounds in Similarity Search

One of the most important operations in similarity search is the nearest neighbor search in which the nearest neighbor of a query object q in a collection of objects C is to be found. The trivial baseline algorithm would iterate over $c \in C$, calculate the distance $d(q, C)$ for each element, and remember the smallest of these values $d_{\min}$ together with the object

c, where this value came from. The time complexity of this process is $O(n)$ and in practical cases, the time is dominated by the n distance calculations.

In order to speed up this calculation, it is inevitable to skip some of these calculations of distances. A lower bound is a function, which assigns to each pair of objects $c_1, c_2 \in C$ a lower bound LB to the distance such that $\mathrm{LB}(c_1, c_2) \leq d(c_1, c_2)$. Such a function can operate on the objects c_1, c_2 themselves or on some approximation or simplification of them, $\mathcal{T}(c_1), \mathcal{T}(c_2)$. Lower bounds are interesting for speeding up nearest neighbor search if they can be evaluated significantly faster than the original distance function. Because then, for every element $c \in C$, the lower bound $\mathrm{LB}(c, q)$ is calculated and the true calculation of $d(c, q)$ is executed if and only if $\mathrm{LB}(c, q) < d_{\min}$, possibly updating $d_{\min}$.

It is sometimes possible to further increase the performance: if a lower bound can be calculated for a collection $G \subseteq C$ of objects, all these objects can be pruned if $\mathrm{LB}(c, q) \geq d_{\min}$ with a single calculation of the lower bound.

This second approach can be used on tree indexing structures in which each inner node is described as a summary object of all data below that node (e.g., a minimal enclosing rectangle for the R^* tree) in order to prune a substantial part of the dataset organized in large subtrees. For data that can be suitably represented in a tree, the average complexity becomes sublinear, e.g., $O(\log n)$.

3. BACR: BLOOM FILTER AGGREGATED CELL REPRESENTATION

The idea behind BACR is to use the memory-efficiency of Bloom filters in order to increase the amount of data that can be processed in memory for specific queries and distance functions and thereby reduce the amount of time needed to answer typical queries in situations, where spatial index structures fail, either due to the inefficiency of well-known aggregations including MBR-type (e.g., the R* tree and variants) and SAX or due to the need of integrating nonspatial information into the index (e.g., the ID of a vehicle, some discrete sensor status, etc.). Furthermore, the BACR representation allows for jointly processing non-local queries: When selecting trajectories that contain two distinct places, classical indices face a problem: Either two local range queries are executed independently from each other and their results are intersected, or the smallest range enclosing both places is being used for a very large search space in which many elements have to be examined. The random placement of information in a very short bit array as provided by the Bloom filter, however, allows for jointly querying for many different places with a constant amount of work and without realizing possibly large intermediate results or intersecting large sets.

Figure 1 depicts this situation: When querying for all trajectories that meet both small dashed rectangles, a classical local index creates two sets, namely $\{A, B, C, D\}$ and $\{A, B, E, F\}$ and calculates the result $\{A, B\}$ as the intersection of these sets. Alternatively, it examines all trajectories that meet the large dashed query area, which results in examining all depicted trajectories. With the BACR representation, the trajectories A and B can be identified as fitting to the query in a single operation without needing to create expensive intermediate results.

The BACR representation of some trajectory is derived by

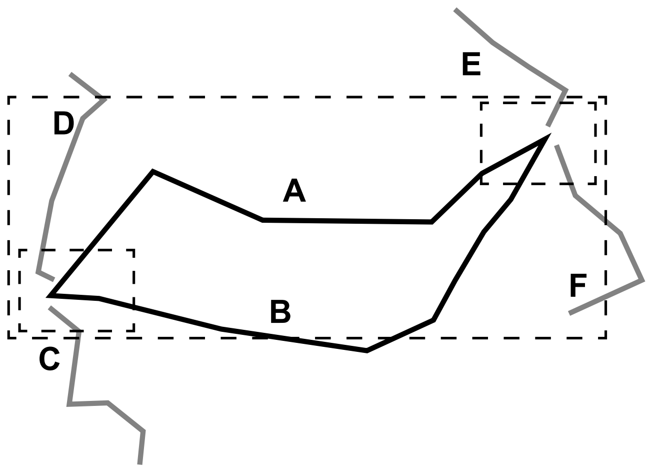


Figure 1: Non-local queries for trajectories

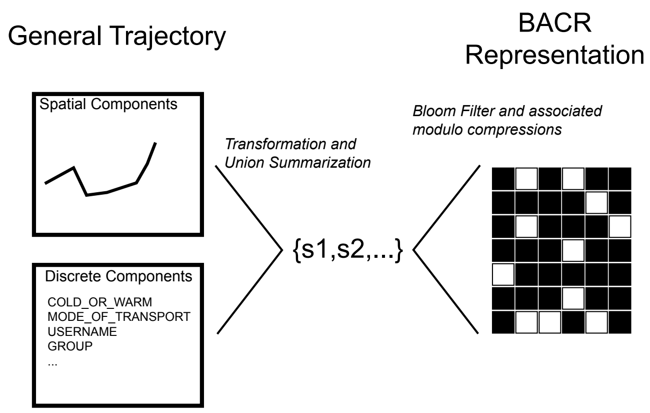


Figure 2: BACR representations of trajectories

first transforming the trajectory into a sequence of sets of strings using a suitable transformation $\mathcal{T}$. For spatial trajectories, we propose to use the widely known geohash Z-curve coding while the index does not exploit properties of the strings generated for pure spatial trajectories [9, 14]. These strings are then inserted into a well-configured, small Bloom filter. This idea of representing location sets as Bloom filters of geohashes has appeared in [10], but only support for the baseline subset query has been discussed. Figure 2 illustrates this construction.

4. SUBSET QUERIES ON BACR

A simplification operation is usually introduced in order to allow for approximate calculation of object relations or in order to speed up other algorithms like similarity search by providing bounds for the error type introduced by simplification. But before the details get presented, some basic queries will first be explained.

4.1 Baseline Exact Subset Query

A BACR representation of a trajectory supports a subset query. This query is directly supported for two types of query objects: a trajectory (alternatively any set of points) or a BACR representation of a trajectory.

For the first case, every element of the trajectory gets transformed using the transformation $\mathcal{T}$, which has been used in creating the BACR representation. Then, the Bloom filter is tested using each single string $\mathcal{T}(p_i)$ for a trajectory $t = (p_1, \ldots, p_n)$. This approach allows for testing any given set of points. Note that this query is only exact up to the inherent error of the transformation $\mathcal{T}$: a point meets a trajectory if and only if there exists a point in the trajectory that maps to the same string under $\mathcal{T}$. For the geohash case, this means that the trajectory meets the query geohash cell. This approach to querying has the advantage that query processing can be abandoned as early as the first contradicting query point q is processed.

For the second case, we can use the fact that the Bloom filter data structure directly allows for deciding whether a Bloom filter models a subset of another Bloom filter. Therefore, we check whether the database filter has been set to one everywhere where the query filter contains a one. If this is not the case, the query is rejected. In this case, we can also speed up query processing in two different ways: we can check the relationship of both filters on a subset of the bit locations of the filter only, which is equivalent to using smaller Bloom filters. Note that this query increases the error given by false positives. Alternatively, we can check all locations and abandon the processing as early as we find a zero indicating a contradiction.

4.2 Fast Exact Subset Queries

In order to query large sets of Bloom filters for relevant subsets in a scalable way, we propose the following binary search construction on filters based on recursive histograms of fractions of zeros.

Definition 1
A ***histogram of fractions of zeros*** *of length l, H^l_F is derived from a Bloom filter F by splitting the filter into equal parts of length l and assigning to a histogram slot i the mean of the $i-th$ part, namely:*

$$H(i) := \frac{1}{l} \sum_{j=li}^{l(i+1)-1} F(j)$$
$$= \text{mean}\{F(li), F(li+1), \ldots, F(li+l-1)\}$$

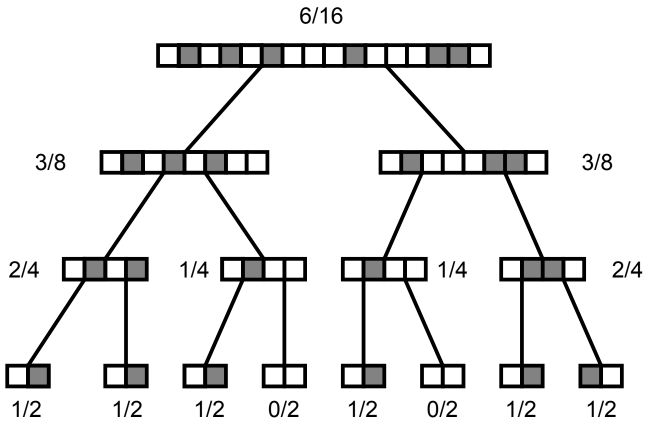


Figure 3: Pyramidal Histogram of Fractions of Zeros

With this definition, we can formulate a novel mechanism to quickly reject BACR representations of which a given query in BACR representation is not a subset:

Observation 1 (Pruning Rule)
Given Bloom filters F_A, F_B of equal configuration and their associated histograms of fractions of zeros H_A, H_B of equal length. Then

$$\exists_i H_A(i) > H_B(i) \Rightarrow A \nsubseteq B$$

Using a histogram of length 1, we could, for example, quickly prune large parts of the database based on a comparison of a single floating point value. However, this extreme case does only cover the (probabilistically expected) amount of elements inserted into the filter.

Currently, the choice of the histogram length is tricky: When it is chosen too small, queries are fast, but the pruning power is too low and the candidate set becomes large. Conversely, choosing a higher histogram length leads to higher query time and unneccessary computation. Therefore, we can refine this pruning approach by using an increasing histogram length recursively in a pyramid.

Definition 2 (Recursive Pruning Pyramid)
Given a database of Bloom filters F_i of length $m = 2^l$ a power of two, create the index pyramid for each filter F_i consisting of the following histograms of fractions of zeros:

$$H^{2^l}_{F_i}, H^{2^{l-1}}_{F_i}, \ldots, H^2_{F_i}.$$

Given a query Bloom filter G, calculate recursively the pyramid

$$H^{2^l}_G, H^{2^{l-1}}_G, \ldots, H^2_G,$$

and reject candidates F_i using the pruning rule Observation 1.

Figure 3 depicts the pyramid construction of a Bloom filter array with 16 bits. Black blocks represent bits that have been set while white bits represent unset bits.

5. JACCARD DISTANCE ON BACR

The Bloom filter and especially the BACR representation allows for more than just extending the element relation of the set model to detecting subsets. Additionally, it is to be expected that the subset relation is too strict when processing trajectories in a fixed cellular subdivision: very similar trajectories might occasionally meet different cells. A single such cell would destroy any subset relation while the similarity of the trajectories is still evident.

However, there are widely-used set similarity measures, which measure the similarity of sets and not only their pairwise relations. One of the most widely known measures of this type is the Jaccard distance.

The Jaccard distance is based on the Jaccard index, which has a very simple definition as the quotient of the size of the intersection and the size of the union:

$$\mathrm{sim}_{\mathrm{Jaccard}}(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

It is a "good" similarity measure in that its converse is a metric called Jaccard distance

$$d_{\mathrm{Jaccard}}(A,B) = 1 - \mathrm{sim}_{\mathrm{Jaccard}}(A,B)$$

and in that it is the outcome of a very intuitive LSH scheme as explained in Section 1. Furthermore, this metric takes values in $[0,1]$, where 0 is attained for equal sets and 1 is attained for disjoint sets.

5.1 Exact Jaccard Distance

It is straightforward to calculate the exact Jaccard distance of two sets of strings by calculating the intersection and the union of both sets. In practice, both can be done easily by maintaining a sorted version of the sets. However, this approach suffers from large memory consumption in order to store all sets as sets of their elements and from the fact that sorting sets can be costly. In order to improve on that, we show in the following that BACR supports approximate and exact calculation of the Jaccard distance without realizing each set of strings in memory and without sorting.

5.2 Approximate Jaccard Distance

Following an idea of Swamidass and Baldi [11], we can estimate the number of elements in a Bloom filter as well as in a union and an intersection knowing only the Bloom filters and their configuration. This allows for queries based on the estimated size of the union, the estimated size of the intersection, and the estimated size of the involved sets themselves. The intersection query can be used to find, for example, all trajectories that have a certain spatial overlap with each other measured by the number of cells in their intersection.

Therefore, the number of elements that have been inserted into a Bloom filter is estimated from the fraction of zeros as follows:

$$n_A \approx -\frac{\log(\phi_{F_A})m}{k}$$

In this equation, n_A denotes the number of elements of a filter A, ϕ_{F_A} the fraction of zeros, m the number of slots, and k the number of hash functions.

Additionally, given two filters, their binary OR can be calculated representing the union of two filters and the number of elements of the union can be estimated:

$$E_{Union} = -\frac{\log(\phi_{F_{A\vee B}})m}{k}$$

Extending on the previous results, we can use a simple set identity to estimate the number of elements of the intersection of two filters using the estimated size of the union of both filters: The number of elements of the union of two sets is the sum of the number of the individual sets. However, the elements of the intersection $|A \cap B|$ get counted twice, once for the set A and once for the set B. We can correct for this by substracting the size of the intersection once:

$$|A \cup B| = |A| + |B| - |A \cap B| \tag{1}$$

This easily translates to an estimator for the size of the intersection as

$$E_{Intersection} = n_A + n_B - E_{Union}$$

Using this, we can estimate the Jaccard distance of two filters just from their filters as

$$E_{Jaccard} = 1 - \frac{E_{Intersection}}{E_{Union}}$$

We use this construction to calculate the approximate Jaccard distance.

However, both Bloom filters used in these approximate calculations could suffer from false positives. Therefore, there is no suitable bound: the estimated results can be both larger or smaller than the actual intersection. However, as the false positive probability can be controlled by changing the filter configuration, so can the accuracy of these approximations.

5.3 A Novel Lower Bound for the Jaccard Distance on BACR

The central element of the proposed BACR representation is the idea of using the Bloom filter data structure as a simplification operator $\mathcal{S}$ mapping a trajectory to a Bloom filter. For spatial trajectories, we propose to use the well-known geohash coding approach in order to first create a relatively small set of strings representing a trajectory. Though we have shown how to efficiently compute an approximation to the Jaccard distance between two such filters, which completely relies on filters, this approach does not allow for exact similarity search. The reason for this is that the estimators of the number of elements can under- or overestimate the number of elements of a filter due to the situation where more or less hash collisions occur than expected.

However, with a minor extension to the data structure and some minor limitation on the way of querying the database, we can give a lower bound on the Jaccard distance for every element of the trajectory database based on their BACR representations. Therefore, the following restrictions have to be taken into account, which can easily be guaranteed in many application scenarios: (1) The correct number n of *different* elements added to each filter is actually maintained and stored additionally to the Bloom filter and (2) the correct set of strings for a query is known at query time.

The first restriction is easily fulfilled by extending the BACR representation with a counter for each trajectory. This is reasonable in most applications, however, the filter does not support adding elements to trajectories in an online manner anymore without errors in this number. Still, this restriction applies for *single* trajectories at a time, which can easily be realized in memory from disk. Similarity search across all trajectories, however, does not need these correct string sets.

The second restriction is fulfilled by disabling queries using BACR representations: For a query, the set of strings has to be known. Consequently, either the query is formulated as a set of strings or a real trajectory is used as the query object and the set of strings is generated using the transformation $\mathcal{T}$ without the simplification operator $\mathcal{S}$. Note that this means that in similarity search, two BACR representations cannot be directly compared. Therefore, each BACR representation should be realizable in memory, that is the actual set of strings used to generate the BACR representation should be stored on disk such that it can be made available occasionally.

With these restrictions, we can start constructing a lower bound: assuming that a query is actually given as a set S_1, while the database is accessible only in form of BACR summaries F_2, we want to construct a function

$$\mathrm{LB}(S_1, F_2) \leq d_{\mathrm{Jaccard}}(S_1, S_2) = 1 - \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

With this function, we can speed up range queries and similarity search as trajectories can be rejected based on the lower bound calculated from their BACR representation only.

We proceed by estimating the numerator $|S_1 \cap S_2|$ by counting the number of elements of S_1, which match to the filter F_2. Let us call this value ψ_1

$$\psi_1 = \#\{\text{Elements of } S_1 \text{ which match } F_2\} \geq |S_1 \cap S_2|$$

It is clear that ψ_1 is larger or equal to the number $|S_1 \cap S_2|$. This intersection contains elements, which are in both S_1 and S_2 and the only source of error is a false positive in the filter F_2 leading to an element of S_1 wrongly counted as an element of S_2 and therefore as an element of $S_1 \cap S_2$.

Following this analysis, we can even give the expected error of this estimator: Given that the false-positive probability of the filter F_2 can be estimated from its fraction of zeros ϕ_2 as in

$$P(f.p.) \approx (1 - \phi_2)^k,$$

we can conclude that the expected number of false positives in calculating ψ_1 is given by

$$eFP \approx |S_1|(1 - \phi_2)^k$$

This probabilistic value eFP can be used to select the appropriate configuration of the Bloom filters in order to reach some amortized running time in similarity search with a known error of the lower bound. It is this value eFP, which controls the tightness of the bound and therefore the efficiency of pruning. Note, however, that this value will have to be propagated through the similarity measure for which a bound is calculated using ψ_1.

Now, we give an estimator for the denominator in a similar way: We approximate $S_1 \cup S_2$ by the number of elements of S_2 plus the number of elements of S_1, which do not match to the filter F_2. Again, the only source of error is given by an element of S_1 matching the filter F_2 leading to an element not counted towards $S_1 \cup S_2$ (we are counting those that do not match). Therefore, this estimator, called ψ_2 is smaller than the union:

$$\psi_2 = |S_2| + \#\{\text{Elements of } S_1 \text{ ,which do not match } F_2\} \\ \leq |S_1 \cup S_2|$$

Again, the error is controlled by $|S_1|$ times checking filter F_2 for the same expected number of false positives of

$$eFP \approx |S_1|(1 - \phi_2)^d,$$

So far, we have used the following elements for calculating some values ψ_1 and ψ_2: (1) The elements of S_1, (2) the number of different elements that have been added to F_2 (e.g., $|S_2|$), and (3) the filter containment relation of F_2 given by the Bloom filter construction.

By combining the inequalities for ϕ_1 and ϕ_2 it is easy to see that

$$LB(S_1, F_2) := 1 - \frac{\psi_1}{\psi_2} \leq d_{\mathrm{Jaccard}}(S_1, S_2) = 1 - \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

We will show that this lower bound is actually non-trivial and very efficient to compute in Section 7. However, this lower bound is not compatible with the union of sets. Consequently, it can be used to prune the search space in a linear walk of the dataset, while this construction cannot prune sets of elements at a time.

6. SIMILARITY SEARCH AND THE UNION OF SETS

The developments of the preceding section have led to a lower bound for the Jaccard distance based on Bloom filters. However, we still have to consider each sketch of an element of the trajectory database individually. This clearly results in a scaling limit: When the amount of trajectories grows, both the memory consumption and the query times grow linearly.

In geometric similarity search, a lot of work has been done with respect to pruning larger or smaller parts of the database in order to solve this problem. The central ingredient of this is separation: There are geometric objects (balls, minimum bounding rectangles, half spaces, etc.) such that most elements belong to one and only one such object. A point, for example, can be inside a rectangle or not. The main problem of indexing trajectories is given by the fact, that trajectories cannot be separated in this way: Given a hypothetical separation of trajectories, consider the concatenation of two trajectories taken from different separation sets, which forms a valid trajectory belongs to both sets at the same time. As a consequence of the absence of separation, many approaches to trajectory index use their point-based substructure instead or generate many small segments.

With this paper, we explore a different direction to recover distance-based prunings for groups of trajectories (e.g., sets of sets):

In order to be able to use distance-based pruning for multiple objects at a time, it must be possible to compare a concrete query element t_q with a summary representation $S = \mathcal{S}(\{t_1, \ldots, t_n\})$ of different trajectories using a function often called **mindist** in a way such that

$$d(t_q, t_i) \geq \mathbf{mindist}(t_q, \mathcal{S}(S)) \;\; \forall\; 1 \leq i \leq n. \qquad (2)$$

In other words, nearness propagates from sets to its elements: if a query is not near to a set, it is not near to any element.

In this case, several queries can replace calculating the distance between a query (or candidate) object t_q and each trajectory $t_i \in S$ by calculating the function **mindist** once for the set.

With respect to our BACR representation, there is a straightforward candidate for the summary operation $\mathcal{S}$ given by the union of sets. This operation is directly supported by Bloom filters using a binary OR operation on the filter array not introducing additional errors. Unfortunately, most set similarities are incompatible with the union operation that is, the function **mindist** cannot be given by the distance of the discretized query trajectory t_q with some "union" of the BACR representation of the trajectories in a set S.

As an example consider the Jaccard distance between two sets Q and A and take a single element set $B = \{b\}$. In this situation, Equation 2 results in

$$\frac{\alpha}{\beta} = \frac{|Q \cap A|}{|Q \cup A|} \leq \frac{|Q \cap (A \cup \{b\})|}{|Q \cup (A \cup \{b\})|} = \frac{\gamma}{\delta},$$

where $\alpha \ldots \delta$ are just names for the individual parts. In the case that $b \notin A$ and $b \notin Q$, the numerator does not change ($\alpha = \gamma$) while the denominator increases by one ($\delta = \beta + 1$). This clearly contradicts the inequality. And it is clear that there is no direct solution in this situation, as the size of the error depends on the intersection of the query set Q with each part of some summary. So either we know the parts of the summary rendering the summary operation useless or we can't build a **mindist** function for variable queries Q in this way.

Still, there is a feasible solution to this problem at least for similarity search: The Jaccard distance is monotonous with the Dice coefficient [12], which is another set similarity index given by

$$\mathrm{sim}_{\mathrm{Dice}}(A, B) = \frac{2|A \cap B|}{nA + nB}$$

In essence, this means that

$$\mathrm{sim}_{\mathrm{Dice}}(A, B) \leq \mathrm{sim}_{\mathrm{Dice}}(C, D) \Leftrightarrow \mathrm{sim}_{\mathrm{Jacc.}}(A, B) \leq \mathrm{sim}_{\mathrm{Jacc.}}(C, D)$$

In other words: when using the Dice index instead of the Jaccard index, the objects will be ordered equally. Note, however, that the complement of the Dice coefficient does not obey the triangular inequality.

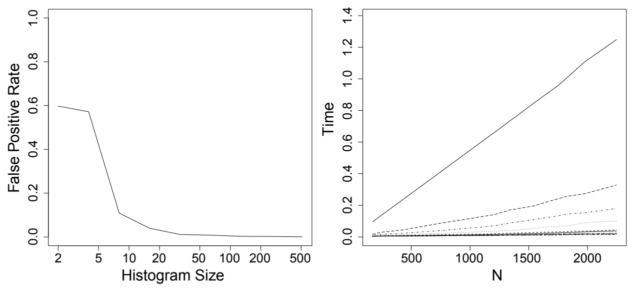
With the Dice coefficient, the union of the sets is not used at all and, consequently, we don't run into the same problem as before. Still, we need to know the sizes of the individual sets or summarize those.

Inequality 2 rewrites to

$$\mathrm{sim}_{\mathrm{Dice}}(t_q, t_i) \leq \mathbf{maxindex}(t_q, \mathcal{S}(S_1, \ldots, S_k)), \qquad (3)$$

where **maxindex** denotes the maximal Dice index possible for the summarization $\mathcal{S}(S)$, which can be calculated using the minimum size m of the different sets $S_1 \ldots S_k$.

Using the BACR representation and a query object t_q realized as a set Q of strings, we can easily calculate an estimate of $|Q \cap \mathcal{S}(S)|$ by testing the Bloom filter for every element of Q. Depending on the configuration of the Bloom filter, this results in a sligthly larger value than $|Q \cap \mathcal{S}(S)|$ due to possible false positives. However, it will never report a smaller value. Putting this together, we see that when we



(a) Impact of Histogram Size on False Positives (b) Scaling for Increasing Database Size

Figure 4: Performance of the Histogram of Fractions of Zeros on the Geolife Dataset

keep track of the minimal size of a set inserted into a union-based summarization, we can calculate a function **mindist**, for which

$$d(t_q, t_i) \geq \mathbf{mindist}(t_q, \mathcal{S}(S)) \qquad (4)$$

holds, where d is the "distance"[1] given by the complement of the Dice coefficient. Using the fact that the Dice coefficient is monotonous with the Jaccard index, we can use this approach for k-nearest neighbor search. The nearest neighbors with respect to the complement of the Dice coefficient are the nearest neighbors with respect to the Jaccard index.

We can use this concept in order to compress sets of similar objects: Some sets will be completely unrelated having an empty intersection. Other sets will have a varying similarity and compressing clusters of similar sets into a single union-summarized BACR representation can be fruitful as long as the filter is powerful not to introduce too many false positives and the actual size of the individual sets is similar enough such that taking the minimum does not result in severe loss of tightness.

However, strategies to select subsets of trajectories to be represented by a single BACR are quickly getting complicated and go beyond the scope of this paper. Still, this is a very interesting question on its own and left for future research.

7. EXPERIMENTS

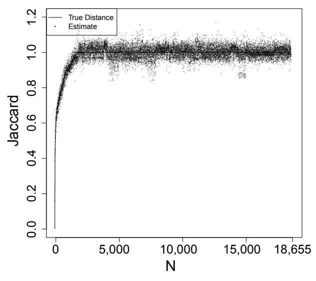
In order to assess the performance and quality of the different approaches presented in this paper, we conducted experiments on synthetic datasets as well as on large trajectory databases.

All algorithms were implemented in C++ using g++ (4.7.2) with full optimizations and have been executed on a medium-performance laptop (Intel Core i7-3540M, Quadcore, 3 GHz, 8GB RAM) using a single core. If not otherwise stated, times have been estimated by performing the operation 100 times in order to reduce random influences.

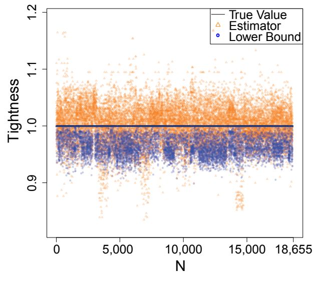
7.1 Performance on Spatial Datasets

Table 1 depicts the results of the main evaluation on three real datasets, a sample "Small Geolife" of the Geolife dataset [16] with 2,890 trajectories containing roughly 5 million points, "Full Geolife" with 18,670 trajectories containing roughly 25

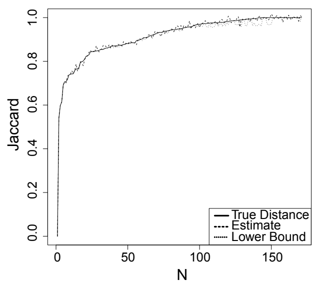
[1]Note that this function does not obey the triangle inequality.



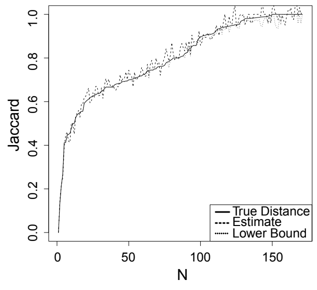
(a) Approximate Jaccard Distance



(b) Tightness



(c) 2048 bit (8 hash functions)



(d) 256 bit (4 hash functions)

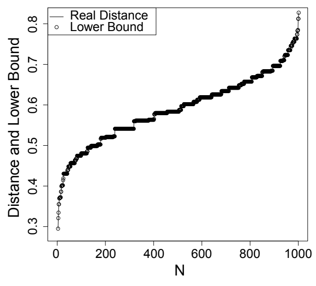
Figure 5: Experimental Results on Geolife

million points, and "Roma" with 316 trajectories containing roughly 22 million points.

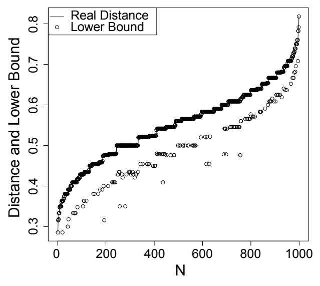
The evaluations were performed with geohashes of five and six characters representing reasonable sizes for trajectory data. The Fraction of Zeros (FOZ) of the Bloom filter is given. Note, that this value should be larger than 50% in order for the Bloom filter to work effectively. All experiments show the expected results: The fast subset query using histograms of zeros is speeding up the Baseline Subset Query using the Bloom filters directly by a varying amount. This was expected, as the Bloom filter results in a random hash field and the skewness of such random field tends to be more useful for a higher fraction of zeros. Also, the false positive rate of the subset query using BACR is as expected: Generally, it is smaller for filters with larger fraction of zeros and higher otherwise.

Figure 4 shows the effect of histograms of fractions of zeros. For increasing length of the histogram, the false positive rate starts out quite high, but drops quickly. Even for small numbers of 16 or 32 for the histogram length, the false positive probability is below 10%. To the right, you can see the impact on running times. The computational complexity of a query increases linearily with the size of the histogram chosen. This can be seen from the increasing slope in the right figure. Consequently, a linear tradeoff between power and false positive rate is observed, which must be tuned according to the application scenario and database size.

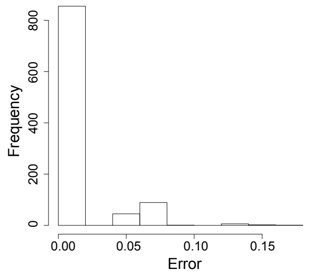
Additionally, Figure 5 depicts the results of indexing Geolife with a geohash length of 7: Due to the large number of geohash cells per trajectory, these represent the trajectories in a very detailed way resulting in smooth results: Figure 5(a) depicts the true Jaccard distance and the result of the approximate Jaccard calculation, which quite well reflect each other. This can also be seen from the tightness



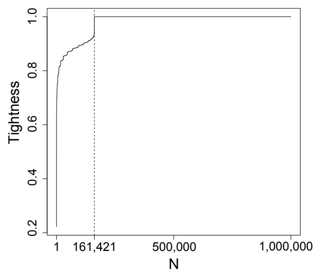
(a) Optimal Case



(b) Suboptimal Case



(c) Histogram of Errors



(d) Tightness

Figure 6: Tightness of the Lower Bound

diagram in Figure 5(b) in which the tightness of the lower bound as well as the tightness of the approximate algorithm are compared. Figures 5(c) and 5(d) depict that an increase in the size of the Bloom filter (together with a fitting number of hash functions) results in better quality. Note, however, that such a fine resolution of geohashes is a good stress test for the system albeit the Jaccard similarity relates only a fraction of the database.

7.2 Evaluation of the Lower Bound on Random String Sets

We performed experiments on this lower bound by creating random sets of 25 strings of length three over an alphabet of three letters. Thereby, some strings will occur several times. Then, these string sets are made unique such that every element is only inside once. These two sets are used to create a Bloom filter using the Murmur hash and seven hash functions.

We use two sizes m for the filter: choosing an optimal size of 256 bits leads to an expected false positive rate of

$$E = 0.6185^{\frac{m}{256}} \leq 0.6185^{\frac{25}{256}} \approx 0.0073$$

while using only 128 bits leads to

$$E = 0.6185^{\frac{m}{128}} \leq 0.6185^{\frac{25}{128}} \approx 0.085440.$$

The effectiveness of a lower bound can be measured in different ways depending on the context. The most basic and general measure is called *tightness* and we define it, following [8], as follows:

$$T = \frac{\text{Value of the Lower Bound}}{\text{True Value}}$$

This creates a value between zero and one, the larger, the better.

For both cases, we create 1,000,000 string sets and calculate the tightness. Figure 6 depicts the results. However,

Metric / Algorithm	Small Geolife	Full Geolife	Roma	Small Geolife	Roma
Number of Trajectories	2,890	18,670	316	2,890	316
Number of Points	4,997,092	24,876,978	21,817,536	4,997,092	21,817,536
Geohash Length	5	5	5	6	6
Number of Hash Functions	3	5	3	3	3
Bits per Trajectory	32	128	192	128	2048
Average FOZ	70.7%	85.45%	59.53%	69.5%	58.76%
Baseline Subset Query	0.70s	44.49s	0.12s	0.81s	0.12s
Fast Subset Query	0.62s	26.31s	0.08s	0.70s	0.08s
Subset Query FP-Rate	2.99 %	5.59%	9.78%	17.06%	28.49%
Baseline Jaccard	25.737s	971.74s	1.62s	184.96s	60.002s
Approximate Jaccard	5.87s	474.82s	0.179s	11.69s	2.12s
Lower Bound	2.93s	124.55s	0.05s	4.2s	0.17s
Average Tightness	0.54	0.47	0.26	0.39	0.23
RMSE Tightness	0.58	0.64	0.69	0.71	0.45

Table 1: Summary of query performance on various datasets

the plots show only the first 1,000 data points in order to make them more readable. For the optimal configuration, the tightness was constantly one and the error was zero. The filter has been configured to hold 25 elements or less. The fraction of zeros of these filters is therefore larger than 50% as we added less elements due to string collisions. The unique string sets had varying sizes from 9 to 24 with a median of 16 for both S_1 and S_2 and the fraction of zeros ranged from 0.511 to 0.785 with a median of 0.63. Note that the false positive rate depends only on the fraction of zeros and that the fraction of zeros of a filter gives a probabilistic hint on how good the presented bound is. For the suboptimal case, the fraction of zeros is much smaller ranging from 0.273 to 0.640 with a median of 0.42. This leads to false positives and consequently to differences between the presented bound and the actual distance. The absolute errors were smaller than 0.258. Though this is a large error given that the Jaccard distance takes values between zero and one, these errors occur seldom. The median (and even the third quadrant) of the error is zero and the mean is 0.011. The lower bound was suboptimal in only 161,421 cases as depicted in 6(d).

The conclusion of these results is that the bound is very effective for random sets. In spatial datasets created with geohash, however, there are only few strings, still, the bound is useful as shown in the following section.

7.3 Performance in Nearest Neighbor Search on Spatial Datasets

In order to fully understand the impact of a lower bound, we should also discuss it with respect to a typical dataset. In nearest neighbor search, the lower bound is usually used to prune elements of the database once they are farther away than the current best one. Therefore, the actual performance of a lower bound depends on the distribution of the distances in the database: the amount of computation time saved by pruning is not actually related to the tightness, which is more useful in comparing different lower bounds.

In this situation, a widely accepted measure is given by the *pruning ratio*, which can be defined as follows:

$$P = \frac{\text{Number of Omitted Elements}}{\text{Total Number of Elements}}$$

This measure creates a value between zero and one with one representing the best case. Similarly, we can define the *time ratio*

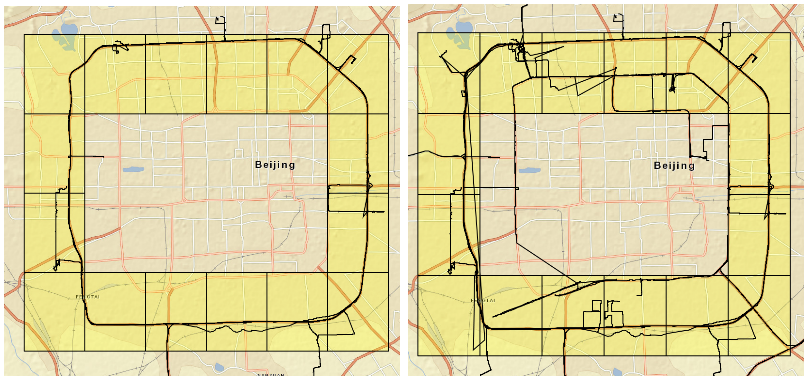
$$TR = \frac{\text{Time used with Lower Bound}}{\text{Time used without Lower Bound}}$$

Additionally, we record the average query times with and without lower bound. In the following, we use a geohash configuration of five characters, 128 bit per trajectory as well as three hash functions.

For the Roma dataset we measured a pruning ratio of 33.79%. The time ratio is even more convincing with a value of 64.12%. This means that the nearest neighbor search was performed by examining only 66.21% of the search space and by using only 35.88% of the time compared to the baseline. We note an average query time of 3.21 ms with the lower bound in place and an average query time of 4.63ms without a lower bound. For the Geolife dataset, we noticed an average time ratio of 31.2% and an average pruning ratio of 34.6% leading to average query times of 3.72ms and 6.54ms. In general, keep in mind, that all experiments were performend inside main memory without disk access and that the memory consumption of the lower bound is very small. Once datasets grow such large that they cannot be held in main memory, the lower bound will reduce disk or network I/O leading to an even stronger performance increase.

7.4 Example Queries and Results

In order to illustrate the usefulness of representing spatial similarity by the set similarity of sets of geohash strings, we implemented the system as a web application in which queries can be created by clicking several locations creating the query geohash set. Figure 7 depicts the results of performing a subset and Jaccard queries for the 4th ring road of Beijing. It is evident that the Jaccard query returned more diverse results. Still, all trajectories selected by the Jaccard query have some significant overlap with the 4th ring. This also relaxes the problem that the subset query is not stable with respect to measurement outliers. While the subset relation can be broken by a single outlier, this is not the case for the Jaccard distance.



(a) Subset Query (b) Jaccard Query

Figure 7: Example Queries

8. CONCLUSION

In this paper, a novel way of combining the Bloom filter technique and the geohash location to string coding approach is proposed in order to create a memory-efficient, inexact representation of trajectories facilitating big data applications. Motivated by the set nature of the geohash coding of trajectories, we study the Jaccard distance of sets, give an approximate calculation and a lower bound, which significantly reduced the complexity of nearest neighbor queries with respect to the Jaccard distance. We show that this lower bound is very strong for uniformly distributed sets of strings, and that it is sufficiently strong to help in similarity search of trajectories.

We also note that the Jaccard distance, and therefore also the presented lower bound, cannot be applied on unions of sets ruling out an index construction based on representing collections of objects using a single feature representation. Still, we note and explain that the Jaccard distance can be replaced by the Dice index, which does not have this limitation. An important remark is that this Dice index, though it is compatible with the union of sets, is not indexable by locality-sensitive hashing rendering the given approach even more unique. For future work, we envision improvements on the construction of the lower bound, the calculation of the lower bound and the representation of time in BACR summaries as well as evaluations and adaptions to bioinformatics and chemistry.

9. REFERENCES

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.

[3] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 327–336. ACM, 1998.

[4] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

[5] K. Deng, K. Xie, K. Zheng, and X. Zhou. Trajectory indexing and retrieval. In *Computing with Spatial Trajectories*, pages 35–60. 2011.

[6] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[7] X. Gong, Y. Xiong, W. Huang, L. Chen, Q. Lu, and Y. Hu. Fast similarity search of multi-dimensional time series via segment rotation. In M. Renz, C. Shahabi, X. Zhou, and M. A. Cheema, editors, *Database Systems for Advanced Applications*, volume 9049 of *Lecture Notes in Computer Science*, pages 108–124. Springer International Publishing, 2015.

[8] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[9] G. Niemeyer. Geohash, 2008.

[10] P. Ruppel and A. Küpper. Geocookie: a space-efficient representation of geographic location sets.

[11] S. J. Swamidass and P. Baldi. Mathematical correction for fingerprint similarity measures to improve chemical retrieval. *Journal of chemical information and modeling*, 47(3):952–964, 2007.

[12] P. Willett, J. M. Barnard, and G. M. Downs. Chemical similarity searching. *Journal of chemical information and computer sciences*, 38(6):983–996, 1998.

[13] J. J.-C. Ying, W.-C. Lee, T.-C. Weng, and V. S. Tseng. Semantic trajectory mining for location prediction. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 34–43. ACM, 2011.

[14] C.-T. Zhang, R. Zhang, and H.-Y. Ou. The z curve database: a graphic representation of genome sequences. *Bioinformatics*, 19(5):593–599, 2003.

[15] K. Zheng, S. Shang, N. J. Yuan, and Y. Yang. Towards efficient search for activity trajectories. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 230–241. IEEE, 2013.

[16] Y. Zheng, X. Xie, and W.-Y. Ma. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.