Spatio-temporal Range Searching Over Compressed Kinetic Sensor Data

Sorelle A. Friedler*
sorelle@cs.umd.edu
http://www.cs.umd.edu/~sorelle

David M. Mount†
mount@cs.umd.edu
http://www.cs.umd.edu/~mount

Dept. of Computer Science, University of Maryland, College Park, MD 20742

Abstract

As sensor networks increase in size and number, efficient techniques are required to process the very large data sets that they generate. Frequently, sensor networks monitor objects in motion within their vicinity; the data associated with the movement of these objects is known as kinetic data. In an earlier paper we introduced an algorithm which, given a set of sensor observations, losslessly compresses this data to a size that is within a constant factor of the asymptotically optimal joint entropy bound. In this paper we present an efficient algorithm for answering spatio-temporal range queries. Our algorithm operates on a compressed representation of the data, without the need to decompress it. We analyze the efficiency of our algorithm in terms of a natural measure of information content, the joint entropy of the sensor outputs. We show that with space roughly equal to entropy, queries can be answered in time that is roughly logarithmic in entropy. In addition, we show experimentally that, on real-world data, our range searching structures use less space and have faster query times than the naive versions. These results represent the first solutions to range searching problems over compressed kinetic sensor data.

1 Introduction

Sensor networks and the data they collect have become increasingly prevalent. They are frequently employed to observe objects in motion and are used to record traffic data [16, 29], observe wildlife migration patterns [23,31], and observe motion from many other settings [2]. In order to perform accurate statistical analyses of this data over arbitrary periods of time, the data must be faithfully recorded and stored. For example, a large sensor network observing a city's traffic patterns may generate gigabytes of data each day [16]. When considered over a year or multiple cities, this quickly becomes terabytes of data. The vast quantities of such data necessitate compression of the sensor observations, yet analyses of these observations is desirable. Ideally, such analysis should operate over the compressed data without the need to decompress it. In order to perform statistical analyses of the data, it is often desirable that retrieval queries be supported. In this paper, we present the first data structure and algorithms for answering range searching queries over compressed data streams arising from large sensor networks.

In an earlier paper [12], we presented an algorithm for losslessly compressing kinetic sensor data and a framework for analyzing its performance. (See Section 2 for a brief introduction.) Here, we are interested in analyzing the data arising from sensor networks; we do not consider the traditional issues in the study of such networks themselves, for example power failure, robustness, etc. [2]. We assume that we are given a set of sensors, which are at fixed locations in a space of constant dimension (our results apply generally to metric spaces of constant doubling dimension [20].) These sensors monitor the movement of a number of kinetic objects. Each sensor monitors an associated region of space, and at regular time steps it records an occupancy count of the number of objects passing through its region.

*The work of Sorelle Friedler has been supported in part by the AT&T Labs Fellowship Program and the Ann G. Wylie Dissertation Fellowship.

†The work of David Mount has been supported in part by the National Science Foundation under grant CCR-0635099 and the Office of Naval Research under grant N00014-08-1-1015

Over time, each sensor produces a string of occupancy counts; the problem considered in [12] is how to compress all these strings.

The assumptions made by the framework [12] about the form of the data generated are realistic in terms of sensor models for capabilities required [14], and in terms of existence of current data within this model. One of the application domains generating data within this framework is vehicle traffic monitoring. Road-embedded sensors, known as *loop detectors*, are frequently placed within highways to detect vehicle motion and translate this into vehicle counts. The aggregated vehicle occupancy counts per time interval are known within the traffic monitoring literature as *volume counts*. These volume counts are collected and archived, often in the form of one string of counts per sensor, the same form described by the framework. For examples of specific data repositories in this form, see [6, 22, 24, 33]. Other databases may store sensor activation time stamps instead of explicit counts, but these can be easily translated into data of the form assumed within the framework used here. Such data sets include those monitoring the locations of fish, birds, moose, and other wildlife [8, 18, 25, 32].

Previous compression of sensor data in the literature has focused largely on approximation algorithms in the streaming model or lossy compression of the data. We consider lossless compression. This is often more appropriate in scientific contexts, where analysis is performed after the data has been collected and accurate results are required. The analysis of these results may necessarily include consideration of outliers or unusual data features that might be smoothed away by lossy compression techniques. In addition, in scientific contexts the loss of data associated with any kind of lossy compression is considered unacceptable. Lossless compression algorithms have been studied in the single-string setting [17,27,35,36] but remain mostly unstudied in a sensor-based setting [12].

In order to query observed sensor data, which ranges over time and space, we need to consider both temporal and spatial queries. *Temporal range queries* are given a time interval and return an aggregation of the observations over that interval for a single sensor. *Spatial range queries* are given some region of space (e.g., a rectangle, sphere, or halfplane) and return an aggregation of the observations within that region at a single time instant. *Spatio-temporal range queries* generalize these by returning an aggregation restricted by both a temporal and a spatial range. We assume that occupancy counts are taken from a commutative semigroup of fixed size, and the result is a semigroup sum over the range. There are many different data structures for range searching (on uncompressed data), depending on the properties of the underlying space, the nature of the ranges, properties of the semigroup, and whether approximation is allowed [1].

We present data structures for storing compressed sensor data and algorithms for performing spatio-temporal range queries over this data. We analyze the quality of these range searching algorithms in terms of both time and space by considering the information content of the set of sensor outputs. There are two well-known ways in which to define the information content of a string, classical statistical (Shannon) entropy and empirical entropy. Statistical entropy [30] is defined under the assumption that the source X is drawn from a stationary, ergodic random process. The normalized statistical entropy, denoted $H(X)$, provides a lower bound on the number of bits needed to encode a character of X. In contrast, the empirical entropy [19, 21], denoted $H_k(X)$, while similar in spirit to the statistical entropy, assumes no underlying random process and relies only on the observed string and the context of the most recent k characters. These definitions and distinctions are discussed in more detail in a companion paper [13].

Previously, retrieval over compressed text (without relying on decompression) has been studied in the context of strings [3, 10, 11, 15] and XML files [9]. For example, Ferragina and Venturini [11] show that it is possible to retrieve all occurrences of a given pattern in the compressed text with query time equal to $O(1+\frac{\ell}{\log T})$ where ℓ is the length of the pattern and T is the length of the string X. Their space requirement is $T \cdot H_k(X) + o(T)$ bits. However, their data structure allows substring queries, which are very different from semigroup range searching queries, which we consider here.

Although here we will present data structures that operate in main memory, for the large data sets generated by sensor networks it may also be useful to consider Input/Output (I/O) efficient structures. Since the data structures described here are based on simple, practical structures, modifications to I/O-efficient versions should be straightforwardly based on known I/O-efficient equivalents. Specifically, the temporal structure described could be modified to be based on an underlying I/O-efficient compressed text structure [7] and combined with a spatio-temporal structure modified to be based on an I/O-efficient kd-tree [26, 28].

Bounds for Range Searching

	Temporal	Spatio-temporal
Preprocessing time	$O(\text{Enc}(X))$	$O(\text{Enc}(\mathbf{X}))$
Query time	$O(\log T)$	$O(((1/\varepsilon^{d-1}) + \log S) \log T)$
Space	$O(\text{Enc}(X))$	$O(\text{Enc}(\mathbf{X}) \log S)$

Table 1: Time and space bounds for temporal range searching and ε-approximate spatio-temporal range searching for fat convex ranges in $\mathbb{R}^d$. S is the number of sensors in the network, T is the length of the observation period, and $\text{Enc}(X)$ and $\text{Enc}(\mathbf{X})$ denote the sizes of the compressed representations for single sensor stream (for temporal range searching) and sensor system (for spatio-temporal range searching), respectively.

In this paper we present the first range query results over compressed kinetic sensor data. Specifically, we consider the problems of temporal range searching and spatio-temporal range searching for fat convex ranges (e.g., spheres, rectangles with low aspect ratio, etc. [4]). Although we consider the case of fat convex ranges here, the techniques could be more generally applied as long as some degree of approximation is maintained. Ultimately, the limiting factor in the choice of spatial range is that of the size and shape of the sensor region, which is given as part of the input and which represents the resolution of the given points' positions.

As mentioned earlier, we analyze our algorithms in terms of the joint entropy of the sensor outputs. The preprocessing makes only one pass over the compressed data, and thus it can be performed as the data is being collected. The query bounds are logarithmic in the input size. The space bounds, given in bits, match the entropy lower bound up to constant factors. Specific bounds are given in Table 1.

In addition to theoretical results, we present experimental evaluation of our temporal range searching structure. These results show that, in addition to being theoretically efficient, our data structure offers a roughly 50-fold improvement in space in comparison to the raw data. These improvements increase as the data sets become larger. Both our temporal and spatio-temporal data structures are quite practical, being based on very simple data structures (tries, binary trees, and quadtrees, in particular).

2 Framework for Kinetic Sensor Data

In an earlier paper [12] we introduced a framework and a lossless compression scheme for discrete kinetic data observed by a sensor network. This framework will be used as a basis for the results of this paper. We begin with some basic definitions about the structure of the sensor network and the associated observed data streams. Consider a static sensor network with S sensors, monitoring the motion of a collection of moving objects. Let P be a point set indicating the sensor locations. All sensors are assumed to operate over T synchronized time steps. Each sensor observes the motion of objects in some region surrounding it, and records an *occupancy count* indicating the number of objects passing within its region during the observed time step. No assumptions are made about the nature of the point motion nor the nature of the sensor regions (e.g., their shapes, density, disjointness, etc.).

Central to our framework is the notion that each sensor's output is statistically dependent only on a relatively small number of nearby sensors. For some point $p \in P$, let $NN_m(p) \subseteq P$ be the m nearest neighbors of p. Sensors i and j with associated sensor positions $p_i, p_j \in P$ are said to be *mutually m-close* if $p_i \in NN_m(p_j)$ and $p_j \in NN_m(p_i)$. For a constant m, a sensor system is said to be *m-local* if all pairs of sensors that are not mutually m-close are statistically independent. Experimental evaluation of the locality assumptions of the framework is given in Section 5.

In [12] we introduced a compression algorithm, *PartitionCompress*, which operates on an m-local sensor system. (The algorithm is sketched in the proof of Lemma 4.1.) It compresses the sensor outputs to within a constant factor c (independent of m but depending on dimension) of the optimal joint entropy bound.

Intuitively, the compression algorithm is based on the following idea. If two sensor streams are statistically independent, they may be compressed independently from each other. If not, optimal compression can only be achieved if they are compressed jointly. The algorithm works by compressing the outputs

from clusters of nearest neighbor groups together, as if they were a single stream. In order to obtain the desired compression bounds, these clusters must be sufficiently well separated so that any two mutually m-close sensors are in the same cluster. *PartitionCompress* partitions the points into a constant number c (independent of m but depending on dimension) of subsets for which this is true and then compresses clusters together to take advantage of local dependencies. The compression of a single cluster may be performed using any string compression algorithm; to obtain the near optimal bound, this algorithm must compress streams to their optimal entropy bound. It is shown in a companion paper [13] that LZ78, the Lempel-Ziv dictionary compression algorithm [36], is sufficient for our purposes.

For the rest of the paper, we will use $\text{Enc}_{alg}(\mathbf{X})$ to denote the length of the encoded set of sensor outputs $\mathbf{X}$, where alg specifies the string compressor used by the compression algorithm of [12]. Since the LZ78 algorithm will suffice for our purposes, let $\text{Enc}(\mathbf{X}) = \text{Enc}_{\text{LZ78}}(\mathbf{X})$. In a companion paper [13], it is shown that $\text{Enc}(\mathbf{X})$ is on the order of the optimal space bound when analyzed in terms of either the statistical or empirical entropy. (A slightly weaker form of independence, called δ-independence, was also considered and it was shown that the bounds hold approximately under this weaker definition.)

3 Temporal Range Searching

In this section we describe a data structure that answers temporal range searching queries over a single compressed sensor stream. Let X be a sequence of sensor counts over time period $[1, T]$, which will be compressed and preprocessed into a data structure so that given any temporal range $[t_0, t_1] \in [1, T]$, the aggregated count over that time period can be calculated efficiently. We assume that the individual sensor counts are drawn from a semigroup, and the sum is taken over this semigroup. The space used by the data structure (in bits) will be asymptotically equal to that of the compressed string, and the query time will be logarithmic in T. Here is the main result of this section. Recall that, given string X, $\text{Enc}(X)$ denotes the length of the compressed encoding of X.

Theorem 3.1. *There exists a temporal range searching data structure, which given string X over a time period of length T, can be built in time $O(\text{Enc}(X))$, achieves query time $O(\log T)$, and uses space $O(\text{Enc}(X))$ bits.*

The remainder of this section is devoted to proving this theorem. Here we consider the simpler special case where the semigroup is in fact a group, which means that both addition and subtraction of counts are allowed. The semigroup case involves a more sophisticated data structure, and will be presented in the full version of the paper.

We begin by describing the preprocessing for our data structure in the group context, where subtraction of counts is allowed. First, the given sequence X is compressed using the LZ78 compression algorithm and the standard accompanying trie (also known as a *dictionary*) containing nodes that represent *words* is created [36]. We begin with a short overview of this algorithm. LZ78 scans over the input, putting characters into a trie so that each edge in the trie represents a single character. As the string is scanned from beginning to end, the prefix is looked up in the trie and the most recent character is added to that path in the trie. The resulting *word* is added to the compressed version of the string by simply storing a pointer to the bottom most node of the path in the dictionary. Let d be the number of words in the dictionary. Each word in the dictionary (possibly excepting the last) is used in the compressed version of the string exactly once. In addition, each word in the dictionary was generated only after all prefixes had previously been added, so the trie is *prefix-complete* [10]. We will make use of the fact, proved in a companion paper [13], that $d \log d = \text{Enc}(X)$.

Let us now discuss our preprocessing of the stream X. It involves two phases. The first takes place during the single scan through the input. The data is compressed using LZ78 compression, the associated trie is created, and pointers to word endings (called *anchor points*) are stored. Additionally, the aggregated value of each word (e.g. the sum of its component counts, or the *word sum*) is added to the associated node in the dictionary. This value can be found by adding the count at the current node to its parent's stored aggregated value as each letter is added to an existing word in the trie. This phase takes time $O(T)$ and we will refer to the result of this phase as the *compressed form* of the input. See Figure 1 for an example.

The second phase, which is the one we will analyze for its additional non-compression related time, consists of creating a binary search tree over the anchor points and initializing auxiliary data structures.

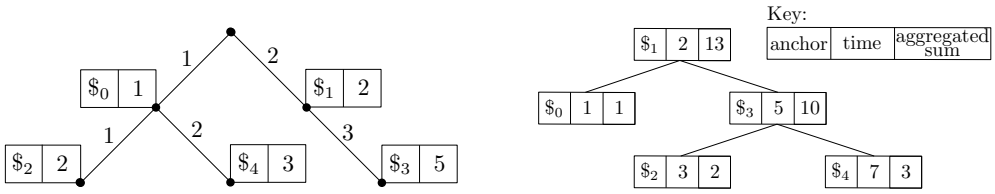

Figure 1: Left: LZ78 trie annotated with associated anchor points and word sums for a single sensor with observation string "12112312". Considered inline, the string with anchor points as breaks between the words becomes 1 $\$_0$ 2 $\$_1$ 11 $\$_2$ 23 $\$_3$ 12 $\$_4$. Right: The corresponding binary search tree based on word start times that also contains aggregated sums for the words contained in each node's subtree.

Building a binary search tree over the anchor points (stored already sorted by word start time) requires $O(d)$ time, since there are d words and each has one associated anchor point. Additionally, we create an aggregation tree over the aggregate word values, so that aggregate values of consecutive words can be easily found when considering substrings. This takes time $O(d)$ when created as an annotation to the existing binary search tree. Finally, we will later need access to a level ancestor data structure, which can be built in $O(d)$ time [5].

Lemma 3.1. *Assuming that the input is given in compressed form, temporal range searching takes preprocessing time* $O(d) = O(\mathrm{Enc}(X))$.

Next we describe query processing. Each temporal query can be categorized as either *internal* or *overlapping* depending on whether the query interval overlaps one word or multiple words, respectively. Internal queries implicitly divide a word into a prefix, query region, and suffix.

Since the trie is prefix-complete, all prefix aggregations are stored in our annotated trie and can be retrieved in $O(1)$ time using these annotations and the level-ancestor data structure. Entire word aggregate values can be retrieved as a group using the annotated binary search tree created over the aggregate word values. Using these basic retrieval systems, internal queries can be found by subtracting the prefix and suffix values from the word total and overlapping queries can be determined by adding the suffix, complete word sums, and prefix values.

The running time is dominated by the $O(\log d)$ time needed to lookup which word(s) overlap the given temporal query using a binary search over the sorted anchor points, and the $O(\log d)$ complete words that might be summed using the aggregation tree for overlapping queries. (The proof will appear in the full version.)

Lemma 3.2. *The query time for temporal range searching in the group setting is* $O(\log d) = O(\log T)$.

Finally, we consider the total number of bits of space used in this process. The storage of the anchor points requires space d and the annotated dictionary takes space d. Under our assumption that the group is of fixed size, the largest sum that can be achieved during this process is $O(T)$. These sums annotate dictionary words, so the modified dictionary takes space at most $O(d \log T)$, which is $O(d \log d)$ since $T = O(d^2)$. In addition, we make use of an auxiliary data structure to solve the level ancestor problem [5]. This data structure requires storage only of the tree, $O(d)$ pointers to nodes in the tree, and a table of $O(d)$ encoded subtrees that each take $O(\log d)$ space. Thus, the total size required by this auxiliary data structure is also $O(d \log d)$.

Lemma 3.3. *The total space in bits required for our temporal range structure in the group setting is* $O(d \log d) = O(\mathrm{Enc}(X))$.

4 Spatio-temporal Range Searching

In this section we consider how to extend the results of the previous section on temporal range searching on a single string to range searching for a sensor system, in which queries include both the spatial and

temporal components of the data. We assume that we are given an m-local sensor system with S sensors. Each sensor is identified with its location p_i in space and a stream X_i of occupancy counts over some common time interval $[1, T]$. We assume that the sensors reside in real d-dimensional space, $\mathbb{R}^d$, where d is a constant. Our approach can be generalized to metric spaces with constant doubling dimension. We model each sensor's location as a point, and the answer to a range query consists of the sensors whose associated point lies within the query region. Let P and $\mathbf{X}$ denote the sets of sensor locations and observation streams, respectively.

Define a *spatio-temporal range query* to be a pair $(Q, [t_0, t_1])$ consisting of a geometric query range Q from some space $\mathcal{Q}$ of allowable ranges (e.g., rectangles, balls, or halfspaces) and a time interval $[t_0, t_1] \subseteq [1, T]$. The problem is to compute the sum of the occupancy counts of the sensors whose locations lie within the range, that is, $P \cap Q$, over the given time interval. In general, the occupancy counts are assumed to be drawn from a commutative semigroup, and the sum is taken over this semigroup. The remainder of this section is devoted to proving the following theorem, which shows that approximate spherical spatio-temporal range queries can be answered efficiently. In fact, these techniques hold for all fat convex ranges, but for simplicity of presentation we will limit ourselves to the spherical case here. (Proofs will be found in the full version.)

Theorem 4.1. *There exists a data structure for answering ε-approximate spatio-temporal spherical range queries for an S-element m-local sensor system $\mathbf{X}$ in $\mathbb{R}^d$ for all sufficiently long time intervals T with preprocessing time $O(\mathrm{Enc}(\mathbf{X}))$, query time $O(((1/\varepsilon^{d-1}) + \log S) \log T)$, and space $O(\mathrm{Enc}(\mathbf{X}) \log S)$ bits.*

Rather than considering a particular range searching problem, we will show that the above problem can be reduced to a generalization of classical range searching. To motivate this reduction, we recall that the compression algorithm presented in [12] groups sensors into clusters, and the sensor outputs within each cluster are then compressed jointly. In order to answer range queries efficiently, it will be necessary to classify each such cluster as lying entirely inside the range, outside the range, or overlapping the range's boundary. In the last case, we will need to further investigate the cluster's internal structure. Efficiency therefore is dependent on the number of clusters that overlap the range's boundary. We will exploit spatial properties of the clusters as defined in [12] to achieve this efficiency. To encapsulate this notion abstractly, we introduce the problem of *range searching over clumps*, in which the points are replaced by balls having certain separation properties. Eventually, we will show how to adapt the BBD-tree structure [4] to answer approximate range queries in this context.

Given any metric space of constant dimension, a *set of clumps* is defined to be a finite set C of balls that satisfies the following *packing property* for some constant γ (depending possibly on dimension): Given any metric ball b of radius r, the number of clumps of C of radius r' that have a nonempty intersection with b is at most $O((1 + (r/r'))^\gamma)$. Given a range shape Q, a clump may either lie entirely within Q, entirely outside Q, or may intersect the boundary of Q. In the last case, we say that the clump is *stabbed* by Q.

The relevance of the notion of clumps to our setting is established in the following lemma. The lemma states that the clusters of sensors within a single partition created by the *PartitionCompress* algorithm of [12], when associated with a bounding ball, form a set of clumps. The *PartitionCompress* algorithm partitions the sensor point set P into a constant number of *groups*, $P_1, \ldots, P_c$ (where c depends only on the dimension of the space). Each group P_i is further partitioned into subsets, called *clusters*, such that if two sensors are in different clusters then their outputs are independent of each other. Given a ball b and real $\varphi > 0$, let $\varphi\, b$ denote the ball concentric with b whose radius is a factor of φ times the radius of b.

Lemma 4.1. *Given a point set P, let $P' \subseteq P$ be any of the groups generated by the* PartitionCompress *algorithm, and let $P'_1, \ldots, P'_h$ denote the associated set of clusters for this group. Then there exists a set of balls $C = \{b_1, \ldots, b_h\}$ that form a set of clumps such that $P'_i \subseteq b_i$.*

The proof of this lemma (given in the full version) relies on the observation that $\frac{1}{2}b_1, ..., \frac{1}{2}b_h$ are pairwise disjoint. This is established based on the geometric properties of the repetitive partitioning process of the *PartitionCompress* algorithm.

We define the problem of *range searching among clumps* as follows: Given a space $\mathcal{Q}$ of allowable ranges and a set C of clumps, each of which is associated with a numeric count from some commutative

semigroup, preprocess the clumps into a collection of subsets, called *generators*, such that given any query range $Q \in \mathcal{Q}$, it is possible to report (1) a subset of these generators that form a disjoint cover of the clumps lying wholly within Q and (2) the subset of clumps that Q stabs. The total space requirements of a data structure for the range searching problem over clumps is the sum of space needed to represent the generators and the clumps, together with the space needed for storing the index structure needed to answer queries. The query time includes number of generators and stabbed clumps returned, plus the time to compute them.

Many data structures used in range searching are based on partition trees [1]. In such data structures, space is recursively subdivided into regions and the points are partitioned among these regions, until each region contains a single point. Each node of the tree is associated with a generator corresponding to the elements of the point set that lie in the leaves descended from this node. Our main result shows that, given a partition-tree based solution to the problem of range-searching among clumps, we can use such a structure to answer spatio-temporal range queries. This is done by adding an auxiliary data structure to each of the nodes of the tree to answer the temporal queries.

Lemma 4.2. *Suppose that we have a partition-tree based data structure that, given a set C of n clumps, can answer range queries over a query space $\mathcal{Q}$ with preprocessing time $pp(n)$, query time $qt(n)$, space $sp(n)$ bits, and has height $h(n)$. Then there exists a data structure that can answer spatio-temportal range queries for an m-local sensor system $\mathbf{X}$ of size S over a range space $\mathcal{Q}$ and time interval of length T with preprocessing time $O(h(S) \cdot pp(S) + \mathrm{Enc}(\mathbf{X}))$, query time $O(qt(S) \cdot \log T)$, and space $O(sp(S) + h(S) \cdot \mathrm{Enc}(\mathbf{X}))$ bits.*

Observe that we can generalize the notion of ε-approximate range searching to approximate range searching over clumps. To do so we define two ranges Q^- and Q^+, representing the inner and outer approximate ranges. For example, in the case of spherical range searching, given a query ball Q, we define $Q^- = Q$ and Q^+ to be the ball concentric with Q but whose radius is scaled relative to Q's radius by a factor of $(1 + \varepsilon)$. (See Arya and Mount [4] for further details.) If a generator lies entirely within Q^+ its points may be counted as lying within the approximate range, if it lies entirely outside of Q^-, its points may be considered to lie outside the approximate range. A clump is classified as being stabbed by Q if and only if it has a nonempty intersection with both Q^- and the complement of Q^+. It is easy to show that such a clump has diameter $\Omega(\varepsilon \cdot \mathrm{diam}(Q))$ [4]. By the packing property of clumps, the number of such clumps is $O(1/\varepsilon^\gamma)$, where the parameter γ depends only on the dimension of the space. We conclude by remarking that it is relatively easy to generalize many standard approximate range searching data structures based on hierarchical partitioning to answer range searching over clumps. We present one example based on the BBD-tree data structure of [4].

Lemma 4.3. *There exists a data structure for answering ε-approximate spherical range searching queries over a set C of n clumps in $\mathbb{R}^d$ with preprocessing time $O(n \log n)$, query time $O((1/\varepsilon^{d-1}) + \log n)$, and space $O(n \cdot (prec(C) + \log n))$ bits, where $prec(C)$ denotes the maximum number of bits of precision in the geometric coordinates used to define C.*

By applying Lemma 4.2 to the above data structure, it follows that we can answer ε-approximate spherical range searching queries for a sensor system of size S over a time period of length T with preprocessing time $O((S \log^2 S) + \mathrm{Enc}(\mathbf{X}))$, query time $O(((1/\varepsilon^{d-1}) + \log S) \log T)$, and space $O((S \cdot (\mathrm{prec}(C) + \log S) + \mathrm{Enc}(\mathbf{X}) \log S)$ bits, where $\mathrm{prec}(C)$ denotes the maximum number of bits of precision in the geometric coordinates used to define C. Under the assumption that T is sufficiently large that the encoding space dominates over time-invariant quantities, this completes the proof of Theorem 4.1.

5 Experimental Results

In addition to the theoretical analysis of the range searching results presented here, we evaluated the temporal range searching structure experimentally. Using a data set provided by the Mitsubishi Electronic Research Laboratory (MERL) [34] consisting of activation times for sensors located in the hallways of their building, we analyzed three aspects of our data structure's performance; locality, space, and time. In short, we found that our assumption that sensors closer to each other are more likely to have similar outputs was correct and that our data structure was able to use less space than a naive structure while

providing faster query times. In the rest of this section, we describe the data set and our experimental methods in greater detail.

The MERL data set consists of activation times, representing people moving, for 213 sensors. These activation times are given with epoch start and end times. Using these start times, and noting that each activation lasted approximately one second, we translated these activation times into streams of data for each sensor in the form described earlier. Each activation is represented by a count of one and seconds in which no activations were reported are represented by a count of zero. These streams are associated with sensors whose locations are known and relationships are shown in a map of the building. Using this map, we create a graph representation in which neighboring sensors are connected by an edge with weight one.

In order to evaluate our assumption that nearby sensors are more likely to have related outputs, we compared the distance between a single sensor (sensor 369, a sensor in the middle of a hallway) to the pairwise joint entropy of that sensor and all other sensors' outputs. Distance was computed as the shortest path distance within the neighborhood graph described earlier, and the joint entropy considered was an empirical generalization of joint entropy [13] with a window size of 10 seconds. The average joint entropy for each distance is shown in Figure 2. The graph shows that those sensors in the neighborhood near sensor 369, those less than distance five away, have outputs with lower joint entropy. After this local neighborhood, the joint entropy raises to a relatively constant threshold for the majority of distances, and finally raises again for far away sensors. The outlying points at distances 8 and 18 represent comparisons with sensors in the unusual areas near the elevators and lunch room, respectively.

We considered the storage space over all sensor data streams for the raw data (consisting of one value per second) versus the temporal range structure (specifically the annotated trie) written to a file. The size taken over all sensors by each of these methods as it varies based on the number of days (in increments of 10) of data can be seen in Figure 3. We call the ratio between the space used by the raw data and the space used by the temporal range structure the *improvement ratio*. The improvement ratio increases as the amount of data increases, ranging from a 14-fold improvement for 1 day to a 66-fold improvement for 80 days of data. This increase is likely caused by the observation of repeated patterns; the first observation must be stored in the annotated trie while later observations can simply extend existing patterns, taking less space.

Query time is considered for varying query interval lengths for 150 days of data (at 1 day intervals). We compare our temporal range searching method to the naive method that aggregates by linearly adding each count. Query times do not include the time to read in the file or, in the case of the temporal range structure, the one-time preprocessing cost. A graph showing the interval length versus the query time for each of these methods is given in Figure 3. Each query time depicted on the graph represents the average of 100 randomly chosen queries of the given interval length. As the interval length increases, the temporal range structure's improvement over the naive method increases as well.

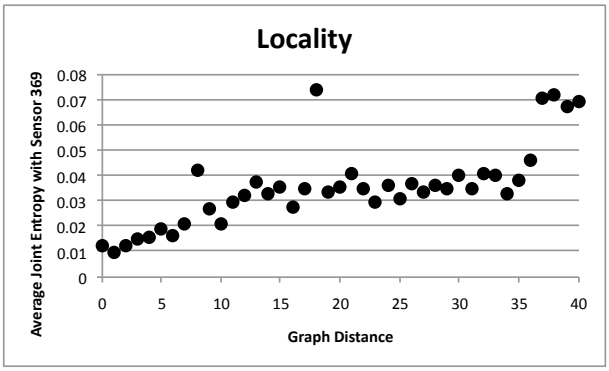

Figure 2: Locality shown via a comparison of the graph distance between sensor 369 and other sensors and the average joint entropy between all such pairs of sensors. As the distance between the sensors increases so does the joint entropy, showing that closer sensors are more likely to have related outputs.

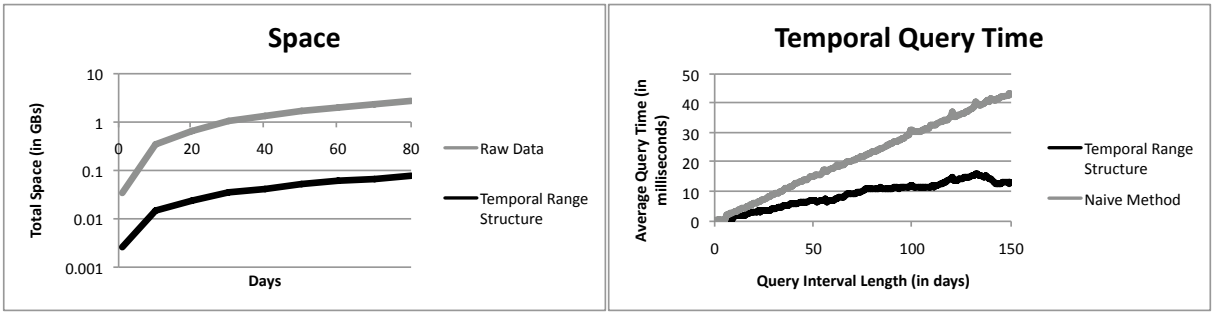

Figure 3: *Left:* The space used by the raw data in comparison to that used by the temporal range structure shown for varying numbers of days of data. Note that the size is shown in a logarithmic scale. As the number of days increase, the space saving increases as well. *Right:* Average temporal query times for 100 randomly chosen queries for each interval length. Query times for two methods are shown; one using the temporal range structure and one using a naive method that simply aggregates values one by one.

References

[1] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, pages 1–56. American Mathematical Society, 1998.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. In *Computer Networks*, pages 393–422, 2002.

[3] A. Amir, G. Benson, and M. Farach-Colton. Let sleeping files lie: Pattern matching in Z-compressed files. *J. Comput. Syst. Sci.*, 52(2):299–307, April 1996.

[4] S. Arya and D. M. Mount. Approximate range searching. *Computational Geometry: Theory and Applications*, 17:135–152, 2000.

[5] M. A. Bender and M. Farach-Colton. The level ancestor problem simplified. *Theoretical Computer Science*, 321:5–12, 2004.

[6] California Department of Transportation. Traffic data branch. http://traffic-counts.dot.ca.gov/.

[7] S.-Y. Chiu, W.-K. Hon, R. Shah, and J. S. Vitter. I/O-efficient compressed text indexes: From theory to practice. In *Data Compression Conference*, pages 426–434, 2010.

[8] H. Dettki, G. Ericsson, and L. Edenius. Real-time moose tracking: an internet based mapping application using GPS/ GSM-collars. *Alces*, 40:13–21, 2004.

[9] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and searching XML data via two zips. In *Proc. of the 15th Intl. Conference on World Wide Web*, pages 751–760, 2006.

[10] P. Ferragina and G. Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, July 2005.

[11] P. Ferragina and R. Venturini. A simple storage scheme for strings achieving entropy bounds. *Theoretical Computer Science*, 372(1):115–121, March 2007.

[12] S. A. Friedler and D. M. Mount. Compressing kinetic data from sensor networks. In *Proc. of the Fifth Intl. Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors)*, pages 191 – 202, 2009.

[13] S. A. Friedler and D. M. Mount. Realistic compression of kinetic sensor data. Technical Report CS-TR-4956, University of Maryland, College Park, 2010.

[14] S. Gandhi, R. Kumar, and S. Suri. Target counting under minimal sensing: Complexity and approximations. In S. Fekete, editor, *AlgoSensors*, pages 30–42, 2008.

[15] R. González and G. Navarro. Statistical encoding of succinct data structures. *Combinatorial Pattern Matching*, pages 294–305, 2006.

[16] A. Guitton, N. Trigoni, and S. Helmer. Fault-tolerant compression algorithms for sensor networks with unreliable links. Technical Report BBKCS-08-01, Birkbeck, University of London, 2008.

[17] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40, Sept. 1952.

[18] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. Rubenstein. Power-aware computing for wildlife tracking: Design tradeoffs and early experiences in zebranet. In *10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 96–107, October 2002.

[19] R. S. Kosaraju and G. Manzini. Compression of low entropy strings with Lempel–Ziv algorithms. *SIAM J. Comput.*, 29(3):893–911, 1999.

[20] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *Symposium on Discrete Algorithms*, 2004.

[21] G. Manzini. An analysis of the Burrows–Wheeler transform. *J. ACM*, 48(3):407–430, May 2001.

[22] Minnesota Department of Transportation. Mn/dot traveler information. http://www.dot.state.mn.us/tmc/trafficinfo/developers.html.

[23] MIT Media Lab. The Owl project. `http://owlproject.media.mit.edu/`.

[24] PORTAL. Portlan oregon regional transportation archive listing. http://portal.its.pdx.edu/.

[25] POST. Pacific ocean shelf tracking project. http://www.postcoml.org/.

[26] O. Procopiuc, P. K. Agarwal, L. Arge, and J. S. Vitter. A dynamic scalable kd-tree. In *Proc. International Symposium on Spatial and Temporal Databases*, volume LNCS 2750, 2003.

[27] J. Rissanen. Generalized Kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20, 1976.

[28] J. Robinson. The K-D-B tree: A search structure for large multidimensional dynamic indexes. In *Proc. SIGMOD International Conference on Management of Data*, pages 10–18, 1981.

[29] N. Saunier and T. Sayed. Automated analysis of road safety with video data. In *Transportation Research Record*, pages 57–64, 2007.

[30] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.

[31] B. J. M. Stutchbury, S. A. Tarof, T. Done, E. Gow, P. M. Kramer, J. Tautin, J. W. Fox, and V. Afanasyev. Tracking long-distance songbird migration by using geolocators. *Science*, page 896, February 2009.

[32] B. J. M. Stutchbury, S. A. Tarof, T. Done, E. Gow, P. M. Kramer, J. Tautin, J. W. Fox, and V. Afanasyev. Tracking long-distance songbird migration by using geolocators. *Science*, page 896, February 2009.

[33] TeraFlow Testbed. Pantheon highway gateway. http://www.teraflowtestbed.net/demos/hgateway.html.

[34] C. R. Wren, Y. A. Ivanov, D. Leigh, and J. Westbues. The MERL motion detector dataset: 2007 workshop on massive datasets. Technical Report TR2007-069, Mitsubishi Electric Research Laboratories, Cambridge, MA, USA, August 2007.

[35] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3), May 1977.

[36] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.