

# Spatio-temporal Range Searching Over Compressed Kinetic Sensor Data

Sorelle A. Friedler\* and David M. Mount\*\*

Dept. of Computer Science, University of Maryland, College Park, MD 20742, USA  
sorelle@cs.umd.edu                      mount@cs.umd.edu  
<http://www.cs.umd.edu/~sorelle>    <http://www.cs.umd.edu/~mount>

**Abstract.** As sensor networks increase in size and number, efficient techniques are required to process the data arising from them. 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, which operates on a compressed representation of the data, without the need to decompress it. We analyze the efficiency of our algorithm in terms of two natural measures of information content, the statistical and empirical joint entropies 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. These results represent the first solution to range searching problems over compressed kinetic sensor data.

## 1 Introduction

Sensor networks and the data they collect have become increasingly prevalent. Sensor networks are frequently employed to observe objects in motion and are used to record traffic data [12, 20], observe wildlife migration patterns [17, 23], 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 [12]. 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 work of Sorelle Friedler has been supported in part by the AT&T Labs Fellowship Program and the University of Maryland 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

the compressed data without decompressing it. Before more sophisticated statistical analyses of the data may be performed, retrieval queries must be supported. In this paper, we present the first range searching queries operating over compressed kinetic sensor data.

In an earlier paper [10], we presented an algorithm for losslessly compressing kinetic sensor data and a framework for analyzing its performance. (This framework and compression algorithm are discussed in more detail in Section 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 [15].) 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. Over time, each sensor produces a string of occupancy counts, and the problem considered in [10] is how to compress all these strings.

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. Lossless compression algorithms have been studied in the single-string setting [13, 19, 24, 25] but remain mostly unstudied in a sensor-based setting [10].

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. *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. *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, 16].

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 [21] 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, 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 Section 3 and Appendix A.

Previous retrieval over compressed text (without relying on decompression) has been studied in the context of strings [3, 9, 11, 18] and XML files [8]. For example, Ferragina and Manzini [9] show that it is possible to retrieve all occurrences of a given pattern in the compressed text with query time equal to the number of occurrences plus the length of the pattern. Their space requirement is  $5T \cdot H_k(X) + o(T)$  bits for a string  $X$  of length  $T$ . However, their data structure allows substring queries, which are very different from semigroup range searching queries, which we consider here.

In this paper we present the first range query results over compressed kinetic sensor data. In addition we generalize the analysis of our previously presented compression algorithm [10] to the empirical context. We analyze the range query results in both a statistical and an empirical context. 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, show that we achieve these results without decompressing the data. Specific bounds are given in Table 1.

---

**Table 1.** Time and space bounds for temporal range searching and  $\varepsilon$ -approximate spherical spatio-temporal range searching 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.

<b>Bounds for Range Searching</b>		
	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	$O(\text{Enc}(X))$	$O(\text{Enc}(\mathbf{X}) \log S)$

## 2 Framework for Kinetic Sensor Data

In an earlier paper [10] we introduced a framework and an 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).

Central to our framework is the notion that each sensor's output is statistically dependent 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  $p_i \in NN_m(j)$  and  $p_j \in NN_m(i)$ . 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.

In [10] we introduced a compression algorithm, *PartitionCompress*, which operates on an  $m$ -local sensor system. (The algorithm is sketched in Section C.) It compresses the sensor outputs to within a constant factor  $c$  (depending on dimension) of the optimal joint entropy bound. 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  $c$  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 much compress streams to their optimal entropy bound. In Appendix A we show that LZ78, the Lempel-Ziv dictionary compression algorithm [25], is sufficient for our purposes.

The results of [10] are presented in terms of statistical entropy, but in the next section we generalize the results to the case of empirical entropy.

### 3 Entropy and Independence

As mentioned in the introduction, the results of this paper apply (with minor variations) to two well known measures of information content, statistical entropy and empirical entropy. While the statistical approach is better known and allows for a simpler presentation, it requires the strong assumption that the stream of characters is generated by an stationary, ergodic random process, and performance bounds hold only in the limit as the string's length approaches infinity. In contrast, the empirical approach is based purely on properties of the given string, and requires no such assumptions. The results of [10] were proven in the context of statistical entropy and rely on statistical concepts, such as joint probability and statistical independence. In this section we show that these notions can be adapted to the empirical setting. (Due to space limitations, technical details have been moved to Appendix A.)

Consider a set of streams  $\mathbf{X} = \{X_1, X_2, \dots, X_Z\}$ , each of length  $T$ , and let  $S_{opt}(\mathbf{X}) = T \cdot H_k(\mathbf{X})$ , where  $H_k(\mathbf{X})$  is the empirical entropy (see [9] and Appendix A). We generalize a result of Ferragina and Manzini by showing that  $S_{opt}(\mathbf{X})$  is a lower bound on the size of a compressed string, assuming an encoder in which each output depends only on the current symbol and the  $k$  immediately preceding symbols. We also introduce the notions of (exact and approximate) empirical and statistical independence and introduce the notion of a  $(\delta, m)$ -local sensor system. 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 [10]. Since the LZ78 algorithm will suffice for our purposes, let  $\text{Enc}(\mathbf{X}) = \text{Enc}_{LZ78}(\mathbf{X})$ . We summarize the principal results of Appendix A below. These state that the encoding size of a sensor system under the previously introduced framework is on the order of the lower bound  $S_{opt}(\mathbf{X})$  in both the statistical and empirical contexts, where the lower bound  $S_{opt}(\mathbf{X})$  has been generalized to  $S_{opt}(\mathbf{X}, \delta)$  to account for our notion of limited independence.

**Theorem 1.** *Given a set  $\mathbf{X}$  of sensor outputs from a statistically  $(\delta, m)$ -local sensor system, for any  $0 \leq \delta < 1 - \Omega(1)$ ,*

$$\text{Enc}(\mathbf{X}) = O(\max\{\delta T, S_{opt}(\mathbf{X}, \delta)\}) \text{ bits.}$$

**Theorem 2.** *Given a set  $\mathbf{X} = \{X_1, X_2, \dots, X_Z\}$  of sensor outputs taken over a sufficiently long time  $T$  from an empirically  $(\delta, m)$ -local sensor*

system, for any  $0 \leq \delta < 1 - \Omega(1)$ ,

$$\begin{aligned} \text{Enc}(\mathbf{X}) &= cT \sum_{i=1}^Z \left( H_k(X_i) + O\left(\frac{\log \log T}{\log T}\right) \right) \\ &= O\left(\max\left\{\delta T, S_{opt}(X, \delta), \frac{T \log \log T}{\log T}\right\}\right) \text{ bits.} \end{aligned}$$

## 4 Temporal Range Searching

In this section we describe a data structure that answers temporal range searching queries over a single compressed sensor stream. Recall that we are given a sequence  $X$  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.** *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. In the next section we consider the simpler group case, where both addition and subtraction are allowed, and in Section 4.2, we consider the semigroup case, where only addition is allowed. We also make use of the fact, proved in Appendix A, that  $d \log d = \text{Enc}(\mathbf{X})$ .

### 4.1 Group Setting

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 creating the standard accompanying trie (also known as a *dictionary*) containing nodes that represent *words* [25]. 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* [9].

The compressed text is modified to be of the form  $W_1\$W_2\$ \dots W_d\$$  where  $\{W_1, \dots, W_d\}$  are the words in the dictionary and  $\$$  is a character not in the original alphabet. Each  $\$$  is associated with the  $W_i$  preceding it, and the location of that  $\$$ , or *anchor point*, is added as an annotation to each dictionary word. (These manipulations were introduced by Ferragina and Manzini [9], and though pointers to the beginning of words would suffice for our application, we use the insertion of  $\$$ 's for notational convenience.) This process requires one scan through the input, which takes time  $O(T)$ . Since each dictionary word appears exactly once in the compressed text, each word has a single associated anchor point. Answering queries as described below will require access to a sorted list of anchor points. There are  $d$  such anchor points, which take  $O(d \log d)$  time to sort.

The next preprocessing step is to precompute the sum of the counts within each word. These totals are added to the associated nodes in the dictionary by starting from the top of the trie and iteratively adding the count at the current node to its parent's sum and storing it at the current node (see Figure B.1). This takes time on the order of the number of words in the dictionary, or  $d$ .

**Lemma 1.** *Assuming that the input is given in compressed form, temporal range searching takes preprocessing time  $O(d \log d) = O(\text{Enc}(X))$ .*

Next we describe query processing. When given a temporal range  $[t_0, t_1]$ , the first step is to locate the anchor points  $\$_0$  and  $\$_1$  such that  $\$_0 \leq t_0$  and  $\$_1 \geq t_1$ , and there are no other  $\$'_0$  or  $\$'_1$  such that  $\$_0 < \$'_0 \leq t_0$  and  $\$_1 > \$'_1 \geq t_1$ . This is performed by a binary search through the sorted list of anchor points, which takes time  $O(\log d)$ . We say that the result is *overlapping*, if there exists some anchor between  $\$_0$  and  $\$_1$  in the compressed text, and otherwise it is *internal*. We handle these as separate cases.

*Overlapping Case:* First sum the counts for all words that are completely contained within the given temporal range. There can be no more than  $d$  of these, so this summation takes at most  $d$  time. Next, the count of the suffix of the requested range, which is the prefix of the word that starts just after  $t_1$  and ends at  $\$_1$ , is added to the sum. By prefix-completeness, this prefix is stored on its own in the trie. The prefix count can be efficiently retrieved in  $O(1)$  time, given a pointer to the leaf node associated with  $\$_1$ , the length of the prefix, and the data structure of [5] for answering level-ancestor queries. (This data structure can be built in  $O(d)$  time and requires  $O(d \log d)$  bits).

Finally, the prefix of the requested range, which is the suffix of the word  $w_0$  beginning at  $s_0$ , is added to complete the sum. The suffix count is calculated by first looking up the prefix of  $w_0$  that ends just before  $t_0$ , and subtracting its count from  $w_0$ 's total count. This prefix count is computed exactly as in the previous paragraph.

*Internal Case:* The dictionary word is subdivided into three non-overlapping sections based on the range query; the prefix, the query region, and the suffix. Due to prefix-completeness, the count for the prefix is recorded in the annotated dictionary. It can be retrieved in  $O(1)$  time, as above, using the level ancestor algorithm [5]. Similarly, the count for the word resulting from the concatenation of the prefix and the query region is also in the dictionary and can be retrieved in  $O(1)$  time. Subtracting this count from the total word count results in the count for the suffix. Subtracting the suffix and prefix counts from the total word count gives the count for the query region, as desired.

The query time, once given a specific temporal range, is  $O(d + \log d)$ . In order to reduce the query time, we supplement the data structure for the overlapping case so that  $d$  words are never summed individually, but rather are looked up in an aggregation tree (of size  $O(d)$ ) from which we use the largest component subtrees. Using this data structure, the number of summed subtrees is  $O(\log d)$ . Thus, the query time is reduced to  $O(\log d)$ .

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

## 4.2 Semigroup Setting

The results from the previous section hold only for group operations. Specifically, they do not hold for queries such as “max” and “min.” We generalize these results to the semigroup setting. In order to handle semigroup operations, for a substring in a given temporal range we need to be able to return the aggregated result in  $O(\log d)$  time without relying on subtraction. We base our auxiliary data structure on the link-cut tree [22]. We augment the link-cut tree to include the aggregated cost of the subpath represented by the node for each node in the binary trees associated with solid paths (see Figure B.2). With these additions, the data structure still takes space  $O(d \log d)$ . Further explanation and proof of the following lemma can be found in Appendix B.2.

**Lemma 4.** *The query time for temporal range searching in the semigroup setting is  $O(\log d)$ .*

## 5 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,  $\mathbb{R}^d$ , where  $d$  is a constant.

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. (Proofs of the lemmas presented here can be found in Appendix C.)

**Theorem 4.** *There exists a data structure for answering  $\varepsilon$ -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  $O(\text{Enc}(\mathbf{X}))$ , query time  $O((1/\varepsilon^{d-1}) + \log S) \log T$ , and space  $O(\text{Enc}(\mathbf{X}) \log S)$  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 [10] groups sensors into small 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 [10] 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 [4] structure to answer range queries in this context.

Given any metric space of constant dimension, a *set of clumps* is defined to be finite set  $C$  of balls that satisfies the following *packing property* for some constant  $\gamma$  (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  $O((1 + (r/r'))^\gamma)$ .

The relevance of the notion of clumps to our setting is established in the following lemma. We refer the reader to the *PartitionCompress* algorithm of [10]. This algorithm partitions the sensor point set  $P$  into a constant number of *groups*,  $P_1, \dots, P_c$  (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  $\varphi$  times the radius of  $b$ .

**Lemma 5.** *Given a point set  $P$ , let  $P' \subseteq P$  be any of the groups generated by the *PartitionCompress* algorithm, and let  $P'_1, \dots, P'_h$  denote the associated set of clusters for this group. Then there exists a set of balls  $C = \{b_1, \dots, b_h\}$  that form a set of clumps such that  $P'_i \subseteq b_i$ .*

In the appendix we define a natural generalization of range searching called *range searching among clumps*. The following lemma shows that any data structure for this problem can be used to answer spatio-temporal range queries.

**Lemma 6.** *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-temporal 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  $O(h(S) \cdot pp(S) + \text{Enc}(\mathbf{X}))$ , query time  $O(qt(S) \cdot \log T)$ , and space  $O(sp(S) + h(S) \cdot \text{Enc}(\mathbf{X}))$  bits.

Many data structures, such as the range searching algorithm appearing in [4] for approximate spherical range searching, exploit packing properties to achieve efficiency. Our next result shows that, through a straightforward adaptation of the algorithm presented there, it is possible to answer such queries in the context of clumps.

**Lemma 7.** *There exists a data structure for answering  $\varepsilon$ -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  $O(n \cdot (\text{prec}(C) + \log n))$  bits, where  $\text{prec}(C)$  denotes the maximum number of bits of precision in the geometric coordinates used to define  $C$ .*

By applying Lemma 6 to the above data structure, it follows that we can answer  $\varepsilon$ -approximate spherical range searching queries for a sensor system of size  $S$  over a time period of length  $T$  with preprocessing time  $O((S \log^2 S) + \text{Enc}(\mathbf{X}))$ , query time  $O(((1/\varepsilon^{d-1}) + \log S) \log T)$ , and space  $O((S \cdot (\text{prec}(C) + \log S) + \text{Enc}(\mathbf{X})) \log S)$  bits, where  $\text{prec}(C)$  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.

## 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.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. In *Computer Networks*, pages 393–422, 2002.
3. Amihood Amir, Gary Benson, and Martin Farach-Colton. Let sleeping files lie: pattern matching in Z-compressed files. *Journal of Computer and System Sciences*, 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. Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theoretical Computer Science*, 321:5–12, 2004.
6. T. M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-IEEE, second edition, 2006.
7. P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398, 2000.

8. Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and searching XML data via two zips. In *Proc. of the 15th International Conference on World Wide Web*, pages 751–760, 2006.
9. Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, July 2005.
10. Sorelle A. Friedler and David M. Mount. Compressing kinetic data from sensor networks. In *Proc. of the 5th International Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors)*, 2009. To appear.
11. Rodrigo González and Gonzalo Navarro. Statistical encoding of succinct data structures. *Combinatorial Pattern Matching*, pages 294–305, 2006.
12. 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.
13. D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40, Sept. 1952.
14. Rao S. Kosaraju and Giovanni Manzini. Compression of low entropy strings with lempel–ziv algorithms. *SIAM J. Comput.*, 29(3):893–911, 1999.
15. Robert Krauthgamer and James R. Lee. Navigating nets: Simple algorithms for proximity search. In *Symposium on Discrete Algorithms*, 2004.
16. Jirka Matousek. Geometric range searching. *Computing Surveys*, 26(4):422–461, 1994.
17. MIT Media Lab. The owl project. <http://owlproject.media.mit.edu/>.
18. Mihai Patrascu. Succincter. In *49th IEEE Symposium on Foundations of Computer Science*, 2008.
19. Jorma Rissanen. Generalized kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20, 1976.
20. Nicolas Saunier and Tarek Sayed. Automated analysis of road safety with video data. In *Transportation Research Record*, pages 57–64, 2007.
21. C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.
22. Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.
23. Bridget J. M. Stutchbury, Scott A. Tarof, Tyler Done, Elizabeth Gow, Patrick M. Kramer, John Tautin, James W. Fox, and Vsevolod Afanasyev. Tracking long-distance songbird migration by using geolocators. *Science*, page 896, February 2009.
24. Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3), May 1977.
25. Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. on Information Theory*, 24(5):530–536, 1978.

## A Entropy and Independence

### A.1 Statistical Setting

We begin by considering entropy and independence in the traditional statistical setting. In this setting, a sensor’s output stream is modeled by a stationary, ergodic random process  $X$  over an alphabet  $\Sigma$  of fixed size.

The *statistical probability*  $p(x)$  of some outcome  $x \in \Sigma$  is the probability associated with that outcome by the underlying random process. The *statistical entropy* of  $X$  is defined to be  $-\sum_{x \in \Sigma} p(x) \log p(x)$ . (Throughout, logs are taken base 2.) The *normalized statistical entropy* generalizes this to strings of increasing length:

$$H(X) = \lim_{k \rightarrow \infty} -\frac{1}{k} \sum_{x \in \Sigma^k} p(x) \log p(x).$$

This value represents the number of bits needed to encode a single character of the stream [6]. Unless otherwise specified, all references to *entropy* will mean normalized entropy. The *normalized joint statistical entropy* of two streams  $X$  and  $Y$  is defined to be

$$H(X, Y) = \lim_{k \rightarrow \infty} -\frac{1}{k} \sum_{x, y \in \Sigma^k} p(x, y) \log p(x, y),$$

where  $p(x, y)$  denotes the joint probability of both  $x$  and  $y$  occurring.

We say that two sensor streams  $X$  and  $Y$  are *statistically independent* if, for all  $k$  and any  $x, y \in \Sigma^k$ , we have  $p(x, y) = p(x)p(y)$ . If  $X$  and  $Y$  are statistically independent then  $H(X, Y) = H(X) + H(Y)$  [6]. The following technical result will be of later use.

**Lemma 8.** *Consider two sensor outputs  $X$  and  $Y$  over the same time period. Let  $X + Y$  denote the componentwise sum of these streams. Then  $H(X + Y) \leq H(X, Y) \leq H(X) + H(Y)$ .*

*Proof.* To prove the first inequality, let  $Z = X + Y$ , and observe that  $p(z) = \sum_{x+y=z} p(x, y)$ . Clearly, if  $x + y = z$ , then  $p(x, y) \leq p(z)$ . Thus,

$$\begin{aligned} H(X + Y) &= - \sum_z p(z) \log p(z) \leq - \sum_z \sum_{\substack{x, y \\ x+y=z}} p(x, y) \log p(x, y) \\ &= - \sum_{x, y} p(x, y) \log p(x, y) = H(X, Y). \end{aligned}$$

By basic properties of conditional entropy (see, e.g., [6]), we have

$$H(X, Y) = H(X) + X(Y|X) \leq H(X) + H(Y),$$

which establishes the second inequality. □

## A.2 Empirical Setting

Unlike statistical entropy, *empirical entropy* is based purely on the observed string, and does not assume an underlying random process. It replaces the probabilities of normalized entropy over substrings of length  $k$  by observed probabilities, conditioned on the value of the previous  $k$  characters. Let  $X$  be a string of length  $T$  over some alphabet  $\Sigma$  of fixed size. For  $k \geq 1$  and  $x \in \Sigma^k$ , let  $c_0(x)$  denote the number of times  $x$  appears in  $X$ , and let  $c(x)$  denote the number of times  $x$  appears without being the suffix of  $X$ . Let  $p_x(x) = c(x)/(T - k)$  denote the *observed probability* of  $x$  in  $X$ . (When  $X$  is clear from context, we will express this as  $p(x)$ .) Following the definitions of Kosaraju and Manzini [14], the 0th order empirical entropy of a string  $X$  is defined to be

$$H_0(X) = - \sum_{a \in \Sigma} p(a) \log p(a) = - \sum_{a \in \Sigma} \frac{c_0(a)}{T} \log \frac{c_0(a)}{T}.$$

For  $a \in \Sigma$ , let  $p_x(a|x) = c(xa)/c(x)$  denote the observed probability that  $a$  is the next character of  $X$  immediately following  $x$ . The  $k$ th order empirical entropy is defined to be

$$H_k(X) = - \frac{1}{T} \sum_{x \in \Sigma^k} c(x) \left[ \sum_{a \in \Sigma} p(a|x) \log p(a|x) \right].$$

It is known that  $T \cdot H_k(X)$  is a lower bound to the output size of any compressor that encodes each symbol with a code that only depends on the symbol itself and the  $k$  immediately preceding symbols [9].

In addition, we introduce some concepts that are analogous to those defined for the statistical entropy. Given two strings  $X, Y \in \Sigma^T$  and  $x, y \in \Sigma^k$ , define  $c(x, y)$  to be the count of the number of indices  $i$ ,  $1 \leq i \leq T - k$ , such that  $X[i \dots i + k - 1] = x$  and  $Y[i \dots i + k - 1] = y$ . Define  $p_{X, Y}(x, y) = c(x, y)/(T - k)$ . For  $a, b \in \Sigma$ , define  $p_{X, Y}(a, b|x, y) = c(xa, yb)/c(x, y)$  to be the observed probability of seeing  $a$  and  $b$  in  $X$  and  $Y$ , respectively, just after seeing  $x$  and  $y$ . The *joint empirical entropy* of  $X$  and  $Y$  is defined to be

$$H_k(X, Y) = - \frac{1}{T} \sum_{x, y \in \Sigma^k} c(x, y) \left[ \sum_{a, b \in \Sigma} p_{X, Y}(a, b|x, y) \log p(a, b|x, y) \right].$$

The joint empirical entropy of a set of strings  $\mathbf{X} = \{X_1, \dots, X_Z\}$  is defined analogously and is denoted  $H_k(\mathbf{X})$ .

We define the *empirical conditional entropy* of two strings  $X, Y \in \Sigma^T$  to be

$$H_k(X|Y) = -\frac{1}{T} \sum_{x,y \in \Sigma^k} c(x,y) \sum_{a,b \in \Sigma} p_{X,Y}(a,b|x,y) \log p_{X,Y}(x,a|y,b),$$

where we define  $p_{X,Y}(x,a|y,b) = p_{X,Y}(x,a|y,b) = p_{X,Y}(a,b|x,y)/p_Y(b|y)$  to be the probability that  $a$  directly follows  $x$  in  $X$  given that  $b$  directly follows  $y$  in  $Y$ . We also define the *empirical mutual information*

$$I_k(X;Y) = \frac{1}{T} \sum_{x,y \in \Sigma^k} c(x,y) \sum_{a,b \in \Sigma} p_{X,Y}(a,b|x,y) \log \frac{p_{X,Y}(a,b|x,y)}{p_X(a|x)p_Y(b|y)}.$$

We say that two strings  $X$  and  $Y$  are *empirically independent* if, for all  $j \leq k+1$  and all  $x, y \in \Sigma^j$ , the observed probability of  $x$  occurring at the same time instant as  $y$  is equal to the product of the observed probabilities of each outcome individually, that is,  $p_{X,Y}(x,y) = p_X(x)p_Y(y)$ . If  $X$  and  $Y$  are empirically independent then this also implies that, for  $a \in \Sigma$  and  $b \in \Sigma$ ,  $p_{X,Y}(a,b|x,y) = p_X(a|x)p_Y(b|y)$ .

The following technical lemma provides a few straightforward generalizations regarding properties of statistical entropy to empirical entropy.

**Lemma 9.** *Consider two strings  $X, Y \in \Sigma^T$ . Let  $X+Y$  denote the componentwise sum of these strings.*

- (i) *If  $X$  and  $Y$  are empirically independent,  $H_k(X,Y) = H_k(X) + H_k(Y)$ .*
- (ii)  *$H_k(X,Y) = H_k(X) + H_k(Y|X)$ .*
- (iii)  *$I_k(X;Y) = H_k(X) - H_k(X|Y)$ .*
- (iv)  *$H_k(X+Y) \leq H_k(X) + H_k(Y)$ .*

*Proof.* We will not prove (i) here, since it will follow as a special case of Lemma 11 below (by setting  $\delta = 0$ ). To prove (ii), observe that

$$\begin{aligned}
H_k(X, Y) &= -\frac{1}{T} \sum_{x, y \in \Sigma^k} c(x, y) \left[ \sum_{a, b \in \Sigma} p_{X, Y}(a, b|x, y) \log p_{X, Y}(a, b|x, y) \right] \\
&= -\frac{1}{T} \sum_{x, y \in \Sigma^k} c(x, y) \left[ \sum_{a, b \in \Sigma} p_{X, Y}(a, b|x, y) \log (p_X(a|x) p_{X, Y}(y, b|x, a)) \right] \\
&= -\frac{1}{T} \sum_{x, y \in \Sigma^k} c(x, y) \left[ \sum_{a, b \in \Sigma} p_{X, Y}(a, b|x, y) \log p_X(a|x) \right] + \\
&\quad -\frac{1}{T} \sum_{x, y \in \Sigma^k} c(x, y) \left[ \sum_{a, b \in \Sigma} p_{X, Y}(a, b|x, y) \log p_{X, Y}(y, b|x, a) \right] \\
&= H_k(X) + H_k(Y|X).
\end{aligned}$$

Symmetrically, we have  $H_k(X, Y) = H_k(Y) + H_k(X|Y)$ .

To prove (iii), observe that

$$\begin{aligned}
I_k(X; Y) &= \frac{1}{T} \sum_{x, y \in \Sigma^k} c(x, y) \sum_{a, b \in \Sigma} p_{X, Y}(a, b|x, y) \log \frac{p_{X, Y}(a, b|x, y)}{p_X(a|x) p_Y(b|y)} \\
&= \frac{1}{T} \sum_{x, y \in \Sigma^k} c(x, y) \sum_{a, b \in \Sigma} p_{X, Y}(a, b|x, y) \log \frac{p_{X, Y}(x, a|y, b)}{p_X(a|x)} \\
&= -\frac{1}{T} \sum_{x, y \in \Sigma^k} c(x, y) \sum_{a, b \in \Sigma} p_{X, Y}(a, b|x, y) \log p_X(a|x) + \\
&\quad \frac{1}{T} \sum_{x, y \in \Sigma^k} c(x, y) \sum_{a, b \in \Sigma} p_{X, Y}(a, b|x, y) \log p_{X, Y}(x, a|y, b) \\
&= -\frac{1}{T} \sum_{x, y \in \Sigma^k} c(x) \sum_{a, b \in \Sigma} p_X(a|x) \log p_X(a|x) - \\
&\quad \left( -\frac{1}{T} \sum_{x, y \in \Sigma^k} c(x, y) \sum_{a, b \in \Sigma} p_{X, Y}(a, b|x, y) \log p_{X, Y}(x, a|y, b) \right) \\
&= H_k(X) - H_k(X|Y).
\end{aligned}$$

Symmetrically, we have  $I_k(X; Y) = H_k(Y) - H_k(Y|X)$ . By (ii) we have  $I_k(X; Y) = H_k(X) + H_k(Y) - H_k(X, Y)$ . Since  $I_k(X; Y)$  is clearly non-negative, this implies that  $H_k(X, Y) \leq H_k(X) + H_k(Y)$ .

To prove (iv), let  $Z = X + Y$ . By the definition of empirical entropy we have  $H_k(X + Y) =$

$$-\frac{1}{T} \sum_{z \in \Sigma^k} \sum_{\substack{x, y \\ x+y=z}} c(x+y) \left[ \sum_{g \in \Sigma} \sum_{\substack{a, b \\ a+b=g}} p_z(a+b|x+y) \log p_z(a+b|x+y) \right],$$

where  $x+y$  is an outcome of length  $k$  and  $a+b$  is an outcome of length 1 in the new string  $X+Y$ . By the same reasoning as in Lemma 8,  $p_{X,Y}(x,y) \leq p_Z(x+y)$ . Substituting this relationship into our equation and, since we desire an upper bound, considering only cases in which  $-p_{X,Y}(a+b|x+y) \log p_{X,Y}(a+b|x+y) \leq -p_{X,Y}(a,b|x,y) \log(p_{X,Y}(a,b|x,y))$ , we find that

$$\begin{aligned} H_k(X+Y) &\leq -\frac{1}{T} \sum_{x+y \in \Sigma^k} c(x,y) \left[ \sum_{a,b \in \Sigma} p_{X,Y}(a,b|x,y) \log p_{X,Y}(a,b|x,y) \right] \\ &= H_k(X,Y). \end{aligned}$$

By (iii) we have  $H_k(X,Y) \leq H(X) + H(Y)$ , which implies that  $H_k(X+Y) \leq H(X) + H(Y)$ , as desired.  $\square$

### A.3 Limited Independence

Perfect statistical or empirical independence may be too strong an assumption to impose on sensor outputs. To deal with this, in this section we introduce a notion of limited independence for both the statistical and empirical settings. Given  $0 \leq \delta < 1$ , we say that a set of sensor streams  $\mathbf{X} = \{X_1, X_2, \dots, X_Z\}$  is *statistically  $\delta$ -independent* if, for any  $k$  and outcomes  $x_i \in \Sigma^k$ ,  $p(x_1)p(x_2)\dots p(x_Z)(1-\delta) \leq p(x_1, x_2, \dots, x_Z) \leq p(x_1)p(x_2)\dots p(x_Z)(1+\delta)$ . In the following lemma, we develop a relationship regarding the entropies of statistically  $\delta$ -independent streams.

**Lemma 10.** *Given  $0 \leq \delta < 1$  and a set of statistically  $\delta$ -independent streams  $\mathbf{X} = \{X_1, X_2, \dots, X_Z\}$ ,  $(1-\delta)(\sum_{i=1}^Z H(X_i)) - O(\delta) \leq H(\mathbf{X}) \leq (1+\delta)(\sum_{i=1}^Z H(X_i)) + O(\delta)$ .*

*Proof.* For simplicity of presentation, here we prove the lemma for sets  $\mathbf{X} = \{X, Y\}$ . The proof for a set of any size follows clearly from this presentation.

Recall that

$$H(X, Y) = \lim_{k \rightarrow \infty} -\frac{1}{k} \sum_{x \in X, y \in Y} p(x, y) \log p(x, y).$$

By the assumption of statistical  $\delta$ -independence, we have

$$\begin{aligned}
H(X, Y) &\leq \lim_{k \rightarrow \infty} -\frac{1}{k} \sum_{x \in X, y \in Y} p(x)p(y)(1 + \delta) \log(p(x, y)) \\
&= \lim_{k \rightarrow \infty} \frac{1}{k} \sum_{x \in X, y \in Y} p(x)p(y)(1 + \delta) \log \frac{1}{p(x, y)} \\
&\leq \lim_{k \rightarrow \infty} \frac{1}{k} \sum_{x \in X, y \in Y} p(x)p(y)(1 + \delta) \log \frac{1}{p(x)p(y)(1 - \delta)} \\
&= \lim_{k \rightarrow \infty} \frac{1 + \delta}{k} \left[ - \sum_{x \in X} (p(x) \log p(x)) - \sum_{y \in Y} (p(y) \log p(y)) + \log \frac{1}{1 - \delta} \right] \\
&= (1 + \delta)(H(X) + H(Y)) + \lim_{k \rightarrow \infty} \frac{1 + \delta}{k} \log \frac{1}{1 - \delta}.
\end{aligned}$$

By a Taylor expansion in the neighborhood of  $\delta = 0$ , we see that  $(1 + \delta) \log \frac{1}{1 - \delta} = O(\delta)$ , which yields  $H(X, Y) \leq (1 + \delta)(H(X) + H(Y)) + O(\delta)$ . The proof that  $(1 - \delta)(H(X) + H(Y)) - O(\delta)$  proceeds symmetrically.  $\square$

We also introduce the idea of limited independence in the context of empirical entropy. Given  $0 \leq \delta < 1$  a set of strings  $\{X_1, X_2, \dots, X_Z\}$  is *empirically  $\delta$ -independent* if, for all  $x_i \in \Sigma^j$  for  $j \leq k + 1$ ,  $p(x_1)p(x_2)\dots p(x_Z)(1 - \delta) \leq p(x_1, x_2, \dots, x_Z) \leq p(x_1)p(x_2)\dots p(x_Z)(1 + \delta)$ .

**Lemma 11.** *Given  $0 \leq \delta < 1$ , and a set of empirically  $\delta$ -independent strings  $\mathbf{X} = \{X_1, X_2, \dots, X_Z\}$  for  $X_i \in \Sigma^j$  where  $j \leq k + 1$ ,*

$$(1 - \delta) \sum_{i=1}^Z H_k(X_i) - O(\delta) \leq H_k(\mathbf{X}) \leq (1 + \delta) \sum_{i=1}^Z H_k(X_i) + O(\delta).$$

*Proof.* For simplicity of presentation, here we prove the lemma for sets  $\mathbf{X} = \{X, Y\}$ . The general case is a straightforward generalization.

$$H_k(X, Y) = -\frac{1}{T} \sum_{x, y \in \Sigma^k} c(x, y) \sum_{a, b \in \Sigma} p(a, b|x, y) \log p(a, b|x, y)$$

Since  $p(a, b|x, y) = c_0(xa, yb)/c(x, y)$  where  $c_0(xa, yb)$  is the number of times the string  $xa \in X$  appears at the same indices as  $yb \in Y$ , we have

$$\begin{aligned} H_k(X, Y) &= -\frac{1}{T} \sum_{x, y \in \Sigma^k} c(x, y) \sum_{a, b \in \Sigma} \frac{c_0(xa, yb)}{c(x, y)} \log p(a, b|x, y) \\ &= -\frac{1}{T} \sum_{x, y \in \Sigma^k} \sum_{a, b \in \Sigma} c_0(xa, yb) \log p(a, b|x, y) \\ &= -\frac{1}{T} \sum_{x, y \in \Sigma^k} \sum_{a, b \in \Sigma} \frac{c_0(xa, yb)(T-k)}{T-k} \log p(a, b|x, y). \end{aligned}$$

Since  $p(xa, yb) = \frac{c_0(xa, yb)}{T-k}$  and  $p(a, b|x, y) = \frac{c_0(xa, yb)}{c(x, y)}$  this is

$$H_k(X, Y) = -\frac{T-k}{T} \sum_{x, y \in \Sigma^k} \sum_{a, b \in \Sigma} p(xa, yb) \log \left( \frac{p(xa, yb)}{p(x, y)} \right).$$

Before proceeding with this analysis, we develop a useful relationship.

$$\begin{aligned} &-\frac{(T-k)}{T} \left[ \sum_{x, y \in \Sigma^k} p(x)p(y) \sum_{a, b \in \Sigma} p(a|x)p(b|y) \log(p(a|x)p(b|y)) \right] \\ &= -\frac{1}{T(T-k)} \left[ \sum_{x, y \in \Sigma^k} c(x)c(y) \sum_{a, b \in \Sigma} p(a|x)p(b|y) \log p(a|x) + \right. \\ &\quad \left. \sum_{x, y \in \Sigma^k} c(x)c(y) \sum_{a, b \in \Sigma} p(a|x)p(b|y) \log p(b|y) \right] \\ &= -\frac{1}{T} \left[ \sum_{x \in \Sigma^k} c(x) \sum_{y \in \Sigma^k} \frac{c(y)}{T-k} \sum_{b \in \Sigma} p(b|y) \sum_{a \in \Sigma} p(a|x) \log p(a|x) + \right. \\ &\quad \left. \sum_{x \in \Sigma^k} \frac{c(x)}{T-k} \sum_{y \in \Sigma^k} c(y) \sum_{a \in \Sigma} p(a|x) \sum_{b \in \Sigma} p(b|y) \log p(b|y) \right] \\ &= -\frac{1}{T} \left[ \sum_{x \in \Sigma^k} c(x) \sum_{a \in \Sigma} p(a|x) \log p(a|x) + \sum_{y \in \Sigma^k} c(y) \sum_{b \in \Sigma} p(b|y) \log p(b|y) \right] \\ &= H_k(X) + H_k(Y). \end{aligned}$$

Now we develop an upper bound on the earlier equation. Let  $f = -p(xa, yb) \log \frac{p(xa, yb)}{p(x, y)} = p(xa, yb) \log \frac{p(x, y)}{p(xa, yb)}$ . Then the equation we're try-

ing to bound is

$$\frac{T-k}{T} \sum_{x,y \in \Sigma^k} \sum_{a,b \in \Sigma} f,$$

where, by the definition of  $\delta$ -independence,

$$\begin{aligned} f &\leq (1+\delta)p(xa)p(yb) \log \left( \frac{p(x,y)}{p(xa,yb)} \right) \\ &\leq (1+\delta)p(xa)p(yb) \log \left( \frac{(1+\delta)p(x)p(y)}{(1-\delta)p(xa)p(yb)} \right). \end{aligned}$$

Since  $p(xa,yb) = p(a|x)p(x)p(b|y)p(y)$ , this is equal to

$$\begin{aligned} &(1+\delta)p(x)p(y)p(a|x)p(b|y) \log \left( \frac{(1+\delta)}{(1-\delta)p(a|x)p(b|y)} \right) \\ &= (1+\delta)p(x)p(y)p(a|x)p(b|y) \left( \log \frac{(1+\delta)}{(1-\delta)} - \log(p(a|x)p(b|y)) \right). \end{aligned}$$

Substituting back in for  $f$  and using our previously developed relationship, we have

$$\begin{aligned} H_k(X,Y) &\leq (1+\delta)(H_k(X) + H_k(Y)) + \\ &\quad \frac{(1+\delta)(T-k)}{T} \sum_{x,y \in \Sigma^k} p(x)p(y) \sum_{a,b \in \Sigma} p(a|x)p(b|y) \log \frac{1+\delta}{1-\delta} \\ &= (1+\delta)(H_k(X) + H_k(Y)) + \frac{(1+\delta)(T-k)}{T} \log \frac{1+\delta}{1-\delta}. \end{aligned}$$

Let  $g(\delta) = \log \frac{1+\delta}{1-\delta} = \log(1 + \frac{2\delta}{1-\delta})$ . Consider the Taylor expansion for  $g(\delta)$  in the neighborhood of  $\delta = 0$  (i.e., the Maclaurin series). The Maclaurin series for  $g(\delta)$  is within a constant factor of the expansion for  $\log(1/(1-\delta)) = \delta + \delta^2/2 + \delta^3/3 + O(\delta^4)$ . Since  $\delta < 1$  by definition,  $\delta^i > \delta^j$  for  $i < j$ , so  $\log(1/(1-\delta)) = O(\delta)$  and  $g(\delta) = O(\delta)$ . Substituting back into our main inequality, we have

$$\begin{aligned} H_k(X,Y) &\leq (1+\delta)(H_k(X) + H_k(Y)) + \frac{(1+\delta)(T-k)}{T} O(\delta) \\ &\leq (1+\delta)(H_k(X) + H_k(Y)) + O(\delta). \end{aligned}$$

The proof that

$$(1-\delta)(H_k(X) + H_k(Y)) - O(\delta) \leq H_k(X,Y)$$

proceeds symmetrically. □

#### A.4 Compression Space Bounds

In this subsection will consider the encoding size that can be achieved by *PartitionCompress* [10]. Recall that *PartitionCompress* relies on a compression algorithm as a subroutine; the compression bounds for this subroutine will impact the final encoding size achieved by *PartitionCompress*. We will analyze this size in both statistical and empirical settings. In either context, we will use  $\text{Enc}_{alg}(\mathbf{X})$  to denote the length of the encoded set of sensor outputs  $\mathbf{X}$ , where *alg* is the compression algorithm used by *PartitionCompress*.  $\text{Enc}_{alg}(\mathbf{X})$  will be the bound on which we build our analyses in later sections.

Given a set of streams  $\mathbf{X} = \{X_1, X_2, \dots, X_Z\}$  in a statistical setting, standard information theory results [6] tell us that the optimal encoded space is  $\sum_{i=1}^Z H(X_i)$  bits; call this  $S_{opt}(\mathbf{X})$ . From Section A.3, we know that the optimal space used by an encoded set of statistically  $\delta$ -independent streams  $\mathbf{X}$  is  $(1 - \delta) \left( \sum_{i=1}^Z H(X_i) \right) - O(\delta)$  bits; call this  $S_{opt}(\mathbf{X}, \delta)$ . Let *opt* be some compression algorithm that achieves the optimal statistical entropy encoding length, for example LZ78. We know from [10] that  $\text{Enc}_{opt}(\mathbf{X}) = O(H(\mathbf{X}))$  bits for a set of observations from an  $m$ -local sensor system, where the hidden constant is exponential in  $m$  and the doubling dimension. We define a *statistically  $(\delta, m)$ -local sensor system* to be the same as an  $m$ -local sensor system but with an assumption of  $\delta$ -independence between the clusters instead of pure independence. We have the following theorem regarding the space used by *PartitionCompress*:

**Theorem 1.** *Given a set  $\mathbf{X}$  of sensor outputs from a statistically  $(\delta, m)$ -local sensor system, for any  $0 \leq \delta < 1 - \Omega(1)$ ,*

$$\text{Enc}(\mathbf{X}) = O(\max\{\delta T, S_{opt}(\mathbf{X}, \delta)\}) \text{ bits.}$$

*Proof.* The optimal space bound is

$$S_{opt}(\mathbf{X}, \delta) = T(1 - \delta) \left( \sum_{i=1}^Z H(X_i) \right) - T \cdot O(\delta)$$

while *PartitionCompress* achieves a bound of

$$O(S_{opt}(\mathbf{X})) = O \left( T \cdot \sum_{i=1}^Z H(X_i) \right).$$

The ratio is

$$\frac{O\left(\sum_{i=1}^Z H(X_i)\right)}{(1-\delta)\left[\sum_{i=1}^Z H(X_i)\right] - O(\delta)}.$$

The rest of the proof proceeds similarly to the proof of Theorem 6, but for  $H(X_i)$  instead of  $H_k(X_i)$  and with an extra constant factor hidden in the final bound.  $\square$

The space established in Theorem 1 is the basic statistical encoded space bound on top of which we will build range searching structures in Sections 4 and 5. It hides constants that are exponential in  $m$  and the doubling dimension. As a direct consequence of Lemma 8 and Theorem 1 we have the following corollary:

**Corollary 1.** *Consider two sensor outputs  $X$  and  $Y$  over the same time period. Let  $X + Y$  denote the componentwise sum of these streams over some commutative semigroup. Then  $Enc_{opt}(X+Y) \leq Enc_{opt}(X) + Enc_{opt}(Y)$  in the statistical setting.*

In the rest of this section, we extend these results to the empirical setting. In order to reason about the empirically optimal space bound for a set of strings  $\mathbf{X}$ , consider the string  $X_{new}$  created from the original set of strings by letting the  $i$ th character of the new string, for  $1 \leq i \leq T$ , be equal to a new character created by concatenating the  $i$ th character of each string in the original set. As mentioned earlier, the new string's optimal encoded space bound is  $T \cdot H_k(X_{new})$ .

**Lemma 12.** *Given a set of strings  $\mathbf{X}$  and a string  $X_{new}$  created from  $\mathbf{X}$  as described above,  $H_k(X_{new}) = H_k(\mathbf{X})$ .*

*Proof.* Recall that the definition of joint empirical entropy is based on the observed probability that single characters occur in all strings at the same string index directly after specific substrings of length  $k$ . Observe that by the construction of  $X_{new}$ , simultaneous occurrences appear for the same indices at which a single combined character appears in  $X_{new}$ . This observation implies that if  $H_k(\mathbf{X})$  is restated to refer to the characters appearing in  $X_{new}$ ,  $H_k(X_{new}) = H_k(\mathbf{X})$  completing the proof.  $\square$

**Corollary 2.**  *$S_{opt}(\mathbf{X}) = T \cdot H_k(\mathbf{X})$  bits is the optimal space bound for an encoded set of strings assuming that each character depends only on the preceding  $k$  characters.*

We will rely on context to distinguish between  $S_{opt}(\mathbf{X})$  in statistical and empirical contexts. Although this construction suggests a compression procedure, it is impractical because in order to capture the repetitive nature of the strings in  $\mathbf{X}$ , the window size  $k$  would need to be large. Instead, we use the more local approach of *PartitionCompress*.

We define an *empirically  $m$ -local sensor system* and a *empirically  $(\delta, m)$ -local sensor system* to be analogous to the definition of an  $m$ -local sensor system, but with an assumption of empirical independence or empirical  $\delta$ -independence respectively, instead of statistical independence. The algorithm *PartitionCompress* relies on an entropy encoding algorithm as a subroutine. In the context of an empirical entropy based analysis it would be appropriate to use the data structure developed by Ferragina and Manzini [7] that gives an optimal space bound of  $O(T \cdot H_k(X_i)) + T \cdot o(1)$  as the subroutine that jointly compresses the streams from a single cluster where  $X_i$  is the merged stream for that single cluster. For Theorems 5 and 6 our goal is to develop a lower bound on the compression that can be achieved using *PartitionCompress* in an empirical setting, so instead of using a specific algorithm we use the bound of  $S_{opt}(\mathbf{X})$  discussed earlier and call the algorithm that achieves this bound *opt*. Assuming empirical independence of the set of strings  $\mathbf{X}$  from  $z$  separate clusters within a single partition, compressing these clusters separately achieves the optimal bound of  $S_{opt}(\mathbf{X}) = T \cdot \sum_{i=1}^z H_k(X_i)$  space for a single partition. As a direct consequence, we have the following theorem.

**Theorem 5.** *Given a set  $\mathbf{X}$  of sensor outputs from an empirically  $m$ -local sensor system,  $\text{Enc}_{opt}(\mathbf{X}) = O(S_{opt}(\mathbf{X}))$  bits.*

The hidden constants from Theorem 5 and for Theorems 6 and 2 are exponential in  $m$  and the doubling dimension. If we consider empirical  $\delta$ -independence, then the lower bound achieved by the compression algorithm (over  $Z$  total clusters in all partitions) remains  $O(T \cdot \sum_{i=1}^Z H_k(X_i))$ , but  $\sum_{i=1}^Z H_k(X_i) \neq H_k(\mathbf{X})$  and so an optimal algorithm may be able to reduce the bound due to the  $\delta$  dependence allowed. By application of Lemma 11, an optimal algorithm's bound is  $S_{opt}(\mathbf{X}, \delta) = T(1 - \delta) \left( \sum_{i=1}^Z H_k(X_i) \right) + T \cdot O(\delta)$ . We have the following theorem regarding the compressed size of the sensor outputs.

**Theorem 6.** *Given a set  $\mathbf{X}$  of sensor outputs from an empirically  $(\delta, m)$ -local sensor system for  $0 \leq \delta < 1 - \Omega(1)$ ,  $\text{Enc}_{opt}(\mathbf{X}) = O(\max\{\delta T, S_{opt}(\mathbf{X})\})$  bits.*

*Proof.* An optimal algorithm would compress each partition to take the most advantage of the dependence between clusters. It would achieve a space bound of

$$S_{opt}(\mathbf{X}, \delta) = T(1 - \delta) \left[ \sum_{i=1}^Z H_k(X_i) \right] - T \cdot O(\delta)$$

for each partition, while *PartitionCompress* compresses each partition to

$$S_{opt}(\mathbf{X}) = T \sum_{i=1}^Z H_k(X_i).$$

The ratio is

$$\frac{\sum_{i=1}^Z H_k(X_i)}{(1 - \delta) \left[ \sum_{i=1}^Z H_k(X_i) \right] - O(\delta)}.$$

Here we consider the two possible cases for the relationship of  $O(\delta)$  to  $\sum_{i=1}^Z H_k(X_i)$ :

1. *Case*  $O(\delta) \geq \sum_{i=1}^Z H_k(X_i)$ :

$$\frac{\sum_{i=1}^Z H_k(X_i)}{(1 - \delta) \left[ \sum_{i=1}^Z H_k(X_i) \right] - O(\delta)} = \left[ \frac{1}{1 - \delta} \right] O(\delta) = O(\delta).$$

For this case, *PartitionCompress*'s space bound is within  $O(\delta)$  of the optimal for a single one of the  $c$  partitions, or a total of  $O(\delta)$  times  $S_{opt}(\mathbf{X}, \delta)$ , which is  $O(\delta \cdot T)$ , since  $O(\delta) \geq \sum_{i=1}^Z H_k(X_i)$ .

2. *Case*  $O(\delta) < \sum_{i=1}^Z H_k(X_i)$ :

$$\frac{1}{1 - \delta} \left[ \frac{1}{1 - \frac{O(\delta)}{(1 - \delta) \sum_{i=1}^Z H_k(X_i)}} \right] \leq \frac{1}{1 - \delta} \left[ \frac{1}{1 - \frac{1}{1 - \delta}} \right] = O\left(\frac{1}{1 - \delta}\right) = O(1 + \delta).$$

For this case, *PartitionCompress*'s space bound is  $O((1 + \delta)S_{opt}(\mathbf{X}, \delta))$  which is  $O(S_{opt}(\mathbf{X}))$  since  $\delta < 1 - \Omega(1)$ .

The final bound is  $\max \{O(\delta T), O(S_{opt}(\mathbf{X}))\}$  total space.  $\square$

For this paper, we will also be interested in the LZ78 algorithm, since the dictionary created in the process of compression is useful for searching compressed text without uncompressing it. While Kosaraju and Manzini [14] show that LZ78 does not achieve the optimal bound of  $T \cdot H_k(X)$ ,

they show that it uses space at most  $T \cdot H_k(X) + O((T \log \log T)/\log T)$ . In our context, this means that each cluster uses space  $T \cdot H_k(X) + O((T \log \log T)/\log T)$ .

**Theorem 2.** *Given a set  $\mathbf{X} = \{X_1, X_2, \dots, X_Z\}$  of sensor outputs taken over a sufficiently long time  $T$  from an empirically  $(\delta, m)$ -local sensor system, for any  $0 \leq \delta < 1 - \Omega(1)$ ,*

$$\begin{aligned} \text{Enc}(\mathbf{X}) &= cT \sum_{i=1}^Z \left( H_k(X_i) + O\left(\frac{\log \log T}{\log T}\right) \right) \\ &= O\left( \max \left\{ \delta T, S_{\text{opt}}(X, \delta), \frac{T \log \log T}{\log T} \right\} \right) \text{ bits.} \end{aligned}$$

*Proof.* An optimal algorithm would compress each partition to take the most advantage of the dependence between clusters. It would achieve a space bound of

$$T(1 - \delta) \left[ \sum_{i=1}^Z H_k(X_i) \right] - T \cdot O(\delta)$$

while using LZ78 as the basis for *PartitionCompress* compresses each partition to

$$T \sum_{i=1}^Z H_k(X_i) + \sum_{i=1}^Z O((T \log \log T)/\log T)$$

where  $Z$  is the total number of clusters over all partitions. The ratio is

$$\begin{aligned} &\frac{\sum_{i=1}^Z H_k(X_i) + \sum_{i=1}^Z O((\log \log T)/\log T)}{(1 - \delta) \left[ \sum_{i=1}^Z H_k(X_i) \right] - O(\delta)} \\ &= \frac{1}{1 - \delta} \left[ \frac{\sum_{i=1}^Z H_k(X_i) + \sum_{i=1}^Z O((\log \log T)/\log T)}{\left[ \sum_{i=1}^Z H_k(X_i) \right] - O(\delta)} \right]. \end{aligned}$$

Here we consider the two possible cases for the relationship of  $O(\delta)$  to  $\sum_{i=1}^Z H_k(X_i)$ :

1. *Case  $O(\delta) \geq \sum_{i=1}^Z H_k(X_i)$ :*

$$\frac{1}{1 - \delta} \left[ \frac{\sum_{i=1}^Z H_k(X_i) + \sum_{i=1}^Z O((\log \log T)/\log T)}{\left[ \sum_{i=1}^Z H_k(X_i) \right] - O(\delta)} \right]$$

$$\begin{aligned}
&= \frac{1}{1-\delta} \left[ \frac{\sum_{i=1}^Z H_k(X_i) + \sum_{i=1}^Z O((\log \log T)/\log T)}{O(\delta)} \right] \\
&\leq \frac{1}{1-\delta} \left[ O(\delta) + \frac{\sum_{i=1}^Z O((\log \log T)/\log T)}{O(\delta)} \right]
\end{aligned}$$

Choose  $T$  large enough so that  $O((\log \log T)/\log T) < O(\delta)$ . Then the ratio is

$$\leq \left[ \frac{1}{1-\delta} \right] O(\delta) = O(\delta)$$

for a single one of the  $c$  partitions, or  $O(\delta)$  total times  $S_{opt}(\mathbf{X}, \delta)$ , which is  $O(\delta T)$  since  $O(\delta) \geq \sum_{i=1}^Z H_k(X_i)$ .

2. *Case*  $O(\delta) < \sum_{i=1}^Z H_k(X_i)$ :

$$\begin{aligned}
&\frac{1}{1-\delta} \left[ \frac{\sum_{i=1}^Z H_k(X_i) + \sum_{i=1}^Z O((\log \log T)/\log T)}{\left[ \sum_{i=1}^Z H_k(X_i) \right] - O(\delta)} \right] \\
&= \frac{1}{1-\delta} \left[ \frac{\sum_{i=1}^Z H_k(X_i) + \sum_{i=1}^Z O((\log \log T)/\log T)}{O\left(\sum_{i=1}^Z H_k(X_i)\right)} \right]. \\
&= \frac{1}{1-\delta} \left[ O(1) + \frac{\sum_{i=1}^Z O((\log \log T)/\log T)}{O\left(\sum_{i=1}^Z H_k(X_i)\right)} \right].
\end{aligned}$$

Here, we consider two sub-cases based on the relationship between  $O((Z \log \log T)/\log T)$  and  $\sum_{i=1}^Z H_k(X_i)$ .

(a) *Case*  $O((\log \log T)/\log T) \geq \sum_{i=1}^Z H_k(X_i)$ :

Then the ratio is at most  $O((1+\delta)(\log \log T)/\log T)$ , for a total space of

$O((1+\delta)((\log \log T)/\log T)S_{opt}(\mathbf{X}))$ , which is  $O(T(\log \log T)/\log T)$  since

$O((\log \log T)/\log T) \geq \sum_{i=1}^Z H_k(X_i) > O(\delta)$ .

(b) *Case*  $O((\log \log T)/\log T) < \sum_{i=1}^Z H_k(X_i)$ :

Then the ratio is

$$\leq O\left(\frac{1}{1-\delta}\right) = O(1+\delta)$$

for a single one of the  $c$  partitions, or  $O(1+\delta)$  total times  $S_{opt}(\mathbf{X}, \delta)$ , which is  $O(S_{opt}(\mathbf{X}))$  since  $\delta < 1 - \Omega(1)$ .

The final bound is  $O(\max\{\delta T, S_{opt}(X, \delta), T(\log \log T)/\log T\})$  total space.  $\square$

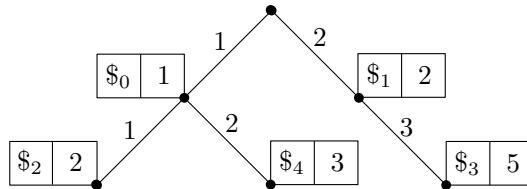
As a direct consequence of Lemma 9(iv) and Theorem 2 we have the following corollary:

**Corollary 3.** *Consider two sensor outputs  $X$  and  $Y$  over the same time period. Let  $X + Y$  denote the componentwise sum of these streams over some commutative semigroup. Then  $Enc_{LZ78}(X + Y) \leq Enc_{LZ78}(X) + Enc_{LZ78}(Y)$  in the empirical setting.*

We have now established  $Enc_{LZ78}(\mathbf{X})$  in both statistical and empirical settings (Theorems 1 and 2 respectively). Throughout this paper we will also be interested in the number of nodes (representing words) in the dictionary resulting from the LZ78 compression process, denoted  $d$ . Note that due to the nature of the dictionary,  $d = O(T/\log T)$  [9], so  $T = \Omega(d \log d)$ . Since  $d \log d$  is the total space needed to store the compressed string and dictionary, in our context  $d \log d = Enc_{LZ78}(\mathbf{X})$ .

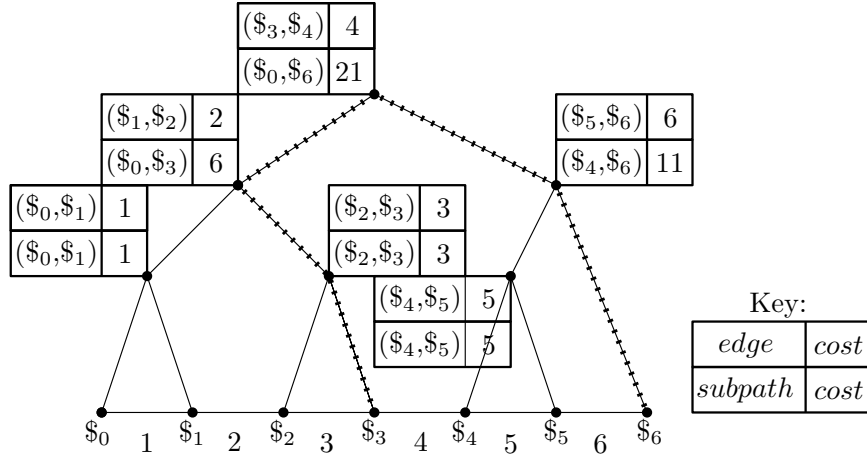
## B Temporal Range Searching

### B.1 Trie Figure



Trie with associated anchor points and counts for a single sensor with observation string “12112312”.

## B.2 Semigroup Setting



A solid path between anchors  $\$0$  and  $\$6$  along sensor output string “123456” with associated binary tree. Each node of the binary tree is annotated with its associated edge cost and its associated subpath cost. The search paths when finding the cost of the subpath between anchors  $\$3$  and  $\$6$  is shown with a dotted line. The cost of this subpath is found to be 0 for the path from  $\$3$  plus 11 for the path from  $\$6$  plus 4 for least common ancestor, for a total cost of 15.

The results from the previous section hold only for group operations. Specifically, they do not hold for queries such as “max” and “min.” In this section we generalize these results to the semigroup setting. In order to handle semigroup operations, for a substring in a given temporal range we need to be able to return the aggregated result in  $O(\log d)$  time without relying on subtraction. The additions explained here are only used to handle the remaining suffix needed for the overlapping case or for lookup in the internal case. In other words, we will only be using this auxiliary data structure when considering queries over time periods within a single word.

We base our auxiliary data structure on the link-cut tree [22]. They annotate edges along the tree to be either solid or dashed so that any path from the root to a leaf node has  $O(\log d)$  dashed edges and any solid path may have as many as  $O(d)$  edges. Each solid path is additionally annotated with a binary tree, so that any node may still be reached through a path of  $O(\log d)$  edges. This binary tree’s nodes represent edges and are annotated with their edge’s cost. We augment the link-cut tree to additionally include the aggregated cost of the subpath represented by the node for each node in the binary trees associated with solid paths (see

Figure B.2). With these additions, the data structure still takes space  $O(d \log d)$ .

To retrieve an aggregated value when given pointers to string endpoints that are fully contained within a solid tree path, start at each endpoint's corresponding leaf node and traverse the path to their least common ancestor. While traversing from the left endpoint, at each parent node that is not the least common ancestor of the endpoints, if the path up the tree goes from a left child to the parent, add the subpath cost stored at the parent to the running total. If the path up the tree goes from a right child to the parent, no cost is added in that step. Proceed symmetrically for the right endpoint. Sum the resulting paths and the least common ancestor's edge cost. This step takes time  $O(\log d)$ , or the depth of a solid path's binary tree. For an example, see Figure B.2.

In order to combine solid paths with dashed paths, recall that since we only need to handle queries within a single word in this section, both endpoints of the substring must be on a single path to the root. We traverse from the given bottom most pointer up the tree until we reach the corresponding ending pointer. The cost of all dashed edges on this path are added to the sum, while solid path segments are handled as described above and the resulting sum is added to the total. There are at most  $\lfloor \log d \rfloor$  dashed edges on the path from any vertex to the root if we make solid vs. dashed edge choices based on the number of nodes in vertex subtrees [22], and traversing any solid path segment takes time in the depth of the binary tree, so this step takes time  $O(\log d)$ .

Finally, note that the endpoint nodes can be identified in the tree by considering  $t_0 - s_0$  and  $s_1 - t_1$  since the queries are along a single path that is indicated by  $s_0$  and  $s_1$ . Annotate the leaf nodes in the solid path binary trees with their numeral positions in the path. Combining the navigation through these trees with following the dashed edges along the identified path, the endpoints of the substring can be found in  $O(\log d)$  time. In total, modifying the query procedures to handle semigroup operations maintains the query time of  $O(\log d)$ .

**Lemma 13.** *The query time for temporal range searching in the semigroup setting is  $O(\log d)$ .*

## C Spatio-Temporal Range Searching Proofs

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 weight 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.

**Lemma 5.** *Given a point set  $P$ , let  $P' \subseteq P$  be any of the groups generated by the *PartitionCompress* algorithm, and let  $P'_1, \dots, P'_h$  denote the associated set of clusters for this group. Then there exists a set of balls  $C = \{b_1, \dots, b_h\}$  that form a set of clumps such that  $P'_i \subseteq b_i$ .*

*Proof.* For the sake of completeness, let us first recall how the set  $P'$  is formed by the *PartitionCompress* algorithm. Initially all the points of  $P$  are unmarked. The algorithm repeatedly selects the unmarked point  $p_i \in P$  that has the smallest  $m$ -nearest neighbor ball (with respect to the entire point set  $P$ ). Let  $b_i$  denote this ball and let  $P'_i = P \cap b_i$ . These points are removed from  $P$ , and all the points of  $P$  lying within  $3b_i$  of  $p_i$  are marked. This process is repeated until no unmarked point of  $P$  remains. Let  $h$  denote the number of iterations until termination, and let  $C$  denote the resulting set of balls. Let  $P' = P'_1 \cup \dots \cup P'_h$ . (This produces one group. To form the next cluster, the process is then applied recursively to the points of  $P$  that were removed. This is all repeated until every point of  $P$  has been assigned to some cluster. See [10] for further details.)

We assert that, for  $1 \leq i \leq h$ , the balls  $\frac{1}{2}b_1, \dots, \frac{1}{2}b_h$  are pairwise disjoint. Consider any pair  $i, j$ , where  $1 \leq i < j \leq h$ . Let  $r_i$  and  $r_j$  denote the radii of  $b_i$  and  $b_j$ , and let  $p_i$  and  $p_j$  denote their respective centers. Since when  $p_i$  is being processed, all the points lying within distance  $3r_i$  are marked, and since only unmarked points are chosen as centers of the balls, we have  $\|p_i p_j\| \geq 3r_i$ . Also, since the ball of radius  $\|p_i p_j\| + r_i$  centered at  $p_j$  contains the  $m$ -nearest neighbor ball of  $p_i$ , it follows that this ball contains strictly more than  $m$  points, from which we conclude that  $r_j \leq \|p_i p_j\| + r_i$ . Combining these observations we have

$$\begin{aligned} \frac{r_i}{2} + \frac{r_j}{2} &\leq \frac{1}{2}(r_i + (\|p_i p_j\| + r_i)) \leq \frac{1}{2}(\|p_i p_j\| + 2r_i) \\ &\leq \frac{1}{2} \left( \|p_i p_j\| + \frac{2}{3}\|p_i p_j\| \right) < \|p_i p_j\|. \end{aligned}$$

Because the sum of their radii is less than the distance between their centers, it follows that the balls  $\frac{1}{2}b_i$  and  $\frac{1}{2}b_j$  are pairwise disjoint, and this completes the proof of the assertion.

To see that  $C$  is a set of clumps, consider any positive real  $r$  and  $r'$ . Let  $b$  be any ball of radius  $r$ , and let  $C'$  denote the subset of balls of  $C$  whose radius is at least  $r'$ . By the above assertion, the centers of any two balls of  $C'$  must be at distance at least  $r'$  from each other. By basic properties of doubling spaces, it follows that  $b$  can be covered by  $O((1 + r/(r'/2))^d)$  balls of radius  $r'/2$ . Clearly, each ball of this set can contain the center of at most one ball of  $C'$ . Therefore,  $|C'| = O((1 + (2r/r'))^d) = O((1 + (r/r'))^{d+1})$ . Setting  $\gamma = d + 1$  completes the proof.  $\square$

**Lemma 6.** *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-temporal 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  $O(h(S) \cdot pp(S) + \text{Enc}(\mathbf{X}))$ , query time  $O(qt(S) \cdot \log T)$ , and space  $O(sp(S) + h(S) \cdot \text{Enc}(\mathbf{X}))$  bits.*

*Proof.* We first run the *PartitionCompress* algorithm on the point set  $P$  of the  $S$  sensor locations. Recall that this partitions  $P$  into  $O(1)$  groups, which by Lemma 5 can each be represented by a collection of clumps. We build a range searching structures for each of the resulting set of clumps. We will answer each query by invoking the range search separately on each of the individual structures, and then summing the results. Henceforth, we consider just the processing of a single group, which we will denote by  $P'$ .

We augment the clump range-search structure for  $P'$  by building one temporal range search structures for each of the individual clumps of  $P'$  as well as for each of the generators, that is, for each of the internal nodes of the associated partition tree. First, recall that each clump consists the sensor streams for some number  $m' \leq m$  of sensors. For each clump we treat the data from this clump as a time stream whose elements are  $m'$ -element vectors, where the  $i$ th element of the vector is the count of the  $i$ th sensor. We compute a temporal range search data structure for the associated stream of vectors (where the semigroup sum is extended to the semigroup sum over vectors). Next, for each node  $u$  of the tree, let  $g_u$  denote the associated generator consisting of the points  $\{p_1, \dots, p_f\}$  stored in the leaves that are descended from  $u$ . Let  $\{X_1, \dots, X_f\}$  denote

the corresponding set of the sensor streams. Let  $X_u$  be the aggregated stream  $\sum_{i=1}^f X_i$ , formed by taking the componentwise sum of the observations from all  $f$  streams. (Unlike the clump case, we collapse all the sensors counts into a single sum, rather than creating an  $f$ -element vector. This is because  $f$  may generally as large as the total number of sensors.) We build a temporal range searching structure for  $X_u$ , and associate this auxiliary tree with  $u$ .

Next, let us consider how to answer a given spatio-temporal query  $\langle Q, [t_0, t_1] \rangle$  over  $P'$ . We first apply the range searching data structure for  $Q$  over the set of clumps associated with  $P'$ . Recall that this returns (1) a subset of generators lying within  $Q$  and (2) the clumps that are stabbed by  $Q$ . The former set may be assumed to be associated with a set of internal nodes of the tree and the latter with a set of leaf nodes of the tree. For each node  $u$  of (1), we invoke the corresponding auxiliary temporal data structure over the aggregated stream  $X_u$  and the time interval  $[t_0, t_1]$ , and include the resulting semigroup sum in the final total. For each leaf node of (2), we invoke the associated auxiliary temporal range search structure for vector  $[t_0, t_1]$  to determine the semigroup vector sum over this interval. For each sensor of the clump we determine whether it lies within  $Q$ , and if so, we include its component of the vector sum in the final total.

The space used by the data structure is equal to the total space  $sp(S)$  for the range searching structure over clumps, plus the space needed for the temporal range search structures for each of the clumps and each of the generators. To bound this quantity, consider the  $h(S)$  levels of the tree. Each of the nodes of this level is associated with a generator, such that each sensor stream contributes to at most one node of the level. It follows from  $m$ -locality and Lemmas 8 and 9(iv) (for the statistical and empirical cases, respectively) that the entropy of the componentwise sum of the stream is not greater than the sum of entropies of the sensor streams at the leaf level, which by Corollaries 1 and 3 is at most  $\text{Enc}_{\text{LZ78}}(\mathbf{X})$  bits. Summing over  $h(S)$  levels yields the desired space bound. Similarly, the preprocessing time of the data structure is just the preprocessing time needed to build the range searching structure over clumps, plus the time needed to construct the individual auxiliary temporal range search structures.

To bound the total query time, observe that the query time is dominated by the time  $O(qt(S))$  to compute the set of nodes whose associated clumps and generators form the answer to the query, together with the

$O(\log T)$  time from Lemma 3 to access each auxiliary data structure to answer the temporal range queries. This completes the proof.  $\square$

**Lemma 7.** *There exists a data structure for answering  $\varepsilon$ -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  $O(n \cdot (\text{prec}(C) + \log n))$  bits, where  $\text{prec}(C)$  denotes the maximum number of bits of precision in the geometric coordinates used to define  $C$ .*

*Proof.* Recall from [4] that a BBD-tree for a set of  $n$  points is type of balanced and compressed quadtree decomposition, in which the tree has size  $O(n)$  and height  $O(\log n)$ . In order to guarantee that the tree has logarithmic depth, in addition to the standard quadtree splitting operations there is a decomposition operation, called *centroid shrinking*. Define a *quadtree box* to be any axis-parallel hypercube that can be formed by starting with the unit hypercube, and repeatedly splitting it into  $2^d$  congruent subcubes, by passing  $d$  axis-parallel hyperplanes through the center of the cell. Given a quadtree box  $b$ , this operation computes a nested quadtree box  $b' \subseteq b$  such that a constant fraction of the points of  $b$  lie within  $b'$ . Corresponding to this operation there is a special node of the tree, called a *shrink node*, whose two child nodes are associated with  $b'$  and  $b \setminus b'$ , respectively. Each node of the tree is naturally associated with a region of space, called its *cell*. The cell associated with each node of the BBD tree is either a quadtree box or the set-theoretic difference of two quadtree boxes, one nested within the other. (See [4] for further details.)

The BBD-tree data structure is generalized to process range searching over clumps as follows. First, we assume that the sensor points have been scaled so they lie within a unit hypercube. The center points of the clumps are inserted into the BBD-tree, just as in [4], with the following exception. Let  $u$  denote a node of the clump-based BBD-tree, let  $q$  denote the cell associated with  $u$ , and let  $s$  denote  $q$ 's maximum side length (also called its size). Each clump whose center lies within  $q$  and whose radius lies between  $s/2$  and  $s$  is stored in a special leaf node which is made a child of  $u$ . It is easy to establish the invariant that the descendants of any node whose cell size is  $s$  are clumps of radius at most  $s$ . By Lemma 5, the number of such leaves per node is  $O(1)$ . Otherwise, the preprocessing is identical to that of the BBD-tree. The preprocessing time is essentially the same as that of the BBD-tree, which is  $O(n \log n)$ . The space is equal to the total space needed for the point coordinates, which is  $O(n \cdot \text{prec}(C))$  bits, and the total space needed for the tree and its pointers, which is  $O(n \log n)$  bits.

In order to answer a query, we follow essentially the same searching procedure given in [4], but with a few differences. Recall that the algorithm recursively descends the tree starting at the root. On its arrival at some leaf node  $u$ , we test whether the associated clump lies within  $Q^+$  (in which case we include it in the set of generators lying within  $Q^+$ ) or is stabbed by the annulus  $Q^+ \setminus Q^-$  (in which case we include it among the stabbed clumps). On arrival at an internal node  $u$ , let  $q_u$  denote the associated cell and let  $s_u$  denote its size. Since the clumps lying within  $u$  have radius at most  $s_u$ , we check whether  $q_u$  dilated by  $s_u$  lies entirely within  $Q^+$ , and if so we include the associated generator among those lying within the range query. On the other hand, if the dilation of  $q_u$  lies outside of  $Q^-$ , we ignore the associated generator. If neither of these cases holds, then we recursively apply the search to the children of  $u$ .

By a straightforward adaptation of the packing arguments given in [4], it follows that the number of nodes visited by this algorithm is  $O((1/\varepsilon^{d-1}) + \log n)$   $\square$