ABSTRACT

Title of dissertation: GEOMETRIC ALGORITHMS FOR OBJECTS IN MOTION

Sorelle Alaina Friedler,
Doctor of Philosophy, 2010

Dissertation directed by: Professor David M. Mount
Department of Computer Science

In this thesis, the theoretical analysis of real-world motivated problems regarding objects in motion is considered. Specifically, four major results are presented addressing the issues of robustness, data collection and compression, realistic theoretical analyses of this compression, and data retrieval.

Robust statistics is the study of statistical estimators that are robust to data outliers. The combination of robust statistics and data structures for moving objects has not previously been studied. In studying this intersection, we consider a problem in the context of an established kinetic data structures framework (called KDS) that relies on advance point motion information and calculates properties continuously. Using the KDS model, we present an approximation algorithm for the kinetic robust k-center problem, a clustering problem that requires k clusters but allows some outlying points to remain unclustered.

For many practical problems that inspired the exploration into robustness, the KDS model is inapplicable due to the point motion restrictions and the advance

flight plans required. We present a new framework for kinetic data that allows calculations on moving points via sensor-recorded observations. This new framework is one of the first within the computational geometry community to allow analysis of moving points without *a priori* knowledge of point motion. Analysis within this framework is based on the entropy of the point set's motion, so efficiency bounds are a function of observed complexity instead of worst-case motion. Analysis is also considered under the more realistic assumptions of empirical entropy and assumptions of limited statistical independence. A compression algorithm within this framework is presented in order to address the storage issues created by the massive data sets sensors collect. Additionally, we show experimentally that this framework and accompanying compression scheme work well in practice. In order to allow for retrieval of the collected and compressed data, we present a spatio-temporal range searching structure that operates without the need to decompress the data. In sum, the collection scheme, compression algorithm, theoretical analyses, and retrieval data structures provide a practical, yet theoretically sound, framework within which observations and analyses can be made of objects in motion.

GEOMETRIC ALGORITHMS FOR OBJECTS IN MOTION

by

Sorelle Alaina Friedler

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2010

Advisory Committee:
Professor William Gasarch
Professor Samir Khuller
Professor David Mount, Chair/Advisor
Professor Steven Selden
Professor Amitabh Varshney

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Work in theoretical computer science has focused, to date, on a fundamental theoretical understanding and analysis of computer science as a discipline. However, recent work has broadened the domain of problems considered to include theoretical analyses of real-world motivated problems [HSV10, CHSV10, Cha09]. I believe that theoretical computer science has important contributions to make in these realms and that a theoretical analysis of practically-motivated problems is critical to the solution of these problems as well as to the growth of theoretical computer science. My work presented here is motivated by such issues and contributes directly to the understanding and analysis of objects in motion.

Collecting information about moving points has become increasingly common. This has often led to massive data sets containing point observations for multiple time steps. As sensing technologies become increasingly inexpensive and available, these data sets will only get larger. Currently, these data sets are quite diverse, ranging from the sounds and positions of owls [MIT], to the movements of people in a building [WILW07], to the locations of cars via GPS navigation devices or road-embedded loop detectors [Tel, POR], or to the locations of fish in the ocean [POS]. The wide scope of applications involving motion has given rise to a diversity of

questions about these data sets. For example, these questions may involve migration of animals, the dynamics of social relationships, and navigation in the presence of traffic. Due to the vast amounts of data involved, these problems require automated and efficient solutions. With the development of sensors of lower cost and higher reliability, the prevalence of applications and the need for efficient processing and storage will increase. Additionally, in many real-world applications it is common for the data set to contain observations that do not exactly fit a given model; thus outliers are common. This thesis answers questions inspired by this real-world data via a sensor-based theoretical framework for motion data.

Many areas within and outside of computer science consider calculation of properties of moving points. The nature of these calculations and the associated computational challenges depend on the particular area of study. For example, when working with wireless sensor networks, the question is often how to capture the data. Problems of sensor limitations due to processing capacity, limited power sources, and communication range are considered along with calculation on the data collected by the entire network [ASSC02]. Within the database community, the question is how to store and answer queries involving these points [Wol02]. Physicists, on the other hand, consider statistical calculations on moving points as part of simulations and similar simulations underly animation in computer graphics [Mon05]. Compression of moving points is also considered within computer graphics for video compression needs [BCMR03, Gal91].

Motion is a continuous phenomenon. However, the practical limits on its measurement have resulted in a variety of methods ranging from continuous modeling

to discrete modeling involving discrete time steps. In the field of computational geometry, most of the research has focused on piecewise continuous motion of objects, called *kinetic data* [GJS96, Ata85, ST95, ST96, BGH97, BG99]. Most of these rely on *a priori* information about point motion and expect a "flight plan" in the form of an algebraic formula of constant degree, thus restricting the point motion as well. A survey of practical and theoretical aspects of modeling motion can be found in [AGE+02]. The *kinetic data structures* (KDS) model proposed by Basch, Guibas, and Hershberger [BGH97, BG99] has become the standard method for dealing with kinetic data. The KDS model requires algebraic expressions representing point motion, but allows these "flight plans" to change. There has not been as much work within computational geometry on discrete modeling of motion [Kah91]. Continuous modeling allows for perfect accuracy in theory, assuming that objects adhere to the given flight plans, while discrete modeling is less precise but more practical, simulating actual data collection.

1.1 Contributions

In this dissertation, I consider statistical problems for both continuous and discrete models. One of the problems that hasn't been considered within the domain of moving points is robustness. Robustness is an important statistical characteristic of an estimator; it ensures that the presence of some data outliers due to measurement error or heterogeneity of the data set does not alter the result by a large amount. The *breakdown point* makes this notion precise and is defined to be the smallest

percentage of arbitrarily large outlying data which can cause an arbitrarily large estimate result [Ham71]. The largest a breakdown point can be is 50 percent, since any larger percentage of outlying data makes it impossible to distinguish between inliers and outliers.

A natural class of geometric problems with statistical properties to consider is *clustering*. These problems group points in some metric into subsets (known as *clusters*) that are the best possible under some criterion. These criteria are often based on a user given parameter k that determines the number of clusters or the number of points per cluster. The best clustering is then defined based on an individual cluster distance measure, such as cluster radius, and a merge function over all the clusters, such as summation [BE97]. When the points are moving, clustering problems can be considered to create clusters that move with the points over time [GGN06] or can create static clusters that provide an approximation of the optimal at any time step [HP04]. Clustering problems have many applications including analysis of social networks, image or video segmentation, and analysis of vehicle motion [SS07]. The work presented in Chapter 3 considers a robust clustering problem on a continuous model (the KDS model)[1].

A more practical approach that aims to allow statistical calculations with greater efficiency is presented in Chapter 4[2]. This work presents a new framework for kinetic data that is based on a discrete sampling model. Data collection through sampling is commonly done using sensors, and the new model takes this approach.

[1]For a published version of the work presented in Chapter 3, see [FM10a].

[2]For a published version of the work presented in Chapter 4, see [FM09].

There is a growing appreciation of the importance of algorithms and data structures for processing large data sets arising from the use of sensor networks, particularly for the statistical analysis of objects in motion.

The framework presented in Chapter 4 assumes very little about the motion of the objects under consideration – it is purely observation-based. Each sensor is assumed to observe the motion of objects in some region surrounding it, and record an *occupancy count* indicating the number of objects passing within its region during the observed time step. The object motion is thought of as continuous even though the observation of it is discrete. (These results apply more generally to any discrete statistic over a domain of constant size.) 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.). It is to be expected that the entities observed by one sensor will also likely be observed by nearby sensors, albeit at a slightly different time. Central to the framework is the notion that each sensor's output is statistically dependent on a relatively small number of nearby sensors. This assumption is verified experimentally in Chapter 5.

With the goal of handling observed data from these sensor networks, it is important to create a realistic system that can store and retrieve the large amounts of data generated. In Chapter 4 we consider the first issue, that of compression for storage. Compression methods may either be *lossless* (allow the original data to be fully reconstructed) or *lossy* (some of the data may be permanently lost through approximation methods). 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 [Huf52, Ris76, ZL77, ZL78] but remain mostly unstudied in a sensor-based setting [FM09].

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 close sensors are in the same cluster. The algorithm partitions the points into a constant number 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. Assuming such an algorithm, the compression algorithm achieves an encoding size within a constant factor of the optimal joint entropy bound.

The compression algorithm presented in Chapter 4 is initially analyzed in terms of the Shannon entropy [Sha48] and assuming pure statistical independence of distant sensor outputs. However, in the realistic settings that motivate this work these assumptions are too strong. In Chapter 5 a more realistic analysis is given of the compression algorithm in terms of the empirical entropy of the sensor outputs. Empirical entropy [KM99] operates over the observed probabilities of data streams and assumes a history window of fixed size within which local dependencies may appear. Both of these properties represent realistic improvements over the Shannon entropy, which requires knowledge of an underlying random process generating the data and assumes an infinite window size (optimality analyses hold in the limit [WZ94]).

In order to fully understand the algorithm, Chapter 5 introduces new empirical entropy constructs analogous to statistical joint entropy, conditional entropy, and independence. These new definitions allow an empirical entropy analysis of the compression algorithm that shows that, when the underlying string compression algorithm used is the Lempel-Ziv dictionary compression algorithm [ZL78], the sensor data compression algorithm is still on the order of the optimal encoding bound.

In addition, Chapter 5 investigates a more lenient independence assumption for the model. Instead of assuming strict statistical or empirical independence, the new assumption allows some limited underlying dependence between even distant sensor outputs. For example, this might represent the general decrease in traffic volume across all sensor regions at night. Under this limited independence model, the sensor data compression algorithm is shown to be on the order of the optimal

encoding bound. With these realistic analyses in place, higher level questions about the data can now be considered.

In order to consider more complex statistical analyses of the data collected by this observation-based framework, it is important to be able to retrieve data from its compressed form. In this effort, I consider a retrieval problem over the compressed data without the need to decompress it. The implied assumption is that when dealing with massive data sets that required compression in order to be stored, it is necessary not to have to decompress the data in order to retrieve the desired information. Recently, many problems of this form have been considered including finding all occurrences of a given pattern in the text [FM05], retrieving a specific substring [FV07], and querying compressed XML databases [FLMM06].

In Chapter 6, I present the first range searching data structures and algorithms operating over compressed text[3]. The specific range searching question considered allows for retrieval of the aggregate number of moving objects counted in a specific spatial region over a given temporal period. An analysis relying on the realistic assumptions of Chapter 5 shows that these data structures and algorithms operate with preprocessing time and space on the order of the encoded size of the data and have query time logarithmic in the total observation time and number of sensors. Additionally, experimental analysis shows that these structures take an order of magnitude less space than the uncompressed version of the data while still providing faster query times than a naive method. Although interesting in its own right, I believe that this range searching structure can serve as the underlying data structure

[3]For a published version of the work presented in Chapter 6, see [FM10b].

for future higher level statistical analyses of data collected within this framework.

My ultimate goal, no matter the framework, is to answer statistical questions about kinetic data with provable efficiency. I believe that many open problems remain in this particular area, as well as in many other areas inspired by observation of real-world data. Kinetic data models can vary in terms of practicality, theoretical purity, point motion assumptions, and data collection methods. Similarly, there are many analysis methods which can be applied to these models, and the choice of method may vary by model. Competitive and exact analyses can consider communication complexity, storage size, algorithm run time, and other model specific features. In addition, all of these questions should be examined with regards to practical applicability in multiple domains. Many problems remain to be considered within these various models. Robust statistical estimators can be considered including the point set diameter, clustering problems, and the least median of squares estimator. Other range searching and retrieval questions could also be considered. There may also be many practically inspired, domain-specific problems that remain to be discovered. In Chapter 7, I introduce some of these open problems. Some are inspired by the specifics of the solutions presented here while others come from the broader context of understanding moving objects.

Chapter 2

Literature Review

2.1 Data Structures for Moving Points

One of the first papers about calculations of properties of moving points in a computational geometry context was written in 1985 by Mikhail Atallah [Ata85]. In this beginning work, points are assumed to follow paths, also known as *flight plans*, modeled by polynomials of degree at most k that are functions of time; these motion paths are provided at the beginning of the algorithm and do not change. Atallah examines the properties of these function interactions in a static context with time as an additional dimension. He gives results regarding the descriptive complexity of the lower envelope (that is, the minimum value over all the functions) and the 2-D convex hull over all time steps. He also gives time complexity results for calculating these changing quantities. While points are modeled by polynomials, they are not assumed to be continuous at all points; instead, the number of undefined periods of time and discontinuous jumps are assumed to be bounded ($O(1)$).

Atallah showed that the concept of Davenport-Schinzel sequences are important in this context. A Davenport-Schinzel sequence of order s is a sequence of characters from an alphabet of size n that does not contain any consecutively re-

peating characters or any alternating subsequences of length $s+2$ [SA95]. Let $\lambda_s(n)$ denote the maximum lengths of such a sequence. Atallah showed that, when considering the functions over time, there can be at most $\lambda_s(n)$ function pieces in the lower envelope of this collection, or $\lambda_s(n)$ possible values for the minimum value of the function set over time. Atallah's proof proceeds by showing that $\lambda_s(n)$ directly models a lower envelope by assigning characters to each function piece on the lower envelope. Since consecutive repeating characters would model the same function appearring next to itself, these do not occur. For functions that do not intersect more than s times, it is impossible to have alternating subsequences of length more than $s+2$. He shows that since $\lambda_s(n)$ is bounded by $O(n \log^* n)$, the number of function pieces on the lower envelope is also $O(n \log^* n)$. (More recent research has tightened these bounds [SA95]).

The bound on the number of lower envelope changes is also used to provide a bound on the number of convex hulls over time. Atallah shows that a single point changes from being in the convex hull to not in the convex hull $O(\lambda_s(n))$ times by defining membership in the convex hull based on the minimization of a function. This is then applied to all n points to get a bound of $O(n \cdot \lambda_s(n))$ convex hulls.

Other early papers include a 1983 paper by Guibas, Ramshaw, and Stolfi [GRS83], which views point motion as following predetermined curves and polygons. It frames questions in terms of the relationships between these curves and polygons. In 1991, Kahan [Kah91] was the first to introduce a framework for motion in which flight plans are not used, and the only required prior knowledge is an upper bound on each point's velocity. Instead, Kahan's model relies on a function that can

be queried to determine current point locations. Schömer and Thiel examined the problem of collision detection between moving polyhedra in a series of papers in 1995 and 1996 [ST95, ST96]. Motion is described in advance by polynomial functions. Similarly, a 1996 paper by Gupta, Janardan, and Smid [GJS96] considered collision detection, minimum separation, and other problems on points, line segments, or hyperrectangles moving on predetermined linear paths. In 1997, Basch, Guibas, and Hershberger [BGH97,BG99] introduced the kinetic data structures (KDS) model and this became the standard for moving point calculation. This framework also requires a predetermined functional model for each point's path, but allows these models to change at discrete time instances. The KDS model is discussed in more detail in Section 2.2.

Despite the progress to date in modeling motion, many real-world inspired issues remain to be addressed. In an effort to precisely identify these areas and spur research into a unified motion framework a group of researchers participated in a workshop in 2002, which produced a survey of these issues across computational geometry, mesh generation, physical simulation, biology, computer vision, robotics, spatio-temporal databases, and mobile wireless networks [AGE+02]. They also suggested directions for future research. Here, we focus on the problems they describe that relate directly to the work of this thesis.

Within computational geometry, Agarwal *et al.* [AGE+02] propose research into *motion-sensitive* algorithms, bounds on the number of combinatorial changes, and decentralization. Motion-sensitive algorithms give complexity bounds that are based not on the worst case bounds given any point motion, but provide efficiency

measures based on the predictability of the moving objects and their relation to each other (note that the framework that we will introduce later in Chapter 4 is motion-sensitive). Similarly, they propose analyzing the number of combinatorial changes to a property as a function of the objects' motion complexity instead of a simple worst-case bound. These proposed methods have the advantage of more realistically modeling the efficiency of algorithms on moving point sets. In addition, for practical use in many situations such as ad-hoc networks, they mention that it would be useful to develop a model for motion that allows computation to be distributed over multiple processors.

In the field of computer vision, motion analysis is used for applications including shape identification and tracking, initialization of a tracking sequence, and handling additional complexity in terms of lighting, motion, or shape. Statistical error analysis is used to create models of object motion. Agarwal *et al.* [AGE+02] suggest that further broadening of research consider multiple frames for tracking initialization, robustly handle error through learning-based methods, and generally view motion through a high-level hierarchical view in order to take advantage of all the data available. Challenges in the field of wireless networks echo the need for a hierarchical view of the data. Agarwal *et al.* [AGE+02] propose that research focus on modeling predictable user motion at many levels in the data hierarchy. The hope is that these models would allow future motion to be predicted based on the model for a single level without detailed knowledge of motion at other levels, for example without knowing the motion of an individual at the lowest level of the hierarchy.

Overall, Agarwal *et al.* [AGE+02] identify eight common themes of issues that

remain to be addressed in motion modeling. These include the ability to handle robustness, both in the context of data error and in the context of higher level analysis of motion trends through the use of hierarchical data structures. The struggle between continuous and discrete models remains to be fully explored, with research in continuous models not being confined to the theoretical community and discrete models not only being analyzed by practical researchers. Coupled with this, methods of identifying current motion of objects with an emphasis on unpredicted motion remain to be fully explored. Finally, the decentralization of data processing, a requirement in many practical applications, is especially important. Section 2.3 returns to some of these questions, and in Chapter 4 we present a framework that addresses some of these issues.

One interesting alternate method for handling kinetic data was proposed by Har-Peled in 2004 [HP04]. He examines the discrete k-center clustering of a set of points moving with motion modeled by polynomials of degree at most μ. The *discrete k-center problem* is given a set of n points and finds the k center points taken from the input that minimize the maximum distance (called the *radius*) from any point to its closest center. For this clustering problem, each center together with the points that are closest to that center constitute a cluster. Instead of maintaining the k clusters as they change over time, Har-Peled finds a single static clustering with $k^{\mu+1}$ clusters that is within a constant factor of the optimal clustering at any time. He begins with a clustering algorithm for a static point set that randomly samples points, partitions those points into k clusters, and then partitions all points that were not covered by the first clustering into an additional k clusters. This

two-round clustering algorithm is then extended to an η-round clustering algorithm by repeating the first two steps for uncovered points.

To handle moving point sets, Har-Peled partitions time into intervals when the relative distances between points do not change. For each of these intervals, the Gonzalez [Gon85] greedy clustering algorithm (which bases the partitioning only on inter-point distances) is used as the black-box clustering algorithm needed by Har-Peled's static clustering algorithm. The radius returned is a 2-approximation of the optimal over that interval, where the time for the optimum over the interval is found by calculating the lowest point on an upper-envelope representing radius lengths. The set with the minimum radius over all the intervals gives the answer to a different kind of clustering problem in a dimension one larger than the original (the dimension for time is added). This set is then expanded to create a static clustering with more centers that holds over time. Har-Peled's algorithm runs in $O(nk)$ time for $k = O(n^{1/14})$. Other algorithms that use a similar strategy of considering the points statically in a higher dimension including time are those posed by Atallah [Ata85], Guibas *et al.* [GRS83], and Gupta [GJS96]. The current standard for calculating properties of moving points is the KDS model that handles motion in an online manner. It is described in the next section.

2.2 Kinetic Data Structures Model

Basch, Guibas, and Hershberger [BGH97,BG99] introduced a model for kinetic data called a *kinetic data structure* (KDS). This model assumes advance knowledge

of point flight plans, but allows these plans to change. Algorithms are developed to track specific properties of moving points in an online manner. This is done through a set of boolean conditions called *certificates* and a corresponding set of update rules. Certificates guarantee geometric relations necessary to a particular problem's solution, and update rules specify how to respond when a certificate fails. Certificate failures are predicted and queued based on the points' planned paths of motion, assumed to be in the form of algebraic expressions. KDSs are evaluated based on properties of the certificate set.

There are four criteria under which the computational cost of a KDS is evaluated: responsiveness, efficiency, compactness, and locality [Gui04]. *Responsiveness* measures the complexity of the cost to repair the solution after a certificate fails. *Efficiency* measures the number of certificate failures as compared to the number of required changes to the solution as the points move. *Compactness* measures the size of the certificate set. *Locality* measures the number of certificates in which each point participates. Guibas provides a more detailed overview of kinetic data structures in [Gui04].

In order to illustrate the KDS model, Basch *et al.* [BGH97, BG99] give a KDS that maintains the 2-D convex hull over time. First they consider the static solution where each point (a, b) is dualized to the line $y = ax + b$ and lower part of the convex hull is viewed as the upper envelope of the set of lines. Finding the upper envelope is done through a divide-and-conquer algorithm. The merging of two upper envelopes proceeds by sweeping from left to right and determining, for each line intersection point on either upper envelope, which envelope is higher. Certificates guarantee

relative properties of x and y coordinates and line slopes in order to maintain this merge under motion. These certificates are set-up to only record relative properties of those lines surrounding them, and the number of certificates kept for each line intersection or line is a constant, so the KDS is compact, local, and responsive. The argument for efficiency is based on a multi-dimensional upper envelope complexity bound of $O(n^2 + \varepsilon)$ (which is modeled in the kinetic context by a three dimensional set consisting of two spatial dimensions and one temporal dimension) [Sha94], which bounds the number of events that may be caused by certificate failures in this KDS system. The complexity of the moving convex hull is $\Omega(n^2)$, so the KDS is efficient.

Although most KDS solutions are based on a particular computational problem, Gao *et al.* [GGN06] introduced a flexible kinetic data structure that can be used to solve a number of different problems involving kinetic point sets. (Later in Chapter 3, we will make use of this structure.) This structure is hierarchical in nature, and can be used both as a tree-like access structure as well as a geometric spanner (defined below). Gao et al. dubbed it a *deformable spanner*. The hierarchy and spanner both update dynamically as the points move, so this data structure provides an underlying framework on which future problems that rely on hierarchy or spanner properties can be built.

A *γ-spanner* is a graph connecting points in a point set S in $\Re^d$ with the property that for any two points in S, the distance between those points on the graph is at most γ times the distance between those points in the underlying metric. The deformable spanner is a $(1+\varepsilon)$-spanner. The *aspect ratio*, denoted α, is defined as the ratio between the maximum and minimum distances between any two points

of S. For moving point sets, the aspect ratio α is actually a function of time. When considering the aspect ratio in the context of time complexity, one simple solution is to consider the maximum ratio over all times. Additionally, $\varepsilon > 0$ refers to a user-given input parameter.

The Gao *et al.* [GGN06] spanner is constructed based on the concept of a hierarchy of discrete centers. Given a point set S, a *hierarchy of discrete centers* is a sequence of subsets $S = S_0 \supseteq S_1 \supseteq ... \supseteq S_{\lceil \log \alpha \rceil}$ such that the following properties hold for $0 \leq i \leq \lceil \log \alpha \rceil$:

- Each center in S_{i-1} is within distance 2^i of some center in S_i, the ith level of the hierarchy.
- Centers in S_i are chosen from S_{i-1}.

A center p at level i is said to *cover* a center q in level $i-1$ if q is within distance 2^i of p. By definition each center q at level $i-1$ is covered by some center in level i. One such center p is selected (arbitrarily) to be q's parent. The center q is called the *child* of p. Other standard tree relationships are used including ancestors and descendants (both of which are considered in the improper sense, so that a node is an ancestor and descendant of itself) and siblings [CLRS01]. *Cousins* are defined as the children of a node's parents' siblings. Some properties about the deformable spanner, which follow immediately from the above properties or are proven in [GGN06], are given below:

- $S_i \subseteq S_{i-1}$
- For any two center $p, q \in S_i$, with associated points $p, q, \in S$, $\|pq\| \geq 2^i$.

- The hierarchy has a height of at most $\lceil \log_2 \alpha \rceil$.
- Any center in S_0 is within distance 2^{i+1} from its ancestor in level S_i.

The deformable spanner maintains four types of certificates: parent-child certificates, edge certificates, separation certificates, and potential neighbor certificates. *Neighbors* of a node p in level i are defined as all nodes in that level within distance $c \cdot 2^i$ of p. The certificates are based on this distance $c \cdot 2^i$, where $c > 4$ [GGN06]. The KDS for this spanner is appropriately efficient, local, compact, and responsive.

Many other problems have been considered using KDS, including clustering, hierarchical data structures, and minimum spanning trees [BBKS00, AGG02, AdBG09, BGZ97, Gui98]. For example, in 2008 Abam *et al.* presented a KDS that maintains a $(1+\varepsilon)$-spanner over point motion independent of the point set's aspect ratio. The spanner is based on the union of Delauney triangulations done over the points where distance is defined based on a metric that uses a diamond instead of a unit circle and the diamond is rotated for each triangulation. The Delauney triangulation based on the Euclidean metric is easy to kineticize since there are local properties that guarantee the triangulation and corresponding local update rules. Abam's spanner uses similar properties and update rules modified for the diamond-based metric and is shown to be efficient.

2.3 Sensors and Streams

Let us now move from the continuous model of motion exemplified by KDS to systems that analyze motion based on discrete time samples. One practical way to

observe and record motion is through the use of sensors in a sensor network. *Sensors* are small nodes with the ability to sense characteristics of their environment in addition to limited data processing and communication ability. *Sensor networks* contain many sensors as nodes networked with each other that are densely deployed to observe some environment. These networks can be applied for military benefit, environmental protection, health monitoring, household convenience, vehicle detection, and in many other situations [ASSC02].

Akyildiz *et al.* [ASSC02] identify eight main issues in sensor network design in a 2002 survey of research in wireless sensor networks. Some of these constraints are specific to sensor networks and require new techniques and technologies. Sensors are assumed to be cheap, which allows broad use, but also means that they are prone to failure. Correspondingly, sensor networks are evaluated based on their *fault tolerance*, their ability to continue operating without interruption if some sensors fail. The specific level of fault tolerance needed, as measured by the probability that no nodes will fail within a given time interval, is dependent on the measurement error and added environmental strain on the sensors based on the application. *Scalability* refers to the ability of the network to operate efficiently on millions of nodes and to take advantage of the high density of sensor deployment. Due to the vast number of sensors in the network and their tendency to fail, the sensor network topology must be highly malleable. Production costs of each node must be low in order to make these sensor networks cost effective, and there are many hardware challenges, both related and unrelated to cost. The network must be able to operate unattended for long periods of time, and may encounter harsh environmental factors depending on

the application and deployment location. In order to utilize this network, the sensors must be able to communicate with each other and, possibly through multiple hops, with a central server. The transmission media chosen should be globally available and not require a line of sight between sender and receiver (as is required by infrared communication). Finally, power consumption on individual sensor nodes is one of the most important areas of research, since the sensors have limited battery life and sensors that run out of power must be removed from the network. Limiting power consumption while sensing, communicating, and processing data is crucial [ASSC02].

The data collected by the sensors at small time intervals over the entirety of their lifespan (which is theoretically infinite) is reported at each time step and makes up what is known as a stream of data. More formally, *data streams* are a sequence of data items that arrive online, are potentially unbounded in size, arrive in an undetermined order, and are discarded after they have been processed. In a 2002 survey of models and issues in data stream systems, done from a databases perspective, Babcock *et al.* [BBD+02] present current query challenges. The main underlying challenge is the problem of unbounded memory requirements due to the potentially unbounded length of the sequence of data. In addition, Arasu *et al.* [ABB+04] show that without knowing the length of the input data stream, it is impossible to bound the memory requirements of a single query using a join operation. Due to these memory constraints, approximations to query answers are desirable. Random sampling, histograms, and other synopsis techniques can be used to provide an overview of the entire data stream. Alternatively, sliding windows allow queries to be answered based only on recent data within some limited time frame. This approximation

has the advantage of being well-defined, deterministic, and, for many applications, emphasizing the recent data the user cares about. However, the data to be included in the window becomes less clear when defined over multiple streams. Other approximation techniques depend on the relative speed of the operations that update data and compute query answers. Fast updates (relative to slow answer computing) suggest batch processing, in which many updates are made before the answer is computed. Fast answer computing (relative to slow updates) suggest stream sampling followed immediately by updates, although such a scenario does not admit provable approximation guarantees [BBD+02].

In addition to the approximation necessitated by the memory challenges inherent in data stream processing, there are also queries that are impossible to answer accurately in the present, blocking query operators and queries based on past data. *Blocking queries* are queries that may not be answered until they have seen the entirety of the data; for example, sorting queries or those involving aggregation (summation, mean, etc.). Approaches to answering these queries include replacing the query with a non-blocking query with an approximately similar result, maintaining an answer given the data seen so far, and reasoning based on an augmented data stream containing assertions about future data. Queries involving past data must rely on data summaries. Creating these summaries is challenging since future queries are unknown and all data may not be stored. Answers to these queries are likely to be approximate. A solution that side-steps this issue is to state to the user that any such queries will consider only the data stream beginning at the time the query is issued [BBD+02].

The literature on sensors and streams is too vast to cover completely here, instead we discuss a small number of relevant papers. Cormode *et al.* [CMZ07] consider a problem over data streams generated by distributed sites. They are the first to consider the the maintenance of the continuous discrete k-center clustering problem and give accuracy guarantees. They propose four methods and compare them theoretically and experimentally. These four algorithms combine local and global methods with the Gonzalez farthest point algorithm [Gon85] and the parallel guessing algorithm. The local algorithms distribute the clustering decisions and then merge to find a global clustering. Cormode *et al.* show that merging α-approximate clusters using a β-approximate technique results in an $(\alpha + \beta)$-approximate clustering. The global algorithms distribute the monitoring of the validity of the current clustering, but assume transmission of that data to a central server for a global recalculation of clustering when required. The Gonzalez farthest point algorithm, used at the distributed sites for local clustering or by the central server for global clustering, sequentially chooses the point farthest from any identified cluster. It is known to give a 2-approximation of the optimal k-center clustering [Gon85]. The parallel guessing algorithm guesses some radius r and creates a new center and marks points within distance r of that center as clustered anytime it encounters a point that is not currently in a cluster. This algorithm is run in parallel for multiple guesses of r and the number of guesses is dependent on the aspect ratio of the point set. The guess with minimum r that still clusters all points into at most k clusters is the resulting clustering. This algorithm is shown to be a $(2 + \varepsilon)$-approximation of the optimal clustering [CMZ07].

Cormode *et al.* show that both local algorithms are $(4 + \varepsilon)$-approximations and both global algorithms are $(2 + \varepsilon)$-approximation of the optimal clustering. Both implementations of the Gonzalez algorithm require $O(n)$ space while both implementations of the parallel guessing algorithm require $O(\frac{k}{\varepsilon} \log \alpha)$ space, where α is the aspect ratio. Communication required is shown to be $O(\frac{km}{\varepsilon} \log \alpha)$ for both versions of the parallel guessing algorithm, where m is the number of distributed sites. However, no communication bounds are known for the Gonzalez local and global algorithms. Experimentally, the communication bounds for the versions based on the Gonzalez algorithm were shown to be worse than those for the local parallel guessing algorithm. The local parallel guessing algorithm was shown experimentally to require communication costs of less than one percent of the cost to send all information to a central server [CMZ07].

Other problems considered over data streams include functional monitoring problems, which keep track of a function output based on the data stream inputs in an online manner. The accuracy of the function value is based on an error parameter ε. Cormode *et al.* [CMY08] considered the problem of monitoring monotone functions over distributed data streams and give worst-case bounds on the number of updates to the function they maintain. Yi and Zhang [YZ09] considered arbitrary d-dimensional functions over a single data stream. They use competitive bounds to show that they update the function they maintain $O(d^2 \log(d\varepsilon))$ times for every one time the function is updated by the optimal algorithm.

2.4 Data Compression and Entropy

The handling of large data sets often requires reducing the necessary storage size through a data compression algorithm. Compression algorithms represent the data set as a single string of information and return a smaller *encoded* string of compressed data. If the encoded string can be restored exactly to its original form, the algorithm is known as *lossless*. If the string cannot be restored exactly after being compressed, the algorithm is known as *lossy*. Shannon's source coding theorem states that in the limit, as the length of a stream of independent, identically distributed (i.i.d.) random variables goes to infinity, the minimum number of required bits to allow lossless compression of each character of the stream is equal to the entropy of the stream [Sha48]. The *entropy* of a string of characters taken from a random source X is defined to be $-\Sigma_x p_x \log_2(p_x)$ where x is an outcome of the random process. The optimal length of an encoded string is equal to the string's entropy. Compression algorithms that achieve this optimum are known as *entropy encoding* algorithms.

Many entropy encoding lossless compression algorithms have been considered including Huffman coding [Huf52], arithmetic coding [Ris76], the Lempel-Ziv dictionary algorithm [ZL78], and the Lempel-Ziv sliding-window algorithm [ZL77]. Huffman coding replaces characters with symbols so that the most probable characters are given the shortest symbols, and the least probable are represented by the longest. Arithmetic compression encodes the entire string in a base so that each character corresponds to a different digit in a fractional number. The entire string is then

translated to a binary number that is precise enough so that the original number can be retrieved. The Lempel-Ziv dictionary algorithm [ZL78] adds matches to a prefix-tree and stores the tree along with an ordered list of pointers instead of the full string. (This algorithm will be used and described in more detail in Chapter 6.)

We will examine the Lempel-Ziv sliding-window algorithm [ZL77] in more depth. The algorithm proceeds by looking for matches between the current time position and some previous time within a given window into the past. The length and position of these multi-character matches are then recorded, which reduces the space of encoding each character. The window moves forward as time progresses. Note that this algorithm operates on the string in an online fashion, containing a valid encoding at any time. The encoding involves splitting the string into a collection of phrases, each of which is then encoded into a tuple. There are two types of tuples. A tuple of the form $'(0, X)'$ indicates a single plaintext character $'X'$, and a tuple of the form $'(1, i, j)'$ indicates a repeated string of length j starting at the i-th preceding character. For example, consider the encoding found using a window size of three on the string $AABBAB$. The encoded string is $(0, A)(1, 1, 1)(0, B)(1, 1, 1)(1, 3, 2)$. Intuitively, it is clear that larger window sizes yield better results since long matches are more likely to be found. In the example, this corresponds to a preference for the pointer $(1, 3, 2)$ and not for the pointers $(1, 1, 1)$ since $(1, 3, 2)$ saves more space.

In 1994, Wyner and Ziv proved that the optimal encoded length for the Lempel-Ziv sliding-window algorithm is achieved by taking the limit as the window size tends to infinity [WZ94]. The precise optimal length is shown to be equal to the entropy of the string. The proof proceeds by showing that the expected num-

ber of bits to encode any character of the string is at most the number of bits to encode the first window plus the normalized expected value of the sum of the match encoding lengths. The latter value is identified using Kac's Lemma, which states that for some character α that has a positive probability $P_0(\alpha)$ of occurring at time 0 (the present), $\Sigma_{i=1}^{\infty} iP_{-i}(\alpha) = 1/P_0(\alpha)$, where $P_{-i}(\alpha)$ is the conditional probability (given that the character at time 0 is α) that the most recent occurrence of α was at time $-i$. Other lemmas are introduced to show that as the length of the string becomes arbitrarily large, the probability that the closest match is farther away than a function dependent on the entropy goes to zero. So the sum of the match encoding lengths, which is equal to the length to encode unmatched phrases plus the length to encode matched phrases, becomes at most the length of the encoding for matched phrases. These are shown to approach the entropy, or information content, of the string.

2.4.1 Empirical Entropy

While entropy is a beautiful theoretical concept, when considering observed streams of data it has two important drawbacks. The first is a consequence of the Wyner and Ziv proof [WZ94]; the entropy encoding bounds on some algorithms only hold as the window size approaches infinity. When considering observed data, it would be better to have the bounds hold within some realistically sized window. The second drawback is a result of the statistical setting in which entropy is considered; an underlying random process must be assumed, and in order to calculate the

entropy we must have access to the associated probability distribution. Again, when collecting data from some observed source we are unlikely to know this information. Empirical entropy was introduced to address these concerns.

Empirical entropy is defined over the observed probabilities of the data and can consider windows of fixed length $k \geq 1$. Given a string X of length T drawn from some alphabet Σ, let $c_0(x)$ be the number of occurrences of $x \in \Sigma^k$ in X and $c(x)$ be the number of occurrences of x in the substring of X not including the final character. Let the observed probability of x appearing in X be denoted $\mathsf{p}_{\mathrm{X}}(x) = c(x)/(T-k)$. Kosaraju and Manzini [KM99] defined the *0th order empirical entropy* of such a string X to be

$$\mathsf{H}_0(X) \;=\; -\sum_{a\in\Sigma} \mathsf{p}_{\mathrm{X}}(a) \log \mathsf{p}_{\mathrm{X}}(a) = -\sum_{a\in\Sigma} \frac{c_0(a)}{T} \log \frac{c_0(a)}{T} \;.$$

The 0th order empirical entropy definition considers only the substrings of length 1 (single characters) and determines the empirical entropy based on the observed probabilities of these characters within the string. The following definition generalizes this to consider substrings of length k. These substrings are analogous to the windows discussed earlier, and allow some history of the string to be taken into account. To quantify this history, we consider the observed probability that some character $a \in \Sigma$ is the character immediately following some substring $x \in \Sigma^k$. Denote this observed probability as $\mathsf{p}_{\mathrm{X}}(a|x) = c(xa)/c(x)$. The *kth order empirical entropy* of a string X is defined by Kosaraju and Manzini [KM99] to be

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

In Chapter 5 we extend these definitions to consider joint empirical entropy, conditional empirical entropy, and empirical independence.

As implied earlier, operating under a fixed and finite window size has real effects when considering an analysis of compression algorithms. In their paper, Kosaraju and Manzini [KM99] consider the analysis of the Lempel-Ziv sliding window algorithm (LZ77) [ZL77] and the Lempel-Ziv dictionary compression algorithm (LZ78) [ZL78] within the concept of λ-optimality. They define an algorithm to be λ-*optimal* if its compression ratio is bounded by $\lambda \mathsf{H}_k(X) + o(\mathsf{H}_k(X))$ for any string X. This definition has the nice property that both low and high entropy strings are guaranteed to be compressed efficiently (previous similar definitions left out the low entropy cases where $\lim_{|X|\to\infty} \mathsf{H}_k(X) = 0$). While, under the Shannon definition of entropy, both LZ77 and LZ78 are optimal entropy encoding algorithms, under this new definition neither is λ-optimal in all cases. LZ77 is shown to be 8-optimal with respect to $\mathsf{H}_0(X)$, but is not λ-optimal with respect to $\mathsf{H}_k(X)$ for any $k \geq 1$. LZ78 cannot be λ-optimal with respect to $\mathsf{H}_k(X)$ for any $k \geq 0$, however when combined with a run-length encoding scheme that efficiently encodes long substrings of identical characters, LZ78 is 3-optimal with respect to $\mathsf{H}_0(X)$. These analyses allow more accurate understandings of the strengths and weaknesses of the different compression schemes. Since this work, empirical entropy has become the standard benchmark for analysis when the realism of the compression scheme is a concern (see, for example, Manzini's analysis of a commonly used compression tool, the Burrows-Wheeler Transform [Man01]). The area of compressed text indexing thus relies heavily on these concepts.

2.5 Compressed Text Indexing

Compressed text indexing (also known as the study of opportunistic data structures or succinct data structures), was first introduced by Ferragina and Manzini in 2000 [FM00]. The goal of such data structures is to compress and store the data in space on the order of its empirical entropy, while still allowing relatively fast query times. (For surveys of this area, see [CHSV10, HSV10, NM07].) The problem considered by Ferragina and Mangini [FM00] with an accompanying opportunistic data structure was the compressed matching problem.

The *compressed matching problem*, introduced by Amir and Benson [AB92], is the problem of, given a string, finding all matching substrings in a compressed text without decompressing it. The compressed matching problem has been studied in numerous sources including [FM00, AB92, ABFC96, FM05]. Ferragina and Manzini's [FM00] original data structure achieved a space bound of $O(\mathsf{H}_k(X)) + o(1)$ bits for some text X of length T and a time bound of $O(|x| + occ \log^{\varepsilon} T)$ where $|x|$ is the length of the pattern being searched for, occ is the number of occurrences of that pattern in X, and $\varepsilon > 0$ is an error parameter. Ferragina and Manzini [FM05] later improved this bound to $5T \cdot \mathsf{H}_k(X) + o(T)$ space and $O(|x| + occ \cdot \log^{1+\varepsilon} T)$ query time or, in a second data structure, $O(T\mathsf{H}_k(X) \log^{\varepsilon} T)$ space and $O(|x| + occ)$ query time. Their data structure relies on the Burrows-Wheeler Transform [BW94], an invertible mapping that transforms the given text into an easier to compress form. In short, it operates by sorting all permutations of the text and then storing only the last character of each sorted version. This rearrangement of the text is generally easier

to compress than the original. Ferragina and Manzini combine some ideas from the Burrows-Wheeler Transform with those of a suffix array to create their *compressed suffix array* that answers queries quickly while taking space on the order of the optimal.

One of the practical successes of this area has been the demonstration of compressed text indexing for XML data. This work relies mainly on paired papers by Ferragina, Luccio, Manzini, and Muthukrishnan [FLMM05,FLMM06]. The first, in 2005, considers the theoretical problem of how to store labeled trees succinctly while still allowing basic tree retrieval operations, such as finding the parent or child of a node and finding all children with a given label [FLMM05]. This paper uses a transform that they dub *xbw*, which was inspired by the Burrows-Wheeler Transform. It creates two arrays from the original data – one captures the structure of the tree and the other the labels of the nodes. The tree structure is captured by annotating nodes with strings containing information about the path from the node to the root. Then nodes are sorted according to these path annotations. These sorted annotations serve as the basis for the tree structure array. Mappings are also introduced to translate from the tree to the array and back. Once these structures are in place, tree queries are supported on top of them. The second paper, in 2006, uses these theoretical ideas to implement a storage and retrieval system for XML data. They show that in practice their storage system takes up to 35% less space and is orders of magnitude faster on some queries.

The query type that is in some ways closest to the query considered in Chapter 6 is that of substring queries. *Substring queries* are given the indexes of some

substring in a compressed text and retrieve the associated substring in its original form. Ferragina and Venturini give a storage scheme that can support these queries and takes space close to the optimal kth order empirical entropy bound and optimal $O(1 + \frac{|x|}{\log_{|\Sigma|} T})$ time, where $|x|$ is the length of the substring being queried, $|\Sigma|$ is the size of the alphabet, and T is the length of the text [FV07]. They achieve this using only binary encodings and tables. This problem has also been considered by [GN06, SG06].

Additionally, the Geometric Burrows-Wheeler Transform [CHSV08] uses range searching techniques and results to better understand text queries by mapping text to a set of points. This transformation is the basis for lower bound results on the compressed matching problem as well as an I/O-efficient version of the compressed matching problem and a version of the compressed matching problem restricted to a substring of the text. A specific type of range searching is discussed in more detail in the following section.

2.6 Approximate Range Searching

In Chapter 6 we introduce range searching over compressed text restricted by both a spatial and temporal range. The temporal structure is inspired by the work on compressed text indexing discussed in the previous section. In this section, we discuss the traditional geometric work on range searching that influences our spatial range structures. Generally, the idea of range searching is, given a point set P and some range R, retrieve the result of some operation over all the points of P that

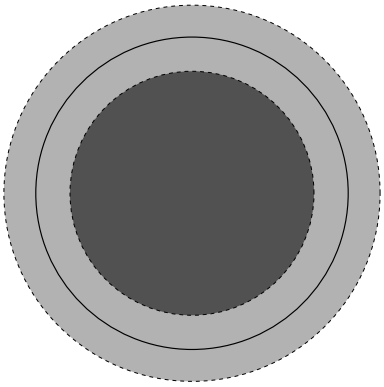

Figure 2.1: An example range with buffer zone is shown. The range is the indicated by the solid edged disk, the inner and outer boundaries are indicated by dashed lines, the required inclusion region is shown in dark grey, and the optional inclusion region is shown in light grey.

lie within R. For example, the range might be some axis-aligned rectangle and the operation might be counting the number of points that lie within it. For surveys of this area, see [Mat94, AE98].

Here, we specifically discuss approximate range searching. Approximate range searching is, given a point set $P \in \mathbb{R}^d$ of size n, some range R, and an error parameter ε, the problem of retrieving the result of an operation over the point set restricted to the range R with error ε. Error has been considered under two models: relative and absolute [AM00, dFM10]. In each, a buffer zone at the edge of the range is created and points strictly on the inside of the zone *must* be included in the operation while points strictly outsize of the zone *must not* be included. Points within the buffer zone can be included, but are not required to be. See Figure 2.1 for an example showing the buffer zone. Under the *relative error model* [AM00], this buffer zone is

an ε multiplicative factor of the size of the original range, while in the *absolute error model* [dFM10], the buffer zone is strictly an ε-width on either size of the range.

One important yet general range to consider within the approximate model is that of *fat convex ranges*. These ranges, which are essentially defined to have bounded aspect ratio [AM00], are used as representations of practical ranges and are appropriate for approximate range searching since ranges with high aspect ratio might have buffer zones that cover inappropriately large portions of the range under the relative error model. Under the absolute error model, da Fonseca and Mount [dFM10] show how to answer approximate range queries for fat simplex ranges with preprocessing time $O((n + 1/\varepsilon^d) \log^{O(1)}(n + 1/\varepsilon^d))$, space $O((1/\varepsilon^d) \log O(1)(1/\varepsilon^d)$, and query time $O((1/\varepsilon^{d-2}+\log(1/\varepsilon)) \log^{O(1)}(1/\varepsilon^{d-2}+\log(1/\varepsilon)))$. Within the relative error model, Arya and Mount [AM00] show how to answer approximate range queries for convex ranges in preprocessing time $O(n \log n)$, space $O(n)$, and query time $O(\log n + (1/\varepsilon)^{d-1}$, they also show that this query time matches the lower bound.

In order to achieve their results in the relative error model, Arya and Mount [AM00] make use of a data structure called the balanced box-decomposition tree or *BBD-tree* introduced by Arya, Mount, Netanyahu, Silverman, and Wu [AMN$^+$98]. The BBD-tree creates a hierarchical decomposition of space and in that way is similar to kd-trees [Ben75] and quadtrees [Sam84]. In a BBD-tree, the cells associated with nodes may be either boxes or the set theoretic difference of two boxes, one contained within the other. Each branching choice is either a *split* of a single box along an axis-aligned plane or a *shrink* that partitions the points by an inner box and the difference of that box and the bounding box of the current cell. With these

simple operations, the BBD-tree is able to achieve two important properties: an $O(\log n)$ tree height and a bounded aspect ratio for the nodes' associated cells. (We make use of this structure in Chapter 6.)

Chapter 3

Approximation Algorithm for the Kinetic Robust K-Center Problem

In this chapter, we consider two complications that frequently arise in real-world applications, motion and the contamination of data by outliers. We consider a fundamental clustering problem, the k-center problem, within the context of these two issues. We are given a finite point set S of size n and an integer k. In the standard k-center problem, the objective is to compute a set of k center points to minimize the maximum distance from any point of S to its closest center, or equivalently, the smallest radius such that S can be covered by k disks of this radius. In the discrete k-center problem the disk centers are drawn from the points of S, and in the absolute k-center problem the disk centers are unrestricted.

We generalize this problem in two ways. First, we assume that points are in continuous motion, and the objective is to maintain a solution over time. Second, we assume that some robustness parameter $0 < t \leq 1$ is given, and the objective is to compute the smallest radius such that there exist k disks of this radius that cover at least $\lceil tn \rceil$ points of S. We present a kinetic data structure (in the KDS framework) that maintains a $(3+\varepsilon)$-approximation for the robust discrete k-center problem and a $(4+\varepsilon)$-approximation for the robust absolute k-center problem, both under the

assumption that k is a constant. We also improve on a previous 8-approximation for the non-robust discrete kinetic k-center problem, for arbitrary k, and show that our data structure achieves a $(4+\varepsilon)$-approximation. All these results hold in any metric space of constant doubling dimension, which includes Euclidean space of constant dimension.

3.1 Introduction

In the design of algorithms for optimization problems in real-world applications, it is often necessary to consider the problem in the presence of complicating issues. We consider two such issues here. The first is the processing of *kinetic data*, that is, objects undergoing continuous motion. The second confounding issue arises when the data is heterogeneous and the statistical trends of the majority may be obscured by the deviant behavior of a minority of points, called outliers. The objective is to produce a solution to some given optimization problem that is *robust* to corruption due to outliers.

These two issues have been considered individually in the context of kinetic data structures and robust statistics, respectively, but no published work to date has involved the combination of the two. The combination of these two issues presents unique challenges. One reason is the different effects these two issues have on the structure of algorithmic solutions. Kinetic data structures are typically concerned with handling local properties involving the interaction of a small number of objects. On the other hand, algorithmic solutions in robust statistics involve global

characteristics of the data set, such as identifying which subset of points constitutes the majority. In this chapter we consider a well-known clustering problem, called the k-center problem, in the context of kinetic data and outliers. We refer to it as the kinetic robust k-center problem.

Many frameworks have been proposed for handling kinetic data [GJS96, Ata85, ST95, ST96]. We assume a common model for processing points in motion, called *kinetic data structures* (KDS), which was proposed by Basch, Guibas, and Hershberger [BG99]. In this model the motion of each point is given by a piecewise function of constant algebraic degree, called a *flight plan*. KDSs track specific properties of moving points. This is done through a set of boolean conditions, called *certificates*, and a corresponding set of update rules. Certificates guarantee geometric relations necessary to a particular problem's solution, and update rules specify how to respond whenever a certificate fails. The KDS framework has become the standard approach for computing discrete structures for kinetic data sets because it provides a general and flexible framework for the development of algorithmic solutions that are both provably correct and efficient. Examples include maintaining convex hulls [BG99], Voronoi diagrams [AGMR98], and minimum spanning trees on geometric graphs [BGZ97].

Recall from Chapter 2.2 that there are four criteria under which the computational cost of a KDS is evaluated: responsiveness, efficiency, compactness, and locality [Gui04]. *Responsiveness* measures the complexity of the cost to repair the solution after a certificate fails. *Efficiency* measures the number of certificate failures as compared to the number of required changes to the solution as the points

move. *Compactness* measures the size of the certificate set. *Locality* measures the number of certificates in which each point participates. (More information about kinetic data structures is provided in Chapter 2.2.)

The other issue of interest in this chapter is robustness. The study of statistical estimators that are insensitive to outliers is the domain of *robust statistics* [RL87]. Robust statistics have been extensively studied in mathematics, operations research, and computer science. Robustness in the presence of outliers is important, for example, when considering heterogenous populations that contain isolated and unusual data points. It is also useful when considering business and public-service applications. Since cost is an important factor, it is desirable to provide a service to a large segment of the population, while limiting expensive service costs involving a small fraction of outliers. Charikar *et al.* [CKMN01] explored the robust facility location problem, which determines the locations of stores while minimizing the distance from customers to the stores and the total cost of opening facilities. In such a model it is unprofitable to open a new facility to service a small number of isolated customers. Degener *et al.* [DGL10] gave a deterministic algorithm for the kinetic version of the non-robust facility location problem and Agarwal and Phillips [AP08] gave a randomized algorithm for the robust 2-center problem with an $O(nk^7 \log^3 n)$ expected execution time.

We consider a clustering problem that involves a combination of these two important elements. Clustering is a frequently studied problem in operations research and computer science. Common formulations include k-center, k-means, and facility location problems [KH79, HS85, FG88, Ple87]. The (standard) *k-center problem* is

defined as follows: Given a set of n points, find k center points that minimize the maximum distance (called the *radius*) from any point to its closest center. In the *discrete version* the centers must be drawn from the original n points. In contrast, in the *absolute version* the centers may be arbitrary points in space [KH79]. Unless otherwise stated, we will assume the discrete version of the problem.

Kariv and Hakimi [KH79] proved that the discrete and absolute versions of the k-center problem are NP-hard in a graph-theoretic context (for arbitrary k). The problem of finding a $(2-\varepsilon)$-approximation for the discrete k-center problem is NP-complete (also in a graph-theoretic context) [Hoc95, Ple80]. The problem in the Euclidean metric cannot be approximated to within a factor of 1.822 (assuming P $\neq$ NP) [FG88]. Demaine *et al.* [DFHT05] give algorithms for the k-center problem on planar graphs and map graphs that achieve time bounds exponential in the radius and k.

Since the k-center problem is NP-hard for arbitrary k or exponential in k for fixed k, we consider approximation algorithms. An algorithm provides a *c-approximation* to the k-center problem if the radius chosen for the k centers is no more than c times the optimal radius. Feder and Greene [FG88] gave a 2-approximation for the geometric k-center problem and Hochbaum and Shmoys [HS85] and Gonzalez [Gon85] gave 2-approximation algorithms for the graph-theoretic version of the k-center problem, both for arbitrary k. In light of the above lower bound of $(2-\varepsilon)$, these approximation algorithms provide the best possible approximation bounds.

The *robust k-center problem* generalizes the k-center problem to handle outliers

by allowing flexibility in the number of points that satisfy the distance criteria. In our formulation we are given a set of n points, an integer k, and a threshold parameter t, where $0 < t \leq 1$. The objective is to compute the smallest radius r such that there exist k disks of radius r that cover at least $\lceil tn \rceil$ points. The non-robust version arises as a special case, when $t = 1$.

Since the robust k-center problem is a generalization of the non-robust version, the 1.822 approximation lower bound [FG88] for the Euclidean context holds for the robust k-center problem as well (assuming P $\neq$ NP). Charikar *et al.* [CKMN01] showed that in the graph-theoretic context, the robust k-center problem with forbidden centers (in which some locations cannot be chosen as centers), has a lower bound of $3 - \varepsilon$. They also gave a 3-approximation algorithm for the robust k-center problem. Chen gave a constant factor approximation algorithm for the robust k-median problem [Che08].

The (non-robust) kinetic k-center problem is a generalization of the static version, so again the 1.822 approximation lower bound [FG88] holds. No other lower bounds are known for the kinetic problem. Gao, Guibas, and Nguyen [GGN06] give an 8-approximation algorithm for the kinetic discrete k-center problem. Har-Peled handles the discrete and absolute kinetic k-center problems with an $O(nk)$ time algorithm, which creates a larger static set of centers that is competitive at any time [HP04].

The kinetic robust k-center problem has not been studied before, but many application domains involving moving points benefit from robust clustering calculations. These include the segmentation problem in vision, which attempts to separate

meaningful parts of a moving image [CTS97, DM98, Wan98]; context-aware applications, which run on mobile devices that are carried by individuals and interact with the environment and each other [WER07]; and traffic detection and management, which we use as our main motivating example. Traffic detection has been studied extensively with tactics that include using sensors [KEK+98, Gri98, SS07], knowledge-based systems [CHM95], and individual vehicle monitoring (e.g., car GPS navigation systems) [Gil06, ACI+00, Gol99]. Our model assumes individual vehicle monitoring with the assumption of a flight plan provided by the navigation system and a desired number, k, of congested areas to monitor.

Throughout this chapter, we will assume that the point set S resides in a space of *constant doubling dimension*. We define the *disk* of radius r centered at point u to be the set of points of S whose distance from u is less than or equal to r. A metric space is said to have constant doubling dimension if any metric disk of radius r can be covered by at most a constant number, λ, of disks of radius $r/2$. Euclidean space of constant dimension is an example. The doubling dimension d is defined to be $\log_2 \lambda$ [KL04]. To generalize the concept of a metric space to a kinetic context, we assume access to functions giving the distance between two points at a given time and the earliest future time at which two points will be within some given distance.

3.1.1 Contributions

As mentioned above, we present the first concurrent consideration of two practical domains, robust statistics and kinetic data structures, and an approximation

algorithm and corresponding efficient kinetic data structure to solve the kinetic robust k-center problem. Our algorithm approximates the static k-center, kinetic k-center, and robust k-center problems as well, since all are special cases of our problem.

The input consists of a kinetic point set S in a metric space of constant doubling dimension d, the number of centers k, a robustness threshold $0 < t \leq 1$, and an approximation parameter $\varepsilon > 0$. Some of our complexity bounds depend on the aspect ratio of the point set, which is defined as follows in a kinetic context. Let $d_{\min}$ and $d_{\max}$ be lower and upper bounds, respectively, on the distance between any two points over the entire motion. The *aspect ratio*, denoted by α, is defined to be $d_{\max}/d_{\min}$. We obtain a $(3+\varepsilon)$-approximation for the static and kinetic forms of the robust discrete k-center problem and a $(4+\varepsilon)$-approximation for the absolute version of the robust k-center problems. Note that the first bound improves upon the 8-approximation for the kinetic discrete k-center problem as given by Gao, Guibas, and Nguyen [GGN06] and generalizes it to the robust setting. However, due to complications arising from the need for robustness, our result assumes that k is constant, while theirs holds for arbitrary k. We improve their result for the non-robust kinetic problem for arbitrary k by showing that our data structure achieves a $(4+\varepsilon)$-approximation, while maintaining the same quality bounds as their KDS (see Section 3.5). To our knowledge, our kinetic robust algorithm is the first approximation algorithm for the kinetic absolute k-center problem (even ignoring robustness). We give an example in Section 3.3.3.1 to show that our $(3+\varepsilon)$-approximation for the robust discrete k-center problem is tight.

The KDS used by our algorithm is efficient. In Section 3.4.3 we will establish bounds of $O((\log \alpha)/\varepsilon^d)$ for locality and $O(n/\varepsilon^{d+1})$ for compactness. Our responsiveness bound is $O((\log n \log \alpha)/\varepsilon^{2d})$, implying that the data structure can be updated quickly. Our efficiency bound of $O(n^2(\log \alpha)/\varepsilon)$ is reasonable since the combinatorial structure upon which our kinetic algorithm is based requires $\Omega(n^2)$ updates in the worst case (even for the non-robust case) [GGN06], so any approach based on this structure requires $\Omega(n^2)$ updates.

3.2 Weak Hierarchical Spanner

Our approach is to extend a spanner construction for kinetic data structures developed by Gao *et al.* [GGN06], which they call a *deformable spanner*. This kinetic structure is defined assuming a point set S in $\mathbb{R}^d$ for any fixed d, but the construction generalizes easily to any metric space of constant doubling dimension. The spanner is based on a hierarchical clustering involving a sparse subset of points, called centers. To avoid confusion with the use of the term *center* as a cluster center, henceforth we use the term *node* for a point in the discrete hierarchy, and the term *center* when referring to the center of a disk in the solution to the k-center problem. We will use the term *point* to refer to an element of S. Each node is associated with a point of S. Because of the close relationship between nodes and the associated points, we will sometimes blur this distinction, for example, by referring both to a node u in the spanner and point u in S, or referring to the distance between two nodes (by which we mean the distance between their associated points).

Recall the following properties of the deformable spanner [GGN06] as presented in Chapter 2.2:

- The hierarchy has a height $O(\log \alpha)$.
- Any node in S_0 is within distance 2^{i+1} of its ancestor in level S_i.

Gao *et al.* [GGN06] showed that the hierarchy of discrete centers could be used to define a spanner for the point set S. Given a parameter $\gamma \geq 1$, called the *stretch factor*, a *γ-spanner* for a point set is a graph where the points are the vertices and the edge set has the property that the shortest path length in the graph between any two points is at most γ times the metric distance between these points. Given a user-supplied parameter $c > 4$, two nodes u and v on level i of the hierarchy are said to be *neighbors* if they lie within distance $c \cdot 2^i$ of each other. For each pair of neighboring nodes in the hierarchy, an edge is created between their associated points. Gao *et al.* show that the resulting graph is a spanner for S, and they establish a relationship between the value of c and the resulting stretch factor. Throughout, we will assume that $c = 8$, which implies that the resulting graph is a 5-spanner. Although we will use the term *spanner* when referring to our structure, we will not be making use of spanner properties directly in our results.

The KDS presented in [GGN06] for the deformable spanner is shown to be efficient, local, compact, and responsive. The algorithm maintains four types of certificates: parent-child certificates, edge certificates, separation certificates, and potential neighbor certificates. We will use these same certificates in our algorithm, but with different update rules.

There is one additional difference between our structure and that of Gao *et al.* In order to obtain our stronger approximation bounds, we will need to make a number of copies of this structure, each with slightly different parameter settings. The number of copies, which depends on the approximation parameter ε, will be denoted by $s(\varepsilon)$, or simply s whenever ε is clear from context. Its value will be given in the next section. Recall that the ith level of the hierarchy of discrete centers is naturally associated with the distance 2^i (both as the covering radius and as the separation distance between nodes). The lowest level of the hierarchy is associated with the distance $2^0 = 1$, which we call the *base distance* of the hierarchy. Each copy in our structure will employ a different base distance. In particular, for $0 \leq p < s$, let $b_p = (1 + \frac{p}{s})$. Observe that $1 \leq b_p < 2$. In our structure, the ith level of the pth spanner copy, denoted $S_i(p)$, will be associated with the distance $b_p 2^i$. The neighbors of a node are defined to be those nodes within distance $c \cdot b_p 2^i$, rather than $c \cdot 2^i$.

To simplify our algorithm presentations, unless otherwise stated, we will assume that $p = 0$, and so $b_p = 1$. There is no loss of generality in doing so, because an equivalent way of viewing the variation in the base distance is to imagine that distances have been scaled. In particular, when dealing with the pth copy, imagine that all distances have divided by b_p, and the hierarchy is then constructed on the scaled points using the default base distance of 1.

Since the lowest level of the hierarchy, S_0 is required to satisfy the requirement that the distance between any two points is at least $2^0 b_p$, it will be useful to assume that distances have been scaled uniformly so that $d_{\min} = 2$.

3.3 Robust K-Center Algorithm

Gao *et al.* [GGN06] gave an 8-approximation for the non-robust discrete version of the kinetic k-center problem (for arbitrary k). In Section 3.5 we improve this to a $(4+\varepsilon)$-approximation for arbitrary k. In this section we present a $(3+\varepsilon)$-approximation algorithm for the robust discrete version of this problem and a $(4+\varepsilon)$-approximation algorithm for the robust absolute version, both for constant k. Recall that the non-robust version is a special case of the robust version (by setting $t = 1$), so these algorithms also apply to the non-robust case.

3.3.1 Intuitive Explanation

For the sake of intuition regarding some of the more complex technical elements of our algorithm we first present the algorithm by Charikar *et al.* [CKMN01] for the static robust discrete k-center problem, which is the basis for our algorithm, and we explain why it cannot be applied directly in the kinetic context. Henceforth we refer to it as the *expanded-greedy algorithm*.

The algorithm is given as input a point set S of cardinality n, a number of centers k, and a robustness threshold t. Radius values are chosen in a parametric search so that all potential optimal values are considered. For a given target radius r and for each point $v \in S$, we define the *greedy disk* G_v to be a disk of radius r centered at v, and the *expanded disk* E_v to be the disk of radius $3r$ centered at v. (We sometimes let G_v and E_v represent the geometric disk and sometimes the subset of S contained within the disk. It will be clear from context which

interpretation is being used.) Initially all the points of S are labeled as uncovered. The algorithm repeatedly picks the node v such that the greedy disk G_v contains the most uncovered points, and it then marks all points within the expanded disk E_v as covered. If after k iterations it succeeds in covering at least $\lceil tn \rceil$ points, it returns successfully, and otherwise it fails. The algorithm is presented in Figure 3.1.

Figure 3.1: expanded-greedy(S, k, t, r)

$n \leftarrow |S|$

$C_r \leftarrow \emptyset$ (C_r will hold the set of k centers being created for radius r)

$V \leftarrow S$ (V holds the set of candidate centers)

for each $v \in V$

construct G_v and E_v and compute count $|G_v|$ of uncovered points in S

within distance r of v

for $j = 1$ to k, let v_j be the $v \in V$ with largest $|G_v|$

add v_j to C_r and mark all points in E_v as covered

for all $v \in V$, update $|G_v|$

if at least $\lceil tn \rceil$ points are covered, return C_r, and otherwise return "failure"

Figure 3.1: An overview of the expanded-greedy algorithm [CKMN01] for a single radius value r.

The analysis of this algorithm's approximation bound (which will be presented later in Section 3.3.3) uses a charging argument, where each point covered by an optimal disk is charged either to the expanded disk that covers it or to a non-overlapping greedy disk [CKMN01]. The proof relies on two main points. First, greediness implies that any optimal disk that does not overlap any greedy disk

cannot cover more points than any greedy disk. Second, if an optimal disk does overlap some greedy disk G_v, then the expanded disk E_v covers all the points of this optimal disk. This implies that optimal disks cannot be repeatedly "damaged" by greedy disks.

To better motivate our kinetic algorithm, it will be helpful to first consider a very simple static algorithm, which does not achieve the desired approximation bound, but we will then show how to improve it. This initial algorithm applies the expanded-greedy algorithm to individual levels of the discrete hierarchy described in Section 3.2. It starts at the highest level of the spanner and works down. For each level i, it runs the expanded-greedy algorithm with radius $r = 2^i$, considering just the nodes at this level as possible centers. It returns the set of centers associated with the lowest level that succeeds in covering at least $\lceil tn \rceil$ points.

There are, however, some assumptions inherent to the expanded-greedy algorithm that do not hold for this simple static algorithm. Let us consider each of these assumptions and our approach for dealing with them.

All important radii will be considered.

The proof of the expanded-greedy algorithm relies on the possibility for the algorithm to pick a node and cover all points within the optimal radius of that node. However, in this initial algorithm, if the optimal radius is slightly larger than 2^i then our algorithm would be forced to choose the centers at the next higher level, nearly doubling the radius value.

As mentioned at the end of Section 3.2, we solve this problem by creating multiple spanners with base distances that vary, thus partitioning the interval between 2^i and 2^{i+1} into $O(1/\varepsilon)$ subintervals (see Section 3.4). The algorithm is then applied to all spanners, and the best result over all is chosen.

All points in S are candidate centers.

Our simple static algorithm considers only nodes in level i as possible centers, and so points of S that do not reside on level i are excluded from consideration. If some of these excluded centers are in the optimal solution, then the algorithm might need to substitute a node at level i at distance up to 2^{i+1} from an optimal center, which would require the need for a larger radius.

It would be unacceptably slow in the kinetic context to consider all the points of S. Our solution instead is to take candidate centers from level $i - l - 1$, for a suitably chosen l (whose value will depend on ε). We will show that, for each optimal center, there is at least one candidate point that is close enough to enable us to obtain our approximation bounds. We are now able to cover all of the points covered by an optimal solution since the optimal center is a descendant of some center in this lower level.

$|G_v|$ and $|E_v|$ are known exactly.

For the static case, the algorithm of Charikar *et al.* [CKMN01] accurately counts the number of points within the greedy and expanded radii of each node.

However, maintaining these counts in a kinetic context would require keeping certificates between each point in S and all potential covering centers. This would increase the compactness and locality complexities (presented later in Section 3.4.3) by an unacceptable amount.

The hierarchical spanner structure allows us to count the number of points in a *fuzzy greedy disk* in which all points within some inner distance are guaranteed to be counted and no points outside of some outer distance are counted. We call this *range sketching*. Due to fuzziness, some of the counted points may lie outside the greedy radius, but we can increase the expanded radius slightly so that any optimal disks affected by this fuzziness are still fully covered by the expanded disk.

3.3.2 Preconditions

In this section we describe the data that we maintain in our kinetic algorithm in order to produce an approximate solution to the robust k-center problem. It will simplify the presentation to assume for now that the points are static and rely on the description of the kinetic data structure in Section 3.4 for proof that these values are maintained correctly in the kinetic context.

Recall that our construction involves multiple copies of the spanner using various base distances. From the real parameter $\varepsilon > 0$ we derive two additional integer parameters $s(\varepsilon) = \lceil 10/\varepsilon \rceil$ and $l(\varepsilon) = 4 - \lfloor \log_2 \varepsilon \rfloor$ (abbreviated respectively as s and l whenever ε is clear). These values will be justified in the analysis appearing in the proof of Theorem 4.2. The value s represents the number of spanners we

maintain, and l determines the number of levels of the hierarchy that we will descend at each step of the algorithm in order to find candidate centers. Observe that $s = O(1/\varepsilon)$ and $l = \log_2(1/\varepsilon) + O(1)$.

Our algorithm depends on a number of radius values, each of which is a function of the level i, the spanner copy $0 \leq p < s(\varepsilon)$, and $l(\varepsilon)$. Given ε, i, and p, we define the *greedy radius* and the *expanded radius* to be, respectively

$$g_i(\varepsilon, p) \;=\; 2^i \left(1 + \frac{p}{s}\right)(1 + 3 \cdot 2^{-l}) \qquad \text{and} \qquad e_i(\varepsilon, p) \;=\; 3g_i(\varepsilon, p).$$

We also define two slightly smaller radii

$$g_i^-(\varepsilon, p) \;=\; 2^i \left(1 + \frac{p}{s}\right)(1 + 2^{-l}) \qquad \text{and} \qquad e_i^-(\varepsilon, p) \;=\; 3g_i^-(\varepsilon, p).$$

When ε and p are clear from context, we abbreviate the values as g_i, e_i, g_i^-, and e_i^-, respectively. Given the important role of level $i - l - 1$ in our constructions, when l is clear from context, we define $i^- = \max(0, i - l - 1)$, and then use i^- in these contexts.

Our algorithm maintains the following information:

- For each node u at level i of the spanner, we maintain:
  - The point of S associated with u, and conversely the nodes associated with each point of S.
  - The parent, children, and neighbors of u.
  - The number of points of S that are descendants of u.
- For each node u at level i^- of the spanner, we maintain:

  - The number of points lying approximately within distance g_i of u.
  - The number of points lying approximately within distance e_i of u.

  (The sense of approximation will be defined formally in Section 3.3.3.)

- For each level i in the discrete hierarchy we maintain:
  - A priority queue associated with level i storing the nodes of S_i– ordered by the counts of points within distance g_i (approximately) as described above. (The use of this priority queue will be clarified in Section 3.4.)

3.3.3 The Discrete Problem

Recall that our algorithm takes as input the set S of n points and additional parameters ε (approximation parameter), k (number of centers), and t (robustness threshold). Also recall that α denotes S's aspect ratio bound. The algorithm makes use of two parameters s and l, which are both functions of ε. It returns a set of k centers chosen from S for a $(3+\varepsilon)$-approximation of the optimal solution to the kinetic, robust k-center problem.

Algorithm Overview

The algorithm is applied to all s spanners in our structure. For each spanner, it applies a binary search over its levels, to determine the smallest covering radius. At each level i, the k best centers for that level are calculated using the per-level subroutine described later in this section. If this algorithm returns in failure (meaning that it failed to cover at least $\lceil tn \rceil$ points of S), the binary search continues by

considering higher levels of the spanner (larger covering radii); otherwise, it continues to search through the lower levels (smaller radii). On termination of the binary search, the k centers resulting from the search are stored as representatives for that spanner. The k centers with minimum radius e_i out of all s spanners are output as the final solution.

Range Sketching

In order to maintain the counts described as necessary preconditions, we need to efficiently count the number of points of S within a fuzzy disk, which we call a *range-sketch query*. We are given a pair of concentric disks (B^-, B), where $B^- \subseteq B$, and returns a count including all points within B^- and no points outside of B [dFM10]. Given a node v at level i^-, let G_v and G_v^- denote the disks centered at v of radii g_i and g_i^-, respectively, and let E_v and E_v^- denote the disks centered at v of radii e_i and e_i^-, respectively. In our algorithm we will apply range-sketch queries of two types, (G_v^-, G_v) and (E_v^-, E_v). The answer to the query (B^-, B) will be represented as a collection of spanner nodes, all from the same level of the spanner, where the desired count is the total number of points descended from these nodes. This collection of nodes will be denoted by $\mu(B)$. See Figure 3.2 for an illustration.

We answer a range-sketch query by first identifying an easily computable superset of nodes covering the query region, and then pruning this to form the desired set of nodes. To determine this superset of $\mu(G_v)$ (or $\mu(E_v)$) we first develop the following lemmas. Recall from Section 3.2 the concept of a node's neighbors, and

Figure 3.2: Example range-sketch query

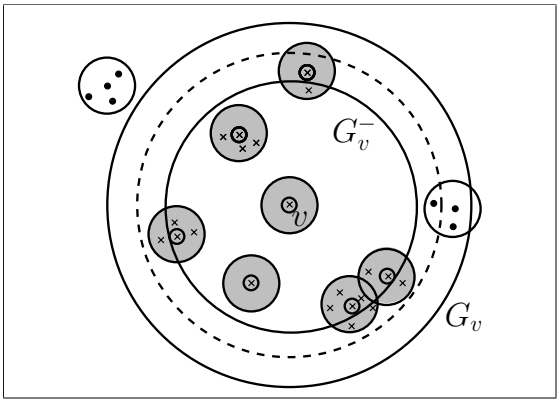

Figure 3.2: For $v \in S_{i^-}$ the range-sketch query returns $\mu(G_v)$, all the nodes of S_{i^-} that lie within G_v^- and no nodes with any descendants that lie outside of G_v. These nodes are circled and disks are shaded. The points drawn as "$\times$" are in $D(\mu(G_v))$ and are counted for the priority queue. The dashed circle has radius $(g_i^- + g_i)/2$.

---

let $c = 8$ denote the parameter used in the definition. More formally, given a subset U of nodes at some level of the spanner, let

$$N(U) \;=\; \bigcup_{u \in U} \left(\{u\} \cup \mathit{neighbors}(u)\right).$$

Given a node v and $h \geq 0$, we define its *h-fold neighbor set* to be:

$$N^{(h)}(v) \;=\; \begin{cases} \{v\} & \text{if } h = 0, \\ N(N^{(h-1)}(v)) & \text{otherwise.} \end{cases}$$

Given a set of nodes U, let $D(U)$ denote the set of descendants of those nodes, and let $D^{(h)}(v) = \bigcup_{u \in N^{(h)}(v)} D(u)$.

Lemma 3.1. *For any node v in S_i and any $h \geq 1$, all the points of S that lie within distance $h \cdot 2^{i+3} - 2^{i+1}$ of v are in $D^{(h)}(v)$.*

Proof. Under our assumption that $c = 8$, $N(v)$ contains all the nodes in level i that are within distance $c \cdot 2^i = 2^{i+3}$ of v. (Recall that we consider the case $p = 0$.) By

induction, $N^{(h)}(v)$ contains all the nodes in level i that lie within distance $h \cdot 2^{i+3}$ of v, and thus, any node u in level i that is not in this set is at distance greater than $h \cdot 2^{i+3}$ from v. Recall that, from basic spanner properties, all of u's descendants are within distance 2^{i+1} of u. It follows that $D^{(h)}(v)$ contains all the points of S within distance $h \cdot 2^{i+3} - 2^{i+1}$ of v. □

The above lemma provides a way to determine a set of nodes that cover all the points of S lying within a given distance of a given node. Using this, we can identify a set of nodes at level i^- whose descendants contain a superset of the points for the range-sketch queries of interest to us.

Lemma 3.2. *Let $h(m) = m(2^{l-2}+1)$. Then for any node v in S_{i^-}:*

(i) The descendants of $N^{h(1)}(v)$ contain all points within distance g_i of v.

(ii) The descendants of $N^{h(3)}(v)$ contain all points within distance e_i of v.

(iii) The descendants of $N^{h(4)}(v)$ contain all points within distance $e_i + g_i$ of v.

Proof. By straightforward manipulations (and under our assumption that $p = 0$), each distance can be rewritten as follows:

(i) $g_i = 2^i(1 + 3 \cdot 2^{-l}) = (2^{l-2}+1)2^{(i^-+3)} - 2^{(i^-+1)}$

(ii) $e_i = 3 \cdot 2^i(1 + 3 \cdot 2^{-l}) < 3(2^{l-2}+1)2^{(i^-+3)} - 2^{(i^-+1)}$

(iii) $e_i + g_i = 4 \cdot 2^i(1 + 3 \cdot 2^{-l}) < 4(2^{l-2}+1)2^{(i^-+3)} - 2^{(i^-+1)}$

The proof follows from Lemma 3.1 applied at level i^-. □

We will apply Lemma 3.2 as a subroutine in our range-sketching procedure. Using the lemma, we first identify a set of nodes whose descendants provide a superset of the points that might contribute to the query result (depending on the radius of interest, g_i, e_i, or $g_i + e_i$), and we then prune this set by eliminating those nodes whose covering disk lies entirely outside the inner disk. To see how to perform this pruning, recall that if u is a node at level i^-, its descendants all lie within distance $2^{(i^-+1)} = 2^{i-l}$ of u. (Note that if $i^- = 0$, then the only descendant is u itself). Thus, if $\|uv\| > (g_i^- + g_i)/2$, then (under our assumption that $p = 0$) it is easy to verify (by the definitions of g_i and g_i^-) that every descendant u' of u satisfies

$$\|u'v\| \;>\; \frac{g_i^- + g_i}{2} - 2^{i-l} \;=\; g_i^- + \frac{g_i - g_i^-}{2} - 2^{i-l} \;=\; g_i^-,$$

and so u' may be omitted from the range-sketch result. Conversely, if $\|uv\| \leq (g_i^- + g_i)/2$, then it is easy to verify that every descendant u' of u satisfies $\|u'v\| \leq g_i$, and so u' may contribute to the range-sketch result. Given this observation, the code is given in Figure 3.3 for the special case of the greedy disk G_v. It returns a set $\mu(G_v)$ of nodes whose descendants satisfy the requirements of the range sketch. The desired count is the sum of the numbers of descendants of these nodes, $|D(\mu(G_v))|$. The procedure returns both the set of nodes and the sum.

Recall from our earlier description of the expanded-greedy algorithm, that whenever a center v is added to the solution at level i, the points lying within the expanded disk E_v are marked as covered. In a kinetic setting it is too expensive to mark these points explicitly. Our approach instead will be to modify the counts

Figure 3.3: range-sketch$_G$(node v, level i)

$sum \leftarrow 0$

$U \leftarrow$ result of Lemma 3.2 part (i) applied to v

for each $u \in U$

    if $\|uv\| > \frac{g_i^- + g_i}{2}$ then $U \leftarrow U \setminus u$

    else $sum \leftarrow sum + |D(u)|$

return (U, sum)

Figure 3.3: The range-sketch and counting subroutine for the $\mu(G_v)$ query. To answer the $\mu(E_v)$ query, all references to $\mu(G_v)$ change to $\mu(E_v)$, g_i^- and g_i to e_i^- and e_i respectively, and Lemma 3.2 part (ii) is used. We will call the range-sketching routine shown here *range-sketch*$_G$ and the one created through these substitutions *range-sketch*$_E$.

Figure 3.4: update-greedy-disks(node v, level i)

```
U ← result of Lemma 3.2 part (iii) applied to v
(μ(E_v), |E_v|) ← range-sketch_E(v, i)
for each u ∈ U
    if ‖uv‖ ≤ e_i + g_i
        (μ(G_u), |G_u|) ← range-sketch_G(u, i)
        for each w ∈ μ(E_v)
            if w is unmarked and w ∈ μ(G_u)
                |G_u| ← |G_u| − |D(w)|
                update u's position in the priority queue
```

Figure 3.4: Subroutine to update greedy disk counts that calls the range-sketch subroutine shown in Figure 3.3.

associated with each greedy disk that overlaps E_v. In particular, we apply range sketching to determine the nodes u in level i^- whose greedy disk G_u overlaps E_v, and for each unmarked node w in their common intersection, we decrease $|G_u|$ by the weight $D(w)$. The procedure is given in Figure 3.4. To prevent nodes from being counted twice, we mark all these nodes w after E_v has been processed (see Figure 3.5).

Main Subroutine and Analysis

We now have the tools needed to introduce the main subroutine for the algorithm. Recall that the input consists of the point set S and parameters k, t, ε. The quantities s and l depend on ε and represent the number of spanners and

the number of levels of resolution, respectively. The current spanner is indexed by $0 \leq p < s$, and the current level of the discrete hierarchy is i. The per-level subroutine (presented in Figure 3.5) calculates the candidate list of k centers for a given spanner p and level i. It is called from the algorithm overview presented earlier.

Figure 3.5: per-level-subroutine(S, k, t, ε, p, level i)

$n \leftarrow |S|$

$C_{p,i} \leftarrow \emptyset$ ($C_{p,i} = \{k$ centers being created for spanner p and level $i\}$)

$V \leftarrow S_{i^-}(p)$ ($V = \{$candidate centers$\}$, $S_{i^-}(p) = \{$level i^- of spanner $p\}$)

for each $v \in V$

  $(\mu(G_v), |G_v|) \leftarrow$ range-sketch$_G(v, i)$

  $(\mu(E_v), |E_v|) \leftarrow$ range-sketch$_E(v, i)$

for $j = 1$ to k

  Let v_j be the $v \in V$ with the largest $|G_v|$

  $C_{p,i} \leftarrow C_{p,i} \cup \{v_j\}$

  update-greedy-disks(v_j, i)

  for each uncovered $w \in \mu(E_v)$, mark w as "covered"

if at least $\lceil tn \rceil$ points are covered return $(C_{p,i}, e_i)$ otherwise return "failure."

Figure 3.5: The per-level subroutine for spanner p and level i of the kinetic robust k-center algorithm. This subroutine calls the range-sketch and update-greedy-disk subroutines shown in Figures 3.3 and 3.4 respectively.

Before giving the kinetic version of the algorithm, we first describe the static version, and we focus on just one stage of the algorithm. In the static context this algorithm requires preprocessing to create the priority queue and perform range sketching to determine initial counts for G_v and E_v in all levels of all spanners.

We comment that this can be done in time $O((1/\varepsilon)^d n \log n \log \alpha)$ since there are n points, priority queue insertions take time $O(\log n)$, centers are calculated for $O(\log \alpha)$ levels, and range sketching takes time $O(1/\varepsilon^d)$ as shown by the following lemma.

Lemma 3.3. *Range sketch queries for level i involving any of the distances g_i, e_i or $g_i + e_i$ can be answered in time $O(1/\varepsilon^d)$.*

Proof. The running time of a range-sketch query for a node v is dominated by the time needed to compute the set $U = N^{h(m)}(v)$ of nodes at level i^- identified by Lemma 3.2. By Lemma 3.1, the distance from v to any of these nodes is at most

$$h(4)2^{(i^-+3)} - 2^{(i^-+1)} \;\leq\; 4(2^{l-2}+1)2^{(i^-+3)} \;\leq\; 16 \cdot 2^{(i^-+l)}.$$

Recall that $i^- = \max(0, i-l-1)$. If $i^- = 0$, the distance to any of the identified nodes is $O(2^l)$. If $i^- = i-l-1$, the distance is $O(2^i)$.

By definition of the hierarchy, the nodes of level i^- are separated from each other by a distance of at least 2^{i^-}. By a standard packing argument, this implies that the number of such nodes within any disk of radius r is at most $O((1+r/2^{i^-})^d)$, where d is the dimension. If $i^- = 0$, it follows that the number of nodes identified in Lemma 3.2 is $O((1+2^l/2^0)^d)$. On the other hand, if $i^- = i-l-1$, the number of nodes is $O((1+2^i/2^{i-l-1})^d)$. In either case, the number of nodes is clearly $O(2^{l \cdot d})$. Since $l = \log_2(1/\varepsilon) + O(1)$, it follows that the number of nodes is $O(1/\varepsilon^d)$.

The nodes of U are determined by computing the $h(m)$-fold neighbor set of v. We can do this by applying $h(m)$ levels of a breadth-first search to the graph of neighbors. The time to do this is proportional to the product of the number of

nodes visited and their degrees. Gao *et al.* [GGN06] show that each node has degree at most $(1 + 2c)^d - 1$. By our assumptions that $c = 8$ and d is fixed, this is $O(1)$. Thus, the total range-sketch query time is proportional to $|U|$, which is $O(1/\varepsilon^d)$. □

Theorem 3.1. *After preprocessing has completed, our algorithm takes time* $O(k(\log n \log\log \alpha)/\varepsilon^{2d})$ *per spanner, which is* $O((\log n \log\log \alpha)/\varepsilon^{2d})$ *under our assumption that* k *is a constant.*

Proof. We first show that, for a single level of a single spanner, the per-level subroutine takes time $O(k(\log n)/\varepsilon^{2d})$. To see this, observe that it performs k iterations. During each iteration, it updates the counts for $O(1/\varepsilon^d)$ greedy disks (those that overlap with disk E_j) based on the $O(1/\varepsilon^d)$ nodes in level i^- that are contained in each greedy disk. Each update also involves adjusting the position of an entry in a priority queue holding at most n points, which can be done in $O(\log n)$ time.

This per-level subroutine is invoked $O(\log\log \alpha)$ times per spanner, since there are $O(\log \alpha)$ levels in the discrete hierarchy, over which the binary search is performed to determine the best radius. Thus, the total time required to update any one spanner is $O(k(\log n \log\log \alpha)/\varepsilon^{2d})$. Since k is a constant, our algorithm takes time $O((\log n \log\log \alpha)/\varepsilon^{2d})$. □

We will now establish the approximation bound of $3 + \varepsilon$.

Theorem 3.2. *Let* r_{opt} *be the optimal radius for the discrete robust* k*-center solution for* S*, and let* r_{apx} *be the radius found by our algorithm. Then for any* $0 < \varepsilon \leq 1$*, we have* $r_{\text{apx}} \leq (3 + \varepsilon) r_{\text{opt}}$.

Proof. Let $v_1, \ldots, v_k$ denote the k optimal centers. We may express the optimal radius value, r_{opt}, as $2^i + x$ for some integer i and $0 \leq x < 2^i$. Let $p = \lceil sx/2^i \rceil$. If $p = s$, set $i \leftarrow i + 1$ and $p = 0$ (effectively rounding up to the next spanner copy). Clearly, $0 \leq p < s$, and $\frac{p-1}{s} 2^i < x \leq \frac{p}{s} 2^i$.

We first show that it is *possible*, given the information we maintain and the algorithm we use, for us to cover as many points as are covered by the optimal solution. For $1 \leq j \leq k$, let O_j denote the *optimal disk* of radius r_{opt} centered at v_j. Let u_j denote the ancestor of v_j in level i^-. (If $i^- = 0$, then $u_j = v_j$.) By the basic properties of the discrete hierarchy we have

$$\|u_j v_j\| \;\leq\; 2^{(i^- + 1)} \;\leq\; 2^{(i-l)}.$$

The node u_j will be considered by our algorithm (during the processing of spanner copy p and level i). Let G_j^- denote the disk of radius $g_i^-(\varepsilon, p)$ centered at u_j. We assert that every point lying within O_j will be included in the range-sketch count for u_j. To see this, let w be a point of S lying within O_j, that is, $\|v_j w\| \leq r_{\text{opt}}$. By the triangle inequality we have

$$\begin{aligned}
\|u_j w\| \;&\leq\; \|u_j v_j\| + \|v_j w\| \;\leq\; 2^{(i-l)} + r_{\text{opt}} \;<\; 2^{(i-l)} + \left(2^i + 2^i \frac{p}{s}\right) \\
&=\; 2^i \left(2^{-l} + \left(1 + \frac{p}{s}\right)\right) \;\leq\; 2^i \left(1 + \frac{p}{s}\right)(1 + 2^{-l}) \\
&=\; g_i^-(\varepsilon, p).
\end{aligned}$$

Therefore w lies within G_j^-, the inner radius for the range-sketch query, and so it must be included among the points counted in the range-sketch query for u_j. In summary, for each optimal disk O_j, there is a point u_j at level i^- of spanner copy p whose range sketch covers a superset of $S \cap O_j$.

To establish the approximation bound, let $u_1, \ldots, u_k$ denote the nodes chosen by our algorithm when run at level i of spanner copy p. (These are generally different from the u_j's described in the previous paragraph.) Let G_j and E_j denote the disks centered at u_j of radii $g_i(\varepsilon, p)$ and $e_i(\varepsilon, p)$, respectively. We will show that the expanded disks centered at these points *will* cover at least as many points as the optimal solution, which is as least $\lceil tn \rceil$. Since the algorithm returns the smallest expanded radius that (approximately) covers at least $\lceil tn \rceil$ points, this implies that $r_{\text{apx}} \leq e_i(\varepsilon, p)$.

Given a disk D, let $|D|$ denote the number of points of S contained within it. We will show that, for $0 \leq j \leq k$, $|O_1 \cup \ldots \cup O_j| \leq |E_1 \cup \ldots \cup E_j|$. Our proof is similar to that of Charikar *et al.* [CKMN01], and is based on an argument that charges each point covered in the optimal solution to a point covered by the solution produced by our algorithm. The proof proceeds by induction on j. The basis case is trivial, since for $j = 0$, both sets are empty. Assuming by induction that $|O_1 \cup \ldots \cup O_{j-1}| \leq |E_1 \cup \ldots \cup E_{j-1}|$, we consider what happens after the next center u_j is added to the approximate solution. We consider two cases:

- If G_j intersects any of the $k-(j-1)$ remaining optimal disks, define O_j be any such optimal disk. Any point w of O_j is within distance $g_i(\varepsilon, p)+2r_{\text{opt}}$ of u_j. It is easy to verify that $r_{\text{opt}} \leq g_i(\varepsilon, p)$, and therefore $\|u_j w\| \leq 3g_i(\varepsilon, p) = e_i(\varepsilon, p)$. Thus, E_j covers all the points of O_j. We charge each point in O_j to itself.

- If G_j does not intersect any of the remaining O_j, let O_j be the remaining optimal disk covering the greatest number of points of S. By our earlier

remarks, there exists a node in level i^- whose range-sketch count includes all the points of O_j. Since G_j was chosen greedily to maximize the number of unmarked points it covers, it follows that the number of unmarked points covered by G_j is at least $|O_j|$. We charge each point in O_j to an unmarked point in G_j.

Each point that is charged is charged either to itself or to some point in a greedy disk G_j that is disjoint from all remaining optimal disks, and therefore, each point is charged at most once. It follows that $|O_1 \cup \ldots \cup O_k| \leq |E_1 \cup \ldots \cup E_k|$.

To complete the analysis of the approximation bound, it suffices to show that $r_{\text{apx}}/r_{\text{opt}} \leq 3 + \varepsilon$. As observed earlier, we have

$$r_{\text{apx}} \;\leq\; e_i(\varepsilon, p) \qquad \text{and} \qquad r_{\text{opt}} \;=\; 2^i + x \;>\; 2^i \left(1 + \frac{p-1}{s}\right).$$

Thus, we have

$$\begin{aligned} \frac{r_{\text{apx}}}{r_{\text{opt}}} \;&<\; \frac{e_i(\varepsilon, p)}{2^i \left(1 + \frac{p-1}{s}\right)} \;=\; \frac{3 \cdot 2^i \left(1 + \frac{p}{s}\right)\left(1 + 3 \cdot 2^{-l}\right)}{2^i \left(1 + \frac{p-1}{s}\right)} \\ &=\; 3\left(1 + \frac{1}{s+p-1}\right)\left(1 + 3 \cdot 2^{-l}\right) \;\leq\; 3\left(1 + \frac{1}{s-1}\right)\left(1 + 3 \cdot 2^{-l}\right). \end{aligned}$$

Under our assumptions that $s = \lceil 10/\varepsilon \rceil$, $l = 4 - \lfloor \log_2 \varepsilon \rfloor$, and $\varepsilon \leq 1$, it follows easily that $s - 1 \geq 9/\varepsilon$, and $3 \cdot 2^{-l} \leq 3\varepsilon/16 \leq \varepsilon/5$. Therefore, we have

$$\frac{r_{\text{apx}}}{r_{\text{opt}}} \;<\; 3\left(1 + \frac{\varepsilon}{9}\right)\left(1 + \frac{\varepsilon}{5}\right) \;\leq\; 3\left(1 + \frac{\varepsilon}{3}\right) \;=\; 3 + \varepsilon,$$

which completes the proof. □

The assumption that $\varepsilon \leq 1$ in the statement of the theorem is a technicality. The analysis may be modified to work for any constant value of ε.

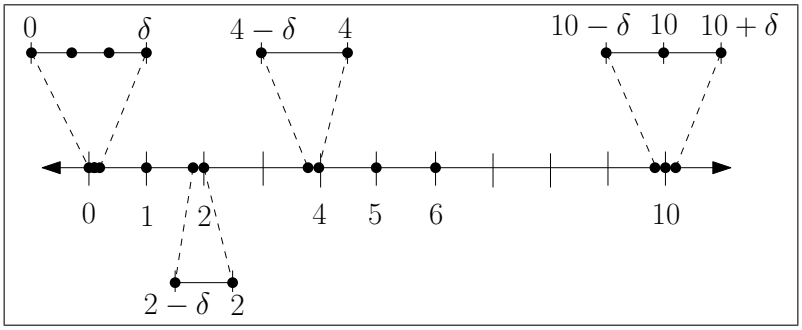

Figure 3.6: An example for which our algorithm gives an approximation ratio of $3 - \varepsilon$, where $k = 2$ and $\lceil tn \rceil = 11$.

3.3.3.1 Tightness of the Approximation Ratio

In this section, we present an example that demonstrates that our $(3 + \varepsilon)$-approximation ratio for the discrete k-center problem is nearly tight. In particular, given any sufficiently small $\varepsilon > 0$, we shall show that our algorithm achieves an approximation ratio of $3 - \varepsilon$ on this example. Let $\delta = \varepsilon/6$, and consider the set of 14 points illustrated in Figure 3.6. This point set consists of a collection of nine clusters placed on the real line, where each cluster contains from one to four points, each lying within distance δ of some integer point. Let $k = 2$ and $\lceil tn \rceil = 11$. It is easy to verify that the optimal radius is $r_{\text{opt}} = 1 + \delta$, which is achieved by placing centers at positions 1 and 5, so that the two disks centered at these points cover the $7 + 4 = 11$ points clustered about $\{0, 1, 2\}$ and $\{4, 5, 6\}$, respectively.

We will establish our bounds under the most favorable assumptions for the approximation algorithm. In particular, we assume that the approximation algorithm is free to select *any* point as a candidate center (not just the nodes in level

i^-). We assume that the base distance for the hierarchy of discrete centers has been chosen so that *any* desired radius arises as the value of the greedy radius, $g_i^-(\varepsilon, p)$, for some level i of some spanner copy p. Finally, we assume that the counts returned by range sketching algorithm are *exact*. Thus, relaxing any of these restrictions in our algorithm will not significantly affect the tightness of the approximation bound.

We assert that, even under these favorable assumptions, $r_{\text{apx}} \geq 3(1-\delta)$. To see this, suppose not. Letting p and i denote, respectively, the spanner copy and level that produce this value, we have $e_i(\varepsilon, p) = r_{\text{apx}} < 3(1-\delta)$. For all sufficiently small δ, this is less than $4-2\delta$. This implies that each expanded disk covers at most the single cluster containing its center and the clusters about the three consecutive integer points on either side of it. We also have

$$g_i^-(\varepsilon, p) \;<\; g_i(\varepsilon, p) \;=\; \frac{e_i(\varepsilon, p)}{3} \;<\; 1-\delta.$$

Since two points from different clusters are separated by a distance of at least $1-\delta$, each greedy disk covers only a single cluster. Since the cluster near 0 has the most points (four), some point of this cluster will be chosen first. The expanded disk centered here covers only the seven points in the clusters near 0, 1, and 2. The next center to be chosen is the cluster near 10 having three points. The expanded disk is not large enough to include any other points. Thus, the algorithm succeeds in covering only $7+3=10$ points, and therefore it fails.

Since $r_{\text{apx}} \geq 3(1-\delta)$, we see the that approximation ratio is

$$\frac{r_{\text{apx}}}{r_{\text{opt}}} \;\geq\; \frac{3(1-\delta)}{1+\delta} \;=\; 3 - \frac{6\delta}{1+\delta} \;\geq\; 3-6\delta \;\geq\; 3-\varepsilon,$$

as desired.

3.3.4 The Absolute Problem

Recall that in the absolute formulation the centers may be any point in space. In this section we present a $(4+\varepsilon)$-approximation algorithm for the absolute, robust k-center problem. The algorithm is the same as for the discrete problem, except that we modify the values of the radii upon which the algorithm is based. In particular, in place of $g_i(\varepsilon, p)$ and $e_i(\varepsilon, p)$, we define:

$$\hat{g}_i(\varepsilon, p) \;=\; 2g_i \qquad \text{and} \qquad \hat{e}_i(\varepsilon, p) \;=\; 2\hat{g}_i(\varepsilon, p).$$

We also define two slightly smaller radii for the range-sketch queries:

$$\hat{g}_i^-(\varepsilon, p) \;=\; 2g_i^-(\varepsilon, p) \qquad \text{and} \qquad \hat{e}_i^-(\varepsilon, p) \;=\; 2\hat{g}_i^-(\varepsilon, p).$$

Since the values of these radii have increased, we also increase the values used in various parts of Lemma 3.2 to $h(2)$, $h(4)$, and $h(6)$, respectively. By a straightforward modification of the analysis of the approximation bound given in the proof of Theorem 4.2 for the discrete case, we have the following.

Theorem 3.3. *Let r_{opt} be the optimal radius for the absolute robust k-center solution for S, and let r_{apx} be the radius found by this absolute algorithm. Then for any $0 < \varepsilon \leq 1$, we have $r_{\text{apx}} \leq (4+\varepsilon) r_{\text{opt}}$.*

Proof. The proof of Theorem 4.2 makes use of two key facts about the greedy and expanded disks. The first is that there exists a spanner copy p and a level i such that, for any optimal disk, there exists a node u in level i^- such that the associated greedy disk G_u^- of radius g_i^- contains this optimal disk. In the absolute case, the center of an optimal disk O may be at an arbitrary point of space, but by choosing

any point of S that is covered by O and centering a disk O' of radius $2r_{\text{opt}}$ at this point, we see that O is contained within O'. Therefore, in our modified algorithm, there exists a node u in level i^- such that the greedy disk $\widehat{G}_u^-$ of radius $\hat{g}_i^- = 2g_i^-$ contains O', and hence contains O as well.

The second key fact used in our analysis of the discrete algorithm is that if any optimal disk overlaps a greedy disk G_u, then the corresponding expanded disk E_u contains the optimal disk. In the absolute case, if any optimal disk O overlaps a greedy disk $\widehat{G}_u$, then every point of O lies within distance $\hat{g}_i + 2r_{\text{opt}} \leq 2g_i + 2g_i = 4g_i = \hat{e}_i$ of u. Therefore, $O \subseteq \widehat{E}_u$. Given these two key facts, the remainder of the proof is the same as that of Theorem 4.2, but with an appropriate adjustment of the specific values of $s(\varepsilon)$ and $l(\varepsilon)$ and with the fact that the expanded radius has increased by a factor of $\hat{e}_i/e_i = 4/3$. □

3.4 Kinetic Spanner Maintenance and Quality

3.4.1 Certificates

The KDS algorithm of Gao, Guibas, and Nguyen [GGN06] for the deformable spanner maintains four types of certificates. These are applied to all levels i of each spanner. A *parent-child certificate* guarantees that a node in level i is within distance 2^{i+1} of its parent. An *edge certificate* guarantees that a pair of neighboring nodes in level i lie within distance $c \cdot 2^i$ of each other (where we choose $c = 8$). A *separation certificate* guarantees that any two distinct neighboring nodes nodes in level i are separated by a distance of at least 2^i. A *potential neighbor certificate* guarantees

that two non-neighboring level-i nodes whose parents are neighbors are separated by a distance of more than $c \cdot 2^i$. (Potential neighbor certificates are maintained so that edge certificates can be easily maintained when points move closer to each other [GGN06].)

Recall that in our data structure we have multiple spanners with differing base distances b_p, for $0 \leq p < s(\varepsilon)$. As before, we present the algorithm only for the simplest case of $p = 0$ and hence $b_p = 1$, but the other cases are identical up to a scaling of distances. Also recall that, for each level i, a set of k centers is maintained as well as a priority queue. The entries in this priority queue are nodes v in S_{i^-}, and the priority values are the counts of points within G_v (computed by range-sketching).

Our update rules are identical to those given in [GGN06], with the following additions to the update rules for parent-child, edge, and separation certificates. Each rule is stated relative to some level i.

Addition of a spanner edge. Increment the counts for G_v and E_v for all neighbors indicated by Lemma 3.2 parts (i) and (ii) that are affected by this edge creation, and update the priority queue for level i appropriately.

Deletion of a spanner edge. Decrement the counts for G_v and E_v for all neighbors indicated by Lemma 3.2 parts (i) and (ii) that are affected by this edge deletion, and update the priority queue for level i appropriately.

Addition of parent-child certificate. Increment the descendant counts for all new ancestor nodes and for all counts for G_v and E_v for neighbors indicated

by Lemma 3.2 parts (i) and (ii), and update the priority queue for level i appropriately.

Failure of parent-child certificate. Decrement the counts for all previous ancestor nodes and for all counts for G_v and E_v for neighbors indicated by Lemma 3.2 parts (i) and (ii), and update the priority queue for level i appropriately.

Whenever any count G_v changes for some node v, it may affect the k-center solution. In particular, if v's count increases and v is not a center, the k-center solution for that spanner is recalculated. Otherwise, the counts and priority queue are updated, but the solution need not be recalculated. Similarly, if v's count decreases and v is a center, the k-center solution for that spanner is recalculated, otherwise only the counts and priority queue are updated. In the cases when recalculating the solution is necessary, our static algorithm is applied. Note that as time progresses our KDS allows outlying points to become inliers and vice versa.

3.4.2 Preconditions

Maintaining the following preconditions ensures that the static algorithm will be applicable at any time during the motion of the points. The preconditions needed for all points are maintained by the certificates and update conditions given in the original deformable spanner [GGN06].

For each level i in the discrete hierarchy, the following level-specific preconditions are maintained. First, as in Section 3.3.2, we maintain a count for each node

in the discrete hierarchy of the number of points that are descendants of that node. These counts are updated whenever parent-child certificates either are created or fail. Also, for each node u at level i^-, we maintain the counts of points lying approximately within distances g_i and e_i, which are computed through the range-sketch subroutine. For each level i, we maintain the counts of the points within the fuzzy disks G_v associated with the nodes v on level i. These counts are stored in a priority queue. Whenever such a count changes (in the procedure update-greedy-disks), the priority queue is updated accordingly. If this update occurs during the course of the algorithm because a center was chosen and nodes were marked as covered, these changes are kept track of until all centers are chosen. The changes are undone in reverse order, so that the counts accurately represent the current state as the points continue to move. The preconditions for each level i are maintained according to the update rules given in Section 3.4.1.

3.4.3 Quality

In order to assure the quality of the spanner, we must reason about compactness, locality, efficiency, and responsiveness. Recall that n is the total number of points, d the dimension, and α the user-supplied upper bound on the aspect ratio. In this section we will show that compactness, locality, and efficiency are bounded by $O(n/\varepsilon^{d+1})$, $O((\log \alpha)/\varepsilon^d)$, and $O(n^2(\log \alpha)/\varepsilon)$ respectively, which match the bounds given by Gao *et al.* [GGN06] up to a factor of $s = O(1/\varepsilon)$. In addition, we will establish a bound of $O((\log n \log \alpha)/\varepsilon^{2d})$ on responsiveness.

3.4.3.1 Compactness and Locality

Compactness and locality conditions ensure that maintaining certificates for the kinetic data structure is not too costly by bounding the number of certificates. Recall that compactness bounds the total number of certificates, and locality bounds the number of certificates in which each point can participate. Since our data structure consists of $s = O(1/\varepsilon)$ copies of the spanner of [GGN06], it follows that the compactness is larger than theirs by a factor of $O(1/\varepsilon)$ and our locality is the same. Thus we have the following.

Theorem 3.4. *Our KDS satisfies compactness and locality with $O(n/\varepsilon^{d+1})$ total certificates and $O((\log \alpha)/\varepsilon^d)$ certificates per point.*

3.4.3.2 Efficiency

The efficiency condition ensures that maintaining the kinetic data structure is not too expensive by bounding the number of certificate failures that can occur. This is compared to the number of required changes to the combinatorial structure of the spanner to determine the efficiency of the KDS.

Theorem 3.5. *Our KDS satisfies efficiency (with respect to the number of spanner updates) with $O(n^2(\log \alpha)/\varepsilon)$ possible certificate maintenance events.*

Proof. The deformable spanner of [GGN06] has $O(n^2 \log \alpha)$ possible maintenance events because, under pseudo-algebraic motion, events only occur when the distance between two points is at the boundary of some certificate on a given level i, namely

2^i or $c \cdot 2^i$. Since there are $O(\log \alpha)$ possible levels and $2n^2$ of these inter-point distances, there are $O(n^2 \log \alpha)$ possible maintenance events [GGN06]. Recall that there are s spanners that differ according to their base distances, but the bound on the number of maintenance events per spanner remains the same, so the total possible number of maintenance events increases by a factor of $s = O(1/\varepsilon)$. Since the spanner has not changed except for the base distance, the number of changes required by the combinatorial structure of each spanner remains $\Omega(n^2)$ [GGN06], so any approach based on a spanner requires $\Omega(n^2)$ changes. So the kinetic data structure is efficient. □

3.4.3.3 Responsiveness

The responsiveness condition ensures that maintaining the kinetic data structure is not too expensive by bounding the amount of time taken to repair each failed certificate. Our spanner satisfies responsiveness with $O((\log n \log \alpha)/\varepsilon^{2d})$ time per certificate update. This time is due to the possibility that a failure or addition of a certificate could require the algorithm to be re-run and the possibility that certificates may need to be updated on each level of a single spanner.

Theorem 3.6. *Our KDS satisfies responsiveness with $O((\log n \log \alpha)/\varepsilon^{2d})$ time per certificate update.*

Proof. As shown in [GGN06], the certificates of each copy of the hierarchical spanner can be updated in $O(1)$ or $O((\log \alpha)/\varepsilon^d)$ time depending on the specific certificate that fails. In our case, the failure or addition of a certificate could require our

static algorithm to be re-run. When the priority queue is updated during the re-running, a list of changes is maintained. After k centers for a level are chosen, the priority queue is returned to its original state (the state assuming all points are uncovered). Since the changes made are undone, this increases the running time by only a constant factor. By Theorem 3.1, the solution can be recalculated in time $O(k(\log n \log\log \alpha)/\varepsilon^{2d})$. Given our assumption that k is a constant, this is $O((\log n \log\log \alpha)/\varepsilon^{2d})$.

The failure or addition of a certificate could also require points to be updated for $O(1/\varepsilon^d)$ nodes (see Lemma 3.3) on all $O(\log \alpha)$ levels. Each such update may induce a change to a priority queue entry, adding an additional factor of $O(\log n)$. Thus, the total time to repair a failed certificate is $O((\log n \log \alpha)/\varepsilon^d)$. The overall responsiveness is the maximum of these two bounds (recalculation and certificate repair), which is bounded by $O((\log n \log \alpha)/\varepsilon^{2d})$, as desired. □

3.5 Non-Robust Kinetic K-Center Algorithm

In this section we mention that by using our hierarchical spanner, it is possible to improve on the non-robust discrete k-center algorithm presented by Gao *et al.* [GGN06]. That algorithm achieved a factor-8 approximation, and we will improve this to a $(4+\varepsilon)$-approximation (both for arbitrary k). Recall that, rather than using a single spanner, we generate $s(\varepsilon) = O(1/\varepsilon)$ spanners with differing base distances. Our approach is to run the algorithm presented by Gao *et al.* on all spanner copies and return the smallest radius over all these runs. The algorithm is

illustrated in Figure 3.7. The value of s is restricted in the proof of Theorem 3.7. For proof of maintenance under motion, we refer the reader to the proof of a similar algorithm given by Gao *et al.* [GGN06].

Theorem 3.7. *Let r_{opt} be the optimal radius for k-centers chosen from the input points and r_{apx} be the radius found by our non-robust kinetic k-center algorithm, then $r_{\text{apx}} \leq (4+\varepsilon)r_{\text{opt}}$.*

Proof. The algorithm chooses some S_i on spanner p with associated radius $2^{i+1}(1+\frac{p}{s})$, where S_i has the minimum radius such that $|S_i| \geq k$. Since $2^{i+1}(1+\frac{p-1}{s}) < 2^{i+1}(1+\frac{p}{s})$, $|S_i| > k$ on spanner $p-1$. So at least two points from S_i are assigned to the same center in the optimal solution. These points are separated by a distance of at least $2^i(1+\frac{p-1}{s})$, so the optimal radius must be at least $2^{i-1}(1+\frac{p-1}{s})$. To determine the approximation ratio we consider the ratio between $r_{\text{opt}} \geq 2^{i-1}(1+\frac{p-1}{s})$ and $r_{\text{apx}} \leq 2^{i+1}(1+\frac{p}{s})$. Choosing $s(\varepsilon) \geq \frac{2}{\varepsilon}+\frac{1}{2}$ results in a $(4+\varepsilon)$-approximation algorithm. □

Figure 3.7: non-robust$(S, k, \varepsilon, \alpha)$

for $p = 0$ to $s - 1$

  for $i = \lceil \log \alpha \rceil$ down-to 0

    if $|S_i(p)| > k$

      $r_p \leftarrow 2^{i+2}(1 + \frac{p}{s})$

      $K_p \leftarrow S_{i+1}$

      while $|K_p| < k$ add an arbitrary node of $S_i(p)$ to K_p

      break out of inner loop

$r \leftarrow \min_p r_p$

output (K_p, r), each point is serviced by its ancestor in K_p

Figure 3.7: The non-robust discrete kinetic k-center algorithm for arbitrary k. Recall that $S_i(p)$ denotes the ith level of the pth spanner.

Chapter 4

A Sensor-Based Framework For Kinetic Data Compression

In this chapter, we introduce a framework for storing and processing kinetic data observed by sensor networks. These sensor networks generate vast quantities of data, which motivates a significant need for data compression. We are given a set of sensors, each of which continuously monitors some region of space. We are interested in the kinetic data generated by a finite set of objects moving through space, as observed by these sensors. Our model relies purely on sensor observations, and, unlike KDS, points are allowed to move freely and the model requires no advance notification of motion plans. Sensor outputs are represented as random processes, where nearby sensors may be statistically dependent. We model the local nature of sensor networks by assuming that two sensor outputs are statistically dependent only if the two sensors are among the k nearest neighbors of each other. We present an algorithm for the lossless compression of the data produced by the network. We show that, under the statistical dependence and locality assumptions of our framework, asymptotically this compression algorithm encodes the data to within a constant factor of the information-theoretic lower bound optimum dictated by the joint entropy of the system. In order to justify our locality assumptions,

we provide a theoretical comparison with a variant of the kinetic data structures framework. We prove that the storage size required by an optimal system operating under our locality assumptions is on the order of the size required by our variant.

4.1 Introduction

In this chapter we consider the problem of how to compress the massive quantities of data that are streamed from large sensor networks. Compression methods can be broadly categorized as being either *lossless* (the original data is fully recoverable), or *lossy* (information may be lost through approximation). Because lossy compression provides much higher compression rates, it is by far the more commonly studied approach in sensor networks. Our ultimate interest is in scientific applications involving the monitoring of the motion of objects in space, where the loss of any data may be harmful to the subsequent analysis. For this reason, we focus on the less studied problem of lossless compression of sensor network data. Virtually all lossless compression techniques that operate on a single stream (such as Huffman coding [Huf52], arithmetic coding [Ris76], Lempel-Ziv [ZL77]) rely on the statistical redundancy present in the data stream in order to achieve high compression rates. In the context of sensor networks, this redundancy arises naturally due to correlations in the outputs of sensors that are spatially close to each other. As with existing methods for lossy compression [DKR06, GNSL09], our approach is based on aggregating correlated streams and compressing these aggregated streams. (More information on data compression can be found in Chapter 2.4.)

A significant amount of research to date has focused on the efficient collection and processing of sensor network data within the network itself, for example, through the minimization of power consumption or communication costs [CMZ07, CMY08, SWP08]. (More information on sensor networks can be found in Chapter 2.3.) We focus on doing lossless compression on the data locally and then downloading it to traditional computer systems for analysis. Clustering the stationary sensors is a strategy that has been previously used to improve the scalability as well as the energy and communication efficiency of the sensor network [JN06]. Compressing the data before transmitting additionally improves the communication efficiency of the network.

We are particularly interested in *kinetic data*, by which we mean data arising from the observation of a discrete set of objects moving in time (as opposed to continuous phenomena such as temperature). We explore how best to store and process these assembled data sets for the purposes of later efficient retrieval, visualization, and statistical analysis of the information contained within them. The data sets generated by sensor networks have a number of spatial, temporal, and statistical properties that render them interesting for study. We assume that we do not get to choose the sensor deployment based on object motion (as done in [NS08]), but instead use sensors at given locations to observe the motion of a discrete set of objects over some domain of interest. Thus, it is to be expected that the entities observed by one sensor will also likely be observed by nearby sensors, albeit at a slightly different time. For example, many of the vehicles driving by one traffic camera are likely to be observed by nearby cameras, perhaps a short time later or earlier. If we assume that the data can be modeled by a random process, it is reasonable to expect that a high degree of

statistical dependence exists between the data streams generated by nearby sensors. If so, the information content of the assembled data will be significantly smaller than the size of the raw data. In other words, the raw sensor streams, when considered in aggregate, will contain a great deal of redundancy. Well-designed storage and processing systems should capitalize on this redundancy to optimize space and processing times. In this chapter we propose a statistical model of kinetic data as observed by a collection of fixed sensors. We will present a method for the lossless compression of the resulting data sets and will show that this method is within a constant factor of the asymptotically optimal bit rate, subject to the assumptions of our model.

Although we address the problem of compression here, we are more generally interested in the storage and processing of large data sets arising from sensor networks [DKR06, DKR07, SM06, Gui02, GTH08]. This will involve the retrieval and statistical analysis of the information contained within them. In Chapter 5 we will discuss this compression scheme under realistic assumptions (those presented here are based on pure theoretical analyses) and in Chapter 6 we consider retrieval via spatio-temporal range searching [FM10b]. Thus, we will discuss compression within the broader context of a framework for processing large kinetic data sets arising from a collection of fixed sensors. We feel that this framework provides a useful context within which to design and analyze efficient data structures and algorithms for kinetic sensor data.

The problem of processing kinetic data has been well studied in the field of computational geometry in a standard computational setting [GJS96, Ata85, ST95, ST96, BG99, Kah91]. A survey of practical and theoretical aspects of modeling mo-

tion can be found in [AGE+02]. Many of these apply in an online context and rely on *a priori* information about point motion. The most successful of these frameworks is the *kinetic data structures* (KDS) model proposed by Basch, Guibas, and Hershberger [BG99]. The basic entities in this framework are points in motion, where the motion is expressed as piecewise algebraic flight plans. Geometric structures are maintained through a set of boolean conditions, called certificates, and a set of associated update rules. The efficiency of algorithms in this model is a function of the number of certificates involved and the efficiency of processing them. In a sensor context, moving data has been considered in relation to sensor placement based on possible object trajectories modeled by a set of 3D curves over space and time [NS08]. (More information about motion and kinetic data structures can be found in Chapter 2.)

As valuable as KDS has been for developing theoretical analyses of point motion (see [Gui04] for a survey), it is unsuitable for many real-world contexts and for theoretical problems that do not have locally determined properties. The requirements of algebraic point motion and advance knowledge of flight plans are either inapplicable or infeasible in many scientific applications. Agarwal *et al.* [AGE+02] identify fundamental directions that future research should pursue. Our work addresses four of these issues; unpredicted motion, motion-sensitivity, robustness, and theoretical discrete models of motion. In our framework we will process a point set without predicted knowledge and no matter its motion. *Motion-sensitive algorithms* admit complexity analyses based on the underlying motion. Imagine a set of points following a straight line or moving continuously in a circle; any algorithm calculat-

ing statistical information about such a point set should be more efficient than the same algorithm operating on a set of randomly moving points. Our motion-sensitive framework will pay a cost in efficiency based on the information content of the point motion. *Robustness* is a quality of statistical estimators that allow outliers. Unlike KDS, we will ignore point identities in favor of statistical properties; KDS focuses on properties in relation to individual points. In the KDS model, a rearrangement of points which maintained a global statistical property could trigger many certificate failures despite the maintenance of the statistical property being calculated. For example, two points which exactly switch position do not change the diameter of the point set, but may cause multiple certificate failures. Through anonymization of the points and discrete time sampling, our framework reduces the overhead in these instances. Finally, Agarwal *et al.* note that most theoretical work relies on continuous frameworks while applied work experimentally evaluates methods based on discrete models. Our framework uses a discrete sampling model, but is still theoretically sound. In addition, the underlying goal which structures KDS operations is maintenance of information into the future; we will process sensed data after the fact, e.g., for efficient retrieval. For these problem types, our framework serves as an alternative to the KDS model.

There has also been study of algorithms that involve the distributed online processing of sensor-network data. One example is the *continuous distributed model* described by Cormode *et al.* [CMZ07]. This model contains a set of sensors, which each observe a stream of data describing the recent changes in local observations. Each sensor may communicate with any other sensor or with a designated central

coordinator. Efficiency is typically expressed as a trade-off between communication complexity and accuracy. This framework has been successfully applied to the maintenance of a number of statistics online [CMZ07, CMY08, BO03]. Another example is the competitive online tracking algorithm of Yi and Zhang [YZ09], in which a tracker-observer pair coordinate to monitor the motion of a moving point. Again, complexity is measured by the amount of communication between the tracker and the observer. The idea of the tracker and observer is reminiscent of an earlier model for incremental motion by Mount *et al.* [MNP+04]. Unlike these models, our framework applies in a traditional (non-distributed) computational setting.

Here is a high-level overview of our framework, which will be described in greater detail in Section 4.2. We assume we are given a fixed set of sensors, which are modeled as points in some metric space. (An approach based on metric spaces, in contrast to standard Euclidean space, offers greater flexibility in how distances are defined between objects. This is useful in wireless settings, where transmission distance may be a function of non-Euclidean considerations, such as topography and the presence of buildings and other structures.) Each sensor is associated with a region of space, which it monitors. The moving entities are modeled as points that move over time. At regular time intervals, each sensor computes statistical information about the points within its region, which are streamed as output. For the purposes of this presentation, we assume that this information is simply an *occupancy count* of the number of points that lie within the sensor's region at the given time instant. In other words, we follow the minimal assumptions made by Gandhi *et al.* [GKS08] and do not rely on a sensor's ability to accurately record distance, angle, etc.

As mentioned above, our objective is to compress this data in a lossless manner by exploiting redundancy in the sensor streams. In order to establish formal bounds on the quality of this compression, we assume (as is common in entropy encoding) that the output of each sensor can be modeled as a stationary, ergodic random process. We allow for statistical dependencies between the sensor streams. Shannon's source coding theorem implies that, in the limit, the minimum number of bits needed to encode the data is bounded from below by the normalized joint entropy of the resulting system of random processes. There are known lossless compression algorithms, such as Lempel-Ziv [ZL77], that achieve this lower bound asymptotically. It would be utterly infeasible, however, to apply this observation *en masse* to the entire joint system of all the sensor streams. Instead, we would like to partition the streams into small subsets, and compress each subset independently. The problem in our context is how to bound the loss of efficiency due to the partitioning process. In order to overcome this problem we need to impose limits on the degree of statistical dependence among the sensors. Our approach is based on a locality assumption. Given a parameter k, we say that a sensor system is *k-local* if each sensor's output is statistically dependent on only its k-nearest sensors.

The full contributions of this chapter are described in the following sections. In Section 4.2, we introduce a new framework for the compression and analysis of kinetic sensor data. In Section 4.3, we prove that any k-local system that resides in a space of fixed dimension can be nicely partitioned in the manner described above, so that joint compressions involve groups of at most $k + 1$ sensors. We show that the final compression is within a factor c of the information-theoretic lower bound,

where c is independent of k, and depends only on the dimension of the space. In Section 4.4, we justify our k-local model theoretically as compared to a variant of the KDS model. We prove that the compressed data from our model takes space on the order of the space used by the KDS variant.

4.2 Data Framework

In this section we present a formal model of the essential features of the sensor networks to which our results will apply. Our main goal is that it realistically model the data sets arising in typical wireless sensor-networks when observing kinetic data while also allowing for a clean theoretical analysis. We assume a fixed set of S sensors operating over a total time period of length T. The sensors are modeled as points in some metric space. We may think of the space as $\mathbb{R}^d$ for some fixed d, but our results apply in any metric space of bounded doubling dimension [KL04]. We model the objects of our system as points moving continuously in this space, and we make no assumptions *a priori* about the nature of this motion. Each sensor observes some *region* surrounding it. In general, our framework makes no assumptions about the size, shape, or density of these regions, but additional assumptions may be imposed later in special cases. The sensor regions need not be disjoint, nor do they need to cover all the moving points at any given time.

Each sensor continually collects statistical information about the points lying within its region, and it outputs this information at synchronized time steps. As mentioned above, we assume throughout that this information is simply an *occu-*

pancy count of the number of points that lie within the region. (The assumption of synchronization is mostly for the sake of convenience of notation. As we shall see, our compression algorithm operates jointly on local groups of a fixed size, and hence it is required only that the sensors of each group behave synchronously.)

As mentioned in the introduction, our framework is based on an information-theoretic approach. Let us begin with a few basic definitions (see, e.g., [CT06]). We assume that the sensor outputs can be modeled by a stationary, ergodic random process. Since the streams are synchronized and the cardinality of the moving point set is finite, we can think of the S sensor streams as a collection of S strings, each of length T, over a finite alphabet. Letting lg denote the logarithm base-2, the *entropy* of a discrete random variable X, denoted $H(X)$, is defined to be $-\sum_x p_x \lg p_x$, where the sum is over the possible values x of X, and p_x is the probability of x.

Recall from Chapter 2.4 that we can generalize entropy to random processes as follows. Given a stationary, ergodic random process X, consider the limit of the entropy of arbitrarily long sequences of X, normalized by the sequence length. This leads to the notion of *normalized entropy*, which is defined to be

$$H(X) \;=\; \lim_{T\to\infty} -\frac{1}{T} \sum_{x,|x|=T} p_x \lg p_x,$$

where the sum is over sequences x of length T, and p_x denotes the probability of this sequence. Normalized entropy considers not only the distribution of individual characters, but the tendencies for certain patterns of characters to repeat over time.

We can also generalize the concept of entropy to collections of random variables. Given a sequence $\boldsymbol{X} = \langle X_1, X_2, \ldots, X_S \rangle$ of (possibly statistically correlated)

random variables, the *joint entropy* is defined to be $H(\boldsymbol{X}) = -\sum_{\boldsymbol{x}} p_{\boldsymbol{x}} \lg p_{\boldsymbol{x}}$, where the sum is taken over all S-tuples $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_S \rangle$ of possible values, and $p_{\boldsymbol{x}}$ is the probability of this joint outcome [CT06]. The generalization to *normalized joint entropy* is straightforward. Normalized joint entropy further strengthens normalized entropy by considering correlations and statistical dependencies between the various streams.

In this chapter we are interested in the lossless compression of the joint sensor stream. Shannon's source coding theorem states that in the limit, as the length of a stream of independent, identically distributed (i.i.d.) random variables goes to infinity, the minimum number of required bits to allow lossless compression of each character of the stream is equal to the entropy of the stream [Sha48]. In our case, Shannon's theorem implies that the optimum bit rate of a lossless encoding of the joint sensor system cannot be less than the normalized joint entropy of the system. Thus, the normalized joint entropy is the gold standard for the asymptotic efficiency of any compression method. Henceforth, all references to "joint entropy" and "entropy" should be understood to mean the normalized versions of each.

As mentioned above, joint compression of all the sensor streams is not feasible. Our approach will be to assume a limit on statistical dependencies among the observed sensor outputs based on geometric locality. It is reasonable to expect that the outputs of nearby sensors will exhibit a higher degree of statistical dependence with each other than more distant ones. Although statistical dependence would be expected to decrease gradually with increasing distance, in order to keep our model as simple and clean as possible, we will assume that beyond some threshold, the

statistical dependence between sensors is so small that it may be treated as zero. (We consider the more realistic version of this assumption in the following chapter.)

There are a number of natural ways to define such a threshold distance. One is an *absolute approach*, which is given a threshold distance parameter r, and in which it is assumed that any two sensors that lie at distance greater than r from each other have statistically independent output streams. The second is a *relative approach* in which an integer k is provided, and it is assumed that two sensor output streams are statistically dependent only if each is among the k nearest sensors of the other. In this chapter we will take the latter approach. One reason is that it adapts to the local density of sensors. Another reason arises by observing that, in the absolute model, all the sensors might lie within distance r of each other. This means that all the sensors could be mutually statistically dependent, which would render optimal compression intractable. On the other hand, if we deal with this by imposing the density restriction that no sensor has more than some number, say k, sensors within distance r, then the absolute approach reduces to a special case of the relative approach.

This locality restriction allows reasoning about sensor outputs in subsets. Previous restrictions of this form include the Lovász Local Lemma [EL75] which also assumes dependence on at most k events. Particle simulations (often used to simulate physical objects for animation) based on smoothed particle hydrodynamics have also used similar locality restrictions to determine which neighboring particles impact each other. These calculations are made over densely sampled particles and are based on a kernel function which determines the impact of one particle on another. This frequently amounts to a cut-off distance after which we assume that

the particles are too far away to impact each other [APKG07]. For a survey on smoothed particle hydrodynamics see [Mon05].

Formally, let $P = \{p_1, p_2, \ldots, p_S\}$ denote the sensor positions. Given some integer parameter k, we assume that each sensor's output can be statistically dependent on only its k nearest sensors. Since statistical dependence is a symmetric relation, two sensors can exhibit dependence only if each is among the k nearest neighbors of the other. More precisely, let $NN_k(i)$ denote the set of k closest sensors to p_i (not including sensor i itself). We say that two sensors i and j are *mutually k-close* if $p_i \in NN_k(j)$ and $p_j \in NN_k(i)$. A system of sensors is said to be *k-local* if for any two sensors that are not mutually k-close, their observations are statistically independent. (Thus, 0-locality means that the sensor observations are mutually independent.) Let $\boldsymbol{X} = \langle X_1, X_2, \ldots, X_S \rangle$ be a system of random streams associated with by S sensors, and let $H(\boldsymbol{X})$ denote its joint entropy. Given two random processes X and Y, define the *conditional entropy* of X given Y to be

$$H(X \mid Y) \;=\; -\sum_{x \in X, y \in Y} p(x,y) \log p(y \mid x).$$

Note that $H(X \mid Y) \leq H(X)$, and if X and Y are statistically independent, then $H(X \mid Y) = H(X)$. By the chain rule for conditional entropy [CT06], we have

$$H(\boldsymbol{X}) \;=\; H(X_1) + H(X_2 \mid X_1) + \ldots + H(X_i \mid X_1, \ldots, X_{i-1}) + \ldots + H(X_S \mid X_1, \ldots, X_{S-1}).$$

Letting

$$D_i(k) \;=\; \{X_j : 1 \leq j < i \text{ and sensors } i \text{ and } j \text{ are mutually } k\text{-close}\}$$

we define the *k-local entropy*, denoted $H_k(\boldsymbol{X})$, to be $\sum_{i=1}^{S} H(X_i \mid D_i(k))$. Note

that $H(\boldsymbol{X}) \leq H_k(\boldsymbol{X})$ and equality holds when $k = S$. By definition of k-locality, $H(X_i \mid X_1, X_2, \ldots, X_{i-1}) = H(X_i \mid D_k(i))$. By applying the chain rule for joint entropy, we have the following easy consequence, which states that, under our locality assumption, k-local entropy is the same as the joint entropy of the entire system.

Lemma 4.1. *Given a k-local sensor system with set of observations $\boldsymbol{X}$, $H(\boldsymbol{X}) = H_k(\boldsymbol{X})$.*

The assumption of statistical independence is rather strong, since two distant sensor streams may be dependent simply because they exhibit a dependence with a common external event, such as the weather or time of day. Presumably, such dependencies would be shared by all sensors, and certainly by the k nearest neighbors. The important aspect of independence is encapsulated in the above lemma, since it indicates that, from the perspective of joint entropy, the k nearest neighbors explain essentially all the dependence with the rest of the system. Although we assume perfect statistical independence beyond the range of the kth nearest neighbor, the practical impact of this assumption is that any dependencies that may exist beyond this range have a negligible impact on the joint entropy of the system, and hence a negligible impact on the degree of compressibility in the system.

One advantage of our characterization of mutually dependent sensor outputs is that it naturally adapts to the distribution of sensors. It is not dependent on messy metric quantities, such as the absolute distances between sensors or the degree of overlap between sensed regions. Note, however, that our model can be applied in contexts where absolute distances are meaningful. For example, consider a setting

in which each sensor monitors a region of radius r. Given two positive parameters α and β, we assume that the number of sensors whose centers lie within any ball of radius r is at most α, and the outputs of any two sensors can be statistically dependent only if they are within distance βr of each other. Then, by a simple packing argument, it follows that such a system is k-local for $k = O(\alpha\,\beta^{O(1)})$, in any space of constant doubling dimension.

4.3 Compression Results

Before presenting the main result of this section, we present a lemma which is combinatorially interesting in its own right. This partitioning lemma combined with a compression algorithm allows us to compress the motion of points as recorded by sensors to an encoding size which is c times the optimal, where c is an integral constant to be specified in the proof of Lemma 4.2.

4.3.1 Partitioning Lemma

First, we present some definitions about properties of the static point set representing sensor locations. Let $r_k(p)$ be the distance from some sensor at location p to its k^{th} nearest neighbor. Recall that points are mutually k-close if they are in each other's k nearest neighbors. We say that a point set $P \in \mathbb{R}^d$ is *k-clusterable* if it can be partitioned into subsets $C_{i1}, C_{i2}, \ldots$ such that $|C_{ij}| \leq k+1$ and if p and q are mutually k-close then p and q are in the same subset of the partition. Intuitively, this means that naturally defined clusters in the set are separated enough so

that points within the same cluster are closer to each other than they are to points outside of the cluster. The following lemma holds for all metrics with constant *doubling dimension*, where these metrics are defined to limit to a constant the number of balls that cover a ball with twice their radius [KL04]. Euclidean spaces are of constant doubling dimension.

Lemma 4.2. *In any doubling space there exists an integral constant c such that for all integral $k > 0$ given any set P in the doubling space, P can be partitioned into $P_1, P_2, \ldots, P_c$ such that for $1 \leq i \leq c$, P_i is* k-clusterable.

The partitioning algorithm which implements Lemma 4.2 is shown in Figure 4.1. It proceeds by iteratively finding the unmarked point p with minimum $r = r_k(p)$, moving all points within r, henceforth called a *cluster*, to the current partition, and marking all points within $3r$ of p. A new partition is created whenever all remaining points have been marked. The marked points are used to create a buffer zone which separates clusters so that all points are closer to points within their cluster than they are to any other points in the partition. The algorithm's inner loop creates these clusters, and the outer loop creates the c partitions.

Proof. Each partition is k-clusterable since (by the marking process) for any cluster with diameter $2r$ there are no points of this partition which are not members of that cluster which are within distance $2r$ of a member of the cluster. So each cluster consists of at most $k + 1$ points, each of whose k nearest neighbors are within the cluster, i.e., are mutually k-close.

We will show that at most c partitions P_i are created by the partitioning al-

partition(point set P, k)

```
for all p ∈ P                          // Determine the k nearest neighbors and the radius
   determine NN_k(p) and r_k(p)        // of the k nearest neighbors ball based on the original
i = 1                                  // point set. These values do not change.
while P ≠ ∅                            // While unpartitioned points remain
   unmarked(P) = P                     //    unmark all remaining points.
   P_i = ∅                             //    Create a new, empty partition.
   while unmarked(P) ≠ ∅               //    While unmarked points remain
      r = min_{p∈unmarked(P)} r_k(p)   //       find the point p with the minimum radius (r)
      p' = p ∈ P : r = r_k(p)          //       nearest neighbor ball and add that point and
      P_i = P_i ∪ {p ∈ P : ||pp'|| ≤ r} //      all points within r to the new partition.
      P = P \ {p ∈ P : ||pp'|| ≤ r}    //       Remove these points from P and mark
      unmarked(P) = unmarked(P) \ {p ∈ unmarked(P) : ||pp'|| ≤ 3r}
   increment i                         //       points within 3r of p.
return {P_1, P_2, ..., P_c}            // Return the resulting partitions.
```

Figure 4.1: The partitioning algorithm which implements Lemma 4.2.

gorithm of Figure 4.1. We refer to each iteration of the outer while loop as a *round*. First note that at the end of the first round all points are either marked or removed from P. Each point that remains after the first round was marked by some point during the first round. Consider some point p which is marked by the first round. Let p' be the point that marked p in round one, and let $r = r_k(p')$ be the radius of $NN_k(p')$. (Note that for any $p \in P$, $r_k(p)$ is defined based on the original point set.) The marked point p is within distance $3r$ of p'. Since there are k points within

distance r of p', there are at least k points within distance $4r$ of p, so $NN_k(p)$ cannot have a radius larger than $4r$. Let M be the set of points that mark p in any round i. Since nearest neighbor balls are chosen in increasing order of radius, no point q with $r_k(q)$ greater than $4r$ can be in M, since p would have been chosen first. So all points in M are within distance $3 \cdot 4r = 12r$ of p.

We now show that points in M are r-sparse, i.e., are separated by at least distance r. Since radii are chosen in increasing order and $r = r_k(p')$ has already been chosen, any point $q' \in M$ must have $r_k(q') > r$. In addition, since all points in $NN_k(q')$ are removed, for all $q'' \in M$, $\|q'q''\| > r$. Given a circle of radius R in a doubling metric, it follows from a standard packing argument that any δ-sparse set that lies entirely within a ball of radius R has cardinality $O(1 + (R/\delta)^{O(1)})$. Taking $R = 12r$ and $\delta = r$, we have that $|M| \leq O(1 + 12^{O(1)}) = O(1)$. For points in $\mathbb{R}^d$ this constant is $1 + 12^d$. Letting c denote this quantity, we see that no point can be marked more than c times, and hence the process terminates after at most c rounds, producing at most c partitions. □

Note that a cluster centered at p' with less than $k + 1$ points does not violate the k-clusterable property since this cluster would have been created by clustering $NN_k(p')$ together as originally identified before any points were partitioned. A cluster without $k + 1$ points is formed because some of the original points in $NN_k(p')$ were previously added to a different partition. Since being mutual k-close is based on the entire set, smaller clusters are still mutually k-close within that partition.

Figure accompanying the proof of Lemma 4.2

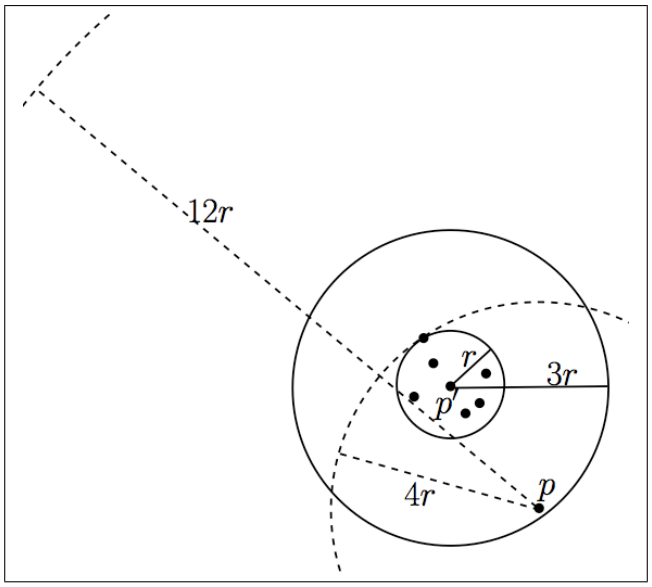

Figure 4.2: Proof illustration for Lemma 4.2 for $k = 6$. Solid circles are centered at point p'. Solid lines show the radii of these circles. Dashed arcs are partial circles centered at point p. Dashed lines show the radii of these circles.

4.3.2 Compression Theorem

We now present the main compression algorithm and analysis. The algorithm, presented in Figure 4.3, compresses each cluster formed by the partitioning algorithm (Figure 4.1) separately and returns the union of these. Each cluster is compressed by creating a new stream in which the t^{th} character is a new character which is the concatenation of the t^{th} character of every stream in that cluster. This new stream is then compressed using an entropy-based compression algorithm which achieves the optimal encoding length in the limit. For example, the Lempel-Ziv sliding-window compression algorithm could be used [ZL77]. We reason about

compress(stream set X, sensor set P, k)

$\{P_1, P_2, \ldots, P_c\}$ = partition (P, k)

for $i = 1$ to c

for all clusters j in P_i containing streams X_{ij1} through $X_{ijh_{ij}}$

$\widehat{X}_{ij} = \bigcup_{t=1}^{T} X_{ij1t}\&X_{ij2t}\&\ldots\&X_{ijh_{ijt}}$ where X_{ijht} is the t^{th} character of X_{ijh}

return $\bigcup_{ij}$ *entropy_compress*$(\widehat{X}_{ij})$

Figure 4.3: The compression algorithm which takes a set X of streams of length T and the associated set P of sensors which recorded them and returns a compressed encoding of length $c \cdot H$, where H is the joint entropy of the streams. The partitioning algorithm shown in Figure 4.1 is called and determines the constant c and represents the concatenation of characters to create a larger character. *entropy_compress* is an entropy-based compression algorithm which achieves the optimal encoding length in the limit and returns an encoded stream.

the size of the resulting stream set encoding.

First, we introduce some notation. Let X be the set of streams containing the information recorded by the sensors of set P where $|X| = |P|$. Given the set of partitions $\{P_i\}$ resulting from the partitioning lemma in Section 4.3.1, $\{X_i\}$ is the set of associated streams. Let $\{C_{ij}\}$ be the set of clusters that are created by the partitioning algorithm, we call $\{X_{ij}\}$ the set of streams in cluster C_{ij} and X_{ijh} is the h^{th} stream in cluster C_{ij} with cardinality h_{ij}.

Theorem 4.1. *A set of streams which represent observations from a k-local sensor system can be compressed to an encoded string which has length at most c times the optimal, where c is a constant depending on the doubling dimension of the underlying*

point set.

Proof. First, we show that each cluster C_{ij} is compressed to a string whose length is equal to the joint entropy of the component streams of that cluster. Each cluster consists of streams $\{X_{ij}\}$ which are merged into one new stream by concatenating the t^{th} character of all the streams to create the t^{th} character of the new stream. This new stream, $\widehat{X}_{ij}$, is then compressed using an optimal compression algorithm. By construction of the streams $\widehat{X}_{ij}$, the entropy $H(\widehat{X}_{ij})$ of a single stream is equal to the joint entropy of its component streams $H(X_{ij1}, X_{ij2}, \ldots, X_{ijh_{ij}})$. The entropy-based encoding algorithm compresses each $\widehat{X}_{ij}$ to an encoded string the length of the stream's entropy and that compression is optimal [WZ94], so $H(X_{ij1}, X_{ij2}, \ldots, X_{ijh_{ij}})$ is the optimal encoding length for cluster C_{ij}.

Our local dependence assumptions, explained in Section 4.2, say that the stream of data from a sensor is only dependent on the streams of its k nearest neighbors. Additionally, recall that in Section 4.2 we defined being mutually k-close to require that streams are only dependent if they come from sensors who are in each other's k nearest neighbor sets. By the partitioning lemma from Section 4.3.1, we know that each cluster C_{ij} is independent of all other clusters in partition P_i. From standard information theoretic results [CT06] we know that for a collection of streams $Y_1, \ldots, Y_S$, $H(Y_1, Y_2, \ldots, Y_S) = \sum_{i=1}^{S} H(Y_i)$ if and only if the Y_i are independent. Since the elements of $\{\{X_{i1}\}, \{X_{i2}\}, \ldots, \{X_{i|\{C_{ij}\}|}\}\}$ are independent, $H(X_i) = \sum_j H(\{X_{ij}\})$. Combining this with the fact that $H(\widehat{X}_{ij})$ is equal to the joint entropy of its component streams, we have that $H(X_i) = \sum_j H(\widehat{X}_{ij})$. $H(X_i)$

is the optimal compression bound for partition P_i, so we achieve the optimal compression for each partition.

Finally, we show that our compression algorithm is a c-approximation of the optimal. We say that a compression algorithm provides a γ-*approximation* if the length of the compressed streams is no more than γ times the optimal length. Recall that c partitions are generated by the partitioning algorithm from Section 4.3.1. Each of these partitions is encoded by a string of length $H(X_i)$ in the limit, so the total encoding size is $\sum_{i=1}^{c} H(X_i) \leq c \cdot \max_i H(X_i) \leq c \cdot H(X)$, where $H(X)$ is the joint entropy, which is a lower bound on the optimal encoding size, and the last inequality follows since $|X| \geq |X_i|$ for all i. So our algorithm provides a c-approximation of the optimal compression. □

Note that using the same method we used to compress the members of individual clusters, we could have combined the characters of all streams and compressed these together. This method would have optimal compression to the joint entropy of the streams. For demonstration of the problem with this method, consider the Lempel-Ziv sliding-window algorithm [ZL77]. The algorithm proceeds by looking for matches between the current time position and some previous time within a given window into the past. The length and position of these matches are then recorded, which saves the space of encoding each character. The window moves forward as time progresses. Larger window sizes yield better results since matches are more likely to be found. The optimal encoded length is achieved by taking the limit as the window size tends to infinity [WZ94]. If all streams are compressed at once, the

optimal compression rate is only achieved in the limit as the window size becomes large and in practice compressing all streams at once requires a much larger window before the compression benefits begin. By only compressing k streams together we limit the effect of this problem.

4.4 Efficiency with Respect to Short-Haul KDS

We believe that k-local entropy is a reasonable measure of the complexity of geometric motion. It might seem at first that any system that is based on monitoring the motion of a large number of moving objects by the incremental counts of a large number of sensors would produce such a huge volume of data that it would be utterly impractical as a basis for computation. Indeed, this is why compression is such an important ingredient in our framework. But, is it reasonable to assume that lossless compression can achieve the desired degree of data reduction needed to make this scheme competitive with purely prescriptive methods such as KDS? In this section, we consider a simple comparison, which suggests that lossless compression can achieve nearly the same bit rates as KDS would need to describe the motion of moving objects.

This may seem like comparing "apples and oranges," since KDS assumes precise knowledge of the future motion of objects through the use of flight plans. In contrast, our framework has no precise knowledge of individual point motions (only the occupancy counts of sensor regions) and must have the flexibility to cope with whatever motion is presented to it. Our analysis will exploit the fact that, if the mo-

tion of each point can be prescribed, then the resulting system must have relatively low entropy. To make the comparison fair, we will need to impose some constraints on the nature of the point motion and the sensor layout. First, to model limited statistical dependence we assume that points change their motion plans after traveling some local distance threshold ℓ. Second, we assume that sensor regions are modeled as disks of constant radius, and (again to limit statistical dependence) not too many disks overlap the same region of space. These assumptions are not part of our framework. They are just useful for this comparison.

Here we will assume that flight plans are linear and that motion is in the plane, but generalizations are not difficult. Let Q denote a collection of n moving objects over some long time period $0 \leq t \leq T$. We assume that the location of the ith object is broken into some number of linear *segments*, each represented by a sequence of tuples $(\boldsymbol{u}_{i,j}, \boldsymbol{v}_{i,j}, t_{i,j}) \in (\mathbb{Z}^2, \mathbb{Z}^2, \mathbb{Z}^+)$, which indicates that in the time interval $t \in (t_{i,j-1}, t_{i,j}]$, the ith object is located at the point $\boldsymbol{u}_{i,j} + t \cdot \boldsymbol{v}_{i,j}$. (Let $t_{i,0} = 0$.) We assume that all these quantities are integers and that the coordinates of $\boldsymbol{u}_{i,j}$, $\boldsymbol{v}_{i,j}$ are each representable with at most b bits. Let $\Delta_{i,j} = t_{i,j} - t_{i,j-1}$ denote the length of the j time interval for the ith point.

In most real motion systems objects change velocities periodically. To model this, we assume we are given a locality parameter ℓ for the system, and we assume that the maximum length of any segment (that is, $\max_{i,j} \Delta_{i,j} \cdot \|\boldsymbol{v}_{i,j}\|$) is at most ℓ. Let m be the minimum number of segments that need to be encoded for any single object. Assuming a fix-length encoding of the numeric values, each segment requires at least $4b$ bits to encode, which implies that the number of bits needed to encode

the entire system of n objects for a single time step is at least

$$B_{\text{KDS}}(n, \ell) \geq \frac{4n \cdot m \cdot b}{T}.$$

We call this the *short-haul KDS bit rate* for this system.

In order to model such a scenario within our framework, let P denote a collection of S sensors in the plane. Let us assume that each sensor region is a disk of radius λ. We may assume that the flight plans have been drawn according to some stable random process, so that the sensor output streams satisfy the assumptions of stationarity and ergodicity. We will need to add the reasonable assumption that the sensors are not too densely clustered (since our notion of locality is based on k-nearest neighbors and not on an arbitrary distance threshold.) More formally, we assume that, for some constant $\gamma \geq 1$, any disk of radius $r > 0$ intersects at most $\gamma \lceil r/\lambda \rceil^2$ sensor regions. Let $\boldsymbol{X} = (X_1, X_2, \ldots, X_S)$ denote the resulting collection of sensor output streams, and let $H_k(n, \ell) \stackrel{\text{def}}{=} H_k(\boldsymbol{X})$ denote the normalized k-local entropy of the resulting system. Our main result shows that the k-local entropy is within a constant factor of the short-haul KDS bit rate, and thus is a reasonably efficient measure of motion complexity even when compared to an approach based on prescribing the motions.

Theorem 4.2. *Consider a short-haul KDS and the sensor-based systems defined above. Then for all sufficiently large k*

$$H_k(n, \ell) \;\leq\; \left(\frac{4\,\ell}{\lambda} \sqrt{\frac{\gamma}{k}} + 1 \right) B_{\text{KDS}}(n, \ell).$$

Before giving the proof, observe that this implies that if the locality parameter

k grows proportionally to $(\ell/\lambda)^2$, then we can encode the observed continuous motion as efficiently as its raw representation. That is, k should be proportional to the square of the number of sensors needed to cover each segment of linear motion. Note that this is independent of the number of sensors and the number of moving objects. It is also important to note that this is independent of the sensor sampling rate. Doubling the sampling frequency will double the size of the raw data set, but it does not increase the information content, and hence does not increase the system entropy.

Corollary 1. *By selecting* $k = \Omega((\ell/\lambda)^2)$*, we have* $H_k(n, \ell) = O(B_{\text{KDS}}(n, \ell))$.

Proof. Consider an arbitrary moving object j of the system, and let $X_{i,j}$ denote the 0–1 observation counts for sensor i considering just this one object. Let $\boldsymbol{X}_{(j)} = (X_{1,j}, X_{2,j}, \ldots, X_{S,j})$ denote the resulting single-object sensor system. Clearly, $H_k(\boldsymbol{X}) \leq \sum_{j=1}^{n} H_k(\boldsymbol{X}_{(j)})$, since the latter is an upper bound on the joint k-local entropy of the entire system, and the sum of observations cannot have greater entropy than the joint system since the sum generally contains less information than the individual observations.

Let m_j be the number of segments representing the motion of object j. Each segment is of length $\leq \ell$. Consider the per-object KDS bit-rate for object j, denoted $B_{\text{KDS}}(j)$. Note that KDS considers the motion of each object individually, so $B_{\text{KDS}} = \sum_{j=1}^{n} B_{\text{KDS}}(j)$. KDS requires $4b$ bits per segment, so $B_{\text{KDS}}(j) \geq \frac{4 \cdot b \cdot m_j}{T}$. Let $\ell' = (\lambda/4)\sqrt{k/\gamma}$. Observe that $\ell' > 0$. Subdivide each of the m_j segments into at most $\lfloor \ell/\ell' \rfloor$ subsegments of length ℓ' and at most one of length less than ℓ' . Then there are a total of at most $m_j(\ell/\ell' + 1)$ subsegments of length ℓ'.

We claim that the joint entropy of the sensors whose regions intersect each subsegment is at most $4b$. To see this, observe that there are 2^{4b} possible linear paths upon which the object may be moving, and each choice completely determines the output of all these sensors (in this single-object system). The entropy is maximized when all paths have equal probability, which implies that the joint entropy is $\log_2 2^{4b} = 4b$. Recall that at most $\gamma\lceil r/\lambda\rceil^2$ sensor regions intersect any disk of radius r. Let $r = \ell'$ be the radius of a disk that covers a subsegment. Then at most $\gamma\lceil(1/4)\sqrt{k/\gamma}\rceil^2$ sensor regions can intersect some subsegment. We assert that all sensors intersecting this subsegment are mutually k-close. To see this, consider some sensors s_1 and s_2, with sensing region centers c_1 and c_2 respectively, that intersect such a subsegment. Observe that, by the triangle inequality, $||c_1c_2|| \leq 2\lambda + \ell'$. Recall that $\ell' = (\lambda/4)\sqrt{k/\gamma}$. Choosing $k \geq 16\gamma$, this means that $\lambda \leq \ell'$, so $2\lambda + \ell' \leq 3\ell'$. Thus, for each sensor s_1 whose region overlaps this subsegment, the centers of the other overlapping sensor regions lie within a disk of radius $3\ell'$ centered at c_1. In order to establish our assertion, it suffices to show that the number of sensor centers lying within such a disk is at most k. Again recall that at most $\gamma\lceil r/\lambda\rceil^2$ sensor regions intersect any disk of radius r, so at most $\gamma\lceil(3/4)\sqrt{k/\gamma}\rceil^2$ sensor regions intersect the disk of radius $3\ell'$. Under our assumption that $k \geq 16\gamma$, it is easy to verify that $\gamma\lceil(3/4)\sqrt{k/\gamma}\rceil^2 \leq k$, as desired. Since the overlapping sensors are all mutually k-close, their k-local entropy is equal to their joint entropy, and so the the k-local entropy is also at most $4b$. Thus, to establish an upper bound on $H_k(\boldsymbol{X}_{(j)})$, it suffices to multiply the total number of subsegments

by $4b$ and normalize by dividing by T. So we have

$$H_k(\boldsymbol{X}_{(j)}) \leq \left(\frac{\ell}{\ell'}+1\right)\frac{4bm_j}{T} \leq \left(\frac{\ell}{\ell'}+1\right)B_{\mathrm{KDS}}(j) = \left(\frac{4\ell}{\lambda}\sqrt{\frac{\gamma}{k}}+1\right)B_{\mathrm{KDS}}(j).$$

Considering the normalized k-local entropy of the entire system, we have

$$H_k(\boldsymbol{X}) \leq \sum_{j=1}^{n}\left(\frac{4\ell}{\lambda}\sqrt{\frac{\gamma}{k}}+1\right)B_{\mathrm{KDS}}(j) = \left(\frac{4\ell}{\lambda}\sqrt{\frac{\gamma}{k}}+1\right)B_{\mathrm{KDS}},$$

which completes the proof. □

Chapter 5

Realistic Issues in Compression of Kinetic Sensor Data

We introduce a realistic analysis for a framework for storing and processing kinetic data observed by sensor networks. The massive data sets generated by these networks motivate a significant need for compression. We are interested in the kinetic data generated by a finite set of objects moving through space. Our previously introduced framework and accompanying compression algorithm (see Chapter 4) assumed a given set of sensors, each of which continuously observes these moving objects in its surrounding region. The model relies purely on sensor observations; it allows points to move freely and requires no advance notification of motion plans.

Here, we extend the initial theoretical analysis of this framework and compression scheme to a more realistic setting. We extend the current understanding of empirical entropy to introduce definitions for joint empirical entropy, conditional empirical entropy, and empirical independence. We also introduce a notion of limited independence between the outputs of the sensors in the system. We show that, even with this notion of limited independence and in both the statistical and empirical settings, the previously introduced compression algorithm achieves an encoding size on the order of the optimal. We present experiments that show that in practice

the constant factor associated with this encoding size is small and that the locality assumption of the model is reasonable.

5.1 Introduction

In Chapter 4 we considered the issues of collection and compression of kinetic sensor data under a theoretical analysis. The framework we presented is purely observational; it relies on no assumptions or advance knowledge about the underlying object motion. We introduced a lossless compression algorithm within this framework and showed that, when considered in terms of the Shannon entropy, the compression algorithm achieved storage space on the order of the optimal joint entropy bound. This compression algorithm relied on the assumption that a sensor's output is statistically dependent only on the output of other locally close sensors.

While the framework presented in Chapter 4 provided theoretical guarantees on the compression rates it achieves, it is based on a theoretical model of sensor networks that may not be satisfied in practice. In particular, it has two significant drawbacks. The first is an analysis based in the statistical setting using Shannon entropy and its extensions. These entropy definitions assume an underlying random process that generates the data, and when analyzing a specific data set the probabilities associated with each random variable are known in advance; when considering observed sensor data, this assumption is unrealistic. We extend the framework analysis to hold under the more realistic definition of *empirical entropy* [KM99] that has the advantage of not assuming an underlying stationary, ergodic random process.

Empirical entropy relies only on the observed probabilities of the sensor data values. In order to perform the complex analyses for the framework in the empirical setting, we also introduce new definitions for empirical entropy constructs that are analogous to existing statistical ones: joint empirical entropy, conditional empirical entropy, and empirical independence.

The second modification to the previously introduced framework that should be made in order to create a more realistic analysis concerns its assumptions of independence. The framework makes the assumption that sensor outputs are dependent only on their neighbors and are purely independent of all other outputs. However, it may be the case that there is some underlying dependence that may be common to many or all sensor outputs. For example, if the sensors are detecting and reporting car traffic counts, while nearby sensors may be more likely to see the same traffic patterns at consecutive time intervals, all sensors are likely to see a decrease in traffic at night and increases during rush hours. In order to analyze these underlying commonalities in the context of the framework for kinetic sensor data we introduce a notion of *limited independence* in both the statistical and empirical settings. We also verify experimentally that the locality assumption that nearby sensors have more related outputs than distant sensors holds.

With the addition of the realistic assumptions of empirical entropy and limited independence, we revisit the space bounds for the framework compression algorithm and prove that the encoding size is on the order of the optimal size under an assumption of limited independence for both the statistical and empirical settings. We also show that in practice the constant factor associated with this encoding size

is small. These extensions confirm that the framework and its accompanying compression scheme are realistic for use with kinetic sensor data. These new bounds on the encoding size have proved useful in space and time analyses for spatio-temporal range searching over compressed kinetic sensor data (see Chapter 6).

In summary, this chapter makes the following contributions to the understanding and realistic analysis of kinetic sensor data:

- An extension of the definition of empirical entropy to definitions for joint empirical entropy, conditional empirical entropy, and empirical independence. (See Section 5.3.)

- An analysis of a lossless compression algorithm based on empirical entropy within a sensor-based framework for kinetic data. (See Section 5.5.)

- The introduction of a notion of limited independence between sensor outputs in both statistical and empirical settings. (See Section 5.4.)

- The analysis of compression space bounds taking the notion of limited independence into account in both statistical and empirical settings. (See Section 5.5.)

- Experimental analyses of kinetic sensor data locality and the constant factor for a lossless compression algorithm's encoding size bound. (See Section 5.6.)

5.2 Statistical Setting

As a point of reference, we begin by considering entropy and independence in the traditional statistical setting. Recall from Chapter 2.4 that in this setting a sensor's output stream is modeled by a stationary, ergodic random process X over an alphabet Σ 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, logarithms are taken base 2.) The *normalized statistical entropy* generalizes this to strings of increasing length:

$$\mathcal{H}_k(X) \;=\; -\frac{1}{k}\sum_{x\in\Sigma^k} p(x)\log p(x),$$

where in the standard definition, k is considered in the limit:

$$\mathcal{H}(X) \;=\; \lim_{k\to\infty} \mathcal{H}_k(X).$$

A fundamental fact from information theory is that this value represents the number of bits needed to encode a single character of the stream [CT06]. 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

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

where $p(x,y)$ denotes the joint probability of both x and y occurring. The normalized joint statistical entropy of a set of strings $\mathbf{X} = \{X_1,\ldots,X_Z\}$ is defined analogously and is denoted $\mathcal{H}(\mathbf{X})$.

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 $\mathcal{H}(X, Y) = \mathcal{H}(X) + \mathcal{H}(Y)$ [CT06]. The following technical result will be of later use.

Lemma 5.1. *Consider two sensor outputs X and Y over the same time period. Let $X + Y$ denote the componentwise sum of these streams. Then $\mathcal{H}(X + Y) \leq \mathcal{H}(X, Y) \leq \mathcal{H}(X) + \mathcal{H}(Y)$.*

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,

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

By basic properties of conditional entropy (see, e.g., [CT06]), we have

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

which establishes the second inequality. □

5.3 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 Σ 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 $\mathsf{p}_{\mathrm{X}}(x) = c(x)/(T-k)$ denote the *observed probability* of x in X. (When X is clear from context, we will express this as $\mathsf{p}(x)$.) Recall from Chapter 2.4.1 that following the definitions of Kosaraju and Manzini [KM99], the 0th order empirical entropy of a string X is defined to be

$$\mathsf{H}_0(X) \;=\; -\sum_{a\in\Sigma} \mathsf{p}(a)\log \mathsf{p}(a) = -\sum_{a\in\Sigma} \frac{c_0(a)}{T}\log\frac{c_0(a)}{T}\ .$$

For $a \in \Sigma$, let $\mathsf{p}_{\mathrm{X}}(a|x) = c(xa)/c(x)$ denote the observed probability that a is the next character of X immediately following x. The kth order empirical entropy is defined to be

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

As observed in Kosaraju and Manzini [KM99], it is easily verified that $T\cdot\mathsf{H}_k(X)$ 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. In the rest of this section, we introduce new extensions of these notions of empirical entropy to 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 $X[i\ldots i+k-1] = x$ and $Y[i\ldots i+k-1] = y$. Define $\mathsf{p}_{\mathrm{X,Y}}(x,y) = c(x,y)/(T-k)$. For $a, b \in \Sigma$, define $\mathsf{p}_{\mathrm{X,Y}}(a,b|x,y) = c(xa,yb)/c(x,y)$ 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

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

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

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

$$\mathsf{H}_k(X|Y) \;=\; -\frac{1}{T}\sum_{x,y\in\Sigma^k} c(x,y)\sum_{a,b\in\Sigma} \mathsf{p}_{\mathrm{X,Y}}(a,b|x,y)\log \mathsf{p}_{\mathrm{X,Y}}(x,a|y,b),$$

where we define $\mathsf{p}_{\mathrm{X,Y}}(x,a|y,b) = \mathsf{p}_{\mathrm{X,Y}}(a,b|x,y)/\mathsf{p}_{\mathrm{Y}}(b|y)$ to be the probability that a directly follows x in X given that b directly follows y in 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, $\mathsf{p}_{\mathrm{X,Y}}(x,y) = \mathsf{p}_{\mathrm{X}}(x)\mathsf{p}_{\mathrm{Y}}(y)$. If X and Y are empirically independent then this also implies that, for $a \in \Sigma$ and $b \in \Sigma$, $\mathsf{p}_{\mathrm{X,Y}}(a,b|x,y) = \mathsf{p}_{\mathrm{X}}(a|x)\mathsf{p}_{\mathrm{Y}}(b|y)$.

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

Lemma 5.2. *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,* $\mathsf{H}_k(X,Y) = \mathsf{H}_k(X) + \mathsf{H}_k(Y)$.

(*ii*) $\mathsf{H}_k(X,Y) = \mathsf{H}_k(X) + \mathsf{H}_k(Y|X)$.

(*iii*) $\mathsf{H}_k(X,Y) \leq \mathsf{H}_k(X) + \mathsf{H}_k(Y)$.

(*iv*) $\mathsf{H}_k(X+Y) \leq \mathsf{H}_k(X) + \mathsf{H}_k(Y)$.

Proof. We will not prove (i) here, since it will follow as a special case of Lemma 5.4 below (by setting $\delta = 0$). To prove (ii), observe that by manipulation of the definitions

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

Symmetrically, we have $\mathsf{H}_k(X,Y) = \mathsf{H}_k(Y) + \mathsf{H}_k(X|Y)$.

To prove (iii), using (ii) we need only prove that $\mathsf{H}_k(Y|X) \leq \mathsf{H}_k(Y)$. By definition, we have

$$\mathsf{H}_k(Y|X) = -\frac{1}{T}\sum_{x,y\in\Sigma^k} c(x,y)\left[\sum_{a,b\in\Sigma} \frac{c(xa,yb)}{c(x,y)}\log\frac{c(xa,yb)}{c(x,y)}\right] .$$

Since clearly $c(x,y) \leq c(y)$ for all x and y, this means that

$$\begin{aligned}\mathsf{H}_k(Y|X) &\leq -\frac{1}{T}\sum_{y\in\Sigma^k} c(y)\left[\sum_{b\in\Sigma}\frac{c(yb)}{c(y)}\log\frac{c(yb)}{c(y)}\right] \\ &= -\frac{1}{T}\sum_{y\in\Sigma^k} c(y)\left[\sum_{b\in\Sigma} p_Y(b|y)\log p_Y(b|y)\right] \\ &= \mathsf{H}_k(Y)\ ,\end{aligned}$$

which completes the proof of (iii).

To prove (iv), let $Z = X + Y$. By the definition of empirical entropy we have $\mathsf{H}_k(X+Y) =$

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

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 5.1, $\mathsf{p}_{\mathrm{X,Y}}(x, y) \leq \mathsf{p}_{\mathrm{Z}}(x + y)$. Substituting this relationship into our equation and, since we desire an upper bound, considering only cases in which

$$-\mathsf{p}_{\mathrm{X,Y}}(a + b|x + y) \log \mathsf{p}_{\mathrm{X,Y}}(a + b|x + y) \leq -\mathsf{p}_{\mathrm{X,Y}}(a, b|x, y) \log(\mathsf{p}_{\mathrm{X,Y}}(a, b|x, y)),$$

we find that

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

By (iii) we have $\mathsf{H}_k(X, Y) \leq \mathsf{H}_k(X) + \mathsf{H}_k(Y)$, which implies that $\mathsf{H}_k(X + Y) \leq \mathsf{H}_k(X) + \mathsf{H}_k(Y)$, as desired. □

5.4 Limited Independence

Perfect statistical or empirical independence is too strong an assumption to impose on sensor outputs. For example, if strings are drawn from independent sources, empirical independence will hold only in the limit. 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 $\mathbf{X} = \{X_1, X_2, ..., X_Z\}$ is *statistically δ-independent* if, for any k and outcomes $x_i \in \Sigma^k$,

$$(1 - \delta) \prod_{i=1}^{Z} p(x_i) \leq p(x_1, x_2, ..., x_Z) \leq (1 + \delta) \prod_{i=1}^{Z} p(x_i).$$

In the following lemma, we develop a relationship regarding the entropies of statistically δ-independent streams.

Lemma 5.3. *Given* $0 \leq \delta < 1$ *and a set of statistically* δ*-independent streams* $\mathbf{X} = \{X_1, X_2, ..., X_Z\}$, $(1-\delta)(\sum_{i=1}^{Z} \mathcal{H}(X_i)) - O(\delta) \ \leq \ \mathcal{H}(\mathbf{X}) \ \leq \ (1+\delta)(\sum_{i=1}^{Z} \mathcal{H}(X_i)) + O(\delta)$.

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

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

By the assumption of statistical δ-independence, and by manipulation of the definitions, we have

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

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

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

$$(1-\delta) \prod_{i=1}^{Z} \mathsf{p}(x_i) \leq \mathsf{p}(x_1, x_2, ..., x_Z) \leq (1+\delta) \prod_{i=1}^{Z} \mathsf{p}(x_i).$$

Lemma 5.4. *Given $0 \leq \delta < 1$, and a set of empirically δ-independent strings $\mathbf{X} = \{X_1, X_2, ..., X_Z\}$ for $X_i \in \Sigma^j$ where $j \leq k+1$,*

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

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

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

Let $\mathsf{p}(a,b|x,y) = \mathsf{p}_{\mathrm{X,Y}}(a,b|x,y)$, and recall that $\mathsf{p}_{\mathrm{X,Y}}(a,b|x,y) = c_0(xa,yb)/c(x,y)$ 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

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

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

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

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

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

Now we develop an upper bound on the earlier equation. Let

$$f = -\mathsf{p}(xa,yb)\log\frac{\mathsf{p}(xa,yb)}{\mathsf{p}(x,y)} = \mathsf{p}(xa,yb)\log\frac{\mathsf{p}(x,y)}{\mathsf{p}(xa,yb)}\ .$$

Then the equation we wish to bound is

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

where, by the definition of δ-independence,

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

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

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

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

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

Let

$$g(\delta) = \log\frac{1+\delta}{1-\delta} = \log\left(1+\frac{2\delta}{1-\delta}\right).$$

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

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

The proof that

$$(1-\delta)(\mathsf{H}_k(X)+\mathsf{H}_k(Y))-O(\delta) \;\leq\; \mathsf{H}_k(X,Y)$$

proceeds symmetrically. □

5.5 Compression Space Bounds

In this section will consider the encoding size that can be achieved by *PartitionCompress* (the compression algorithm introduced in Chapter 4). 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 Chapter 6. Also recall that *PartitionCompress* creates sets of jointly encoded streams representing clusters. We will consider the set of these streams $\mathbf{X} = \{X_1, ..., X_Z\}$ representing the clusters from a single one of the c created partitions.

5.5.1 Statistical Setting

Given a set of streams $\mathbf{X} = \{X_1, X_2, ..., X_Z\}$ from independent clusters in a single partition in a statistical setting, standard information theory results [CT06] tell us that the optimal encoded space is $\sum_{i=1}^{Z} \mathcal{H}(X_i)$ bits. Call this $S_{opt}(\mathbf{X})$. From Section 5.4, we know that the optimal space used by an encoded set of statistically

δ-independent streams $\mathbf{X}$ is $(1-\delta)\left(\sum_{i=1}^{Z}\mathcal{H}(X_i)\right) - O(\delta)$ 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, the Lempel-Ziv dictionary compression algorithm [ZL78].

Recall that 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 $p_i \in NN_m(p_j)$ 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. The compression of a single cluster of neighboring sensors may be performed using any string compression algorithm; to obtain the near optimal bound, this algorithm must compress streams to their optimal entropy bound. We will show that LZ78 is sufficient for our purposes.

We know from Chapter 4 that $\mathrm{Enc}_{opt}(\mathbf{X}) = O(\mathcal{H}(\mathbf{X}))$ 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 (δ, m)-local sensor system* to be the same as an m-local sensor system but with an assumption of δ-independence between the clusters instead of pure independence. We have the following theorem regarding the space used by *PartitionCompress*:

Theorem 5.1. *Given a set* $\mathbf{X}$ *of sensor outputs from a statistically* (δ, 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. Let $\mathbf{X} = \{X_1, ..., X_Z\}$ be the jointly encoded streams representing clusters in a single partition. The optimal space bound for that single partition is

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

while *PartitionCompress* achieves a bound of

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

for a single partition. The ratio is

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

The rest of the proof proceeds similarly to the proof of Theorem 5.3, but for $\mathcal{H}(X_i)$ instead of $\mathsf{H}_k(X_i)$ and with an extra constant factor hidden in the final bound. Note that since this analysis is only for a single partition, there is also a factor of c hidden in the final bound. □

The space established in Theorem 5.1 is the basic statistical encoded space bound. It hides constants that are exponential in m and the doubling dimension. As a direct consequence of Lemma 5.1 and Theorem 5.1 we have the following corollary:

Corollary 2. *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.*

5.5.2 Empirical Setting

In the rest of this section, we extend the results of Section 5.5.1 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^* over the alphabet Σ^Z created from the original set of strings by letting the ith character of the new string, for $1 \leq i \leq T$, be equal to a new character created by concatenating the ith character of each string in the original set. As mentioned earlier, the new string's optimal encoded space bound is $T \cdot \mathsf{H}_k(X^*)$.

Lemma 5.5. *Given a set of strings* $\mathbf{X}$ *and a string* X^* *created from* $\mathbf{X}$ *as described above,* $\mathsf{H}_k(X^*) = \mathsf{H}_k(\mathbf{X})$.

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^*, simultaneous occurrences appear for the same indices at which a single combined character appears in X^*. This observation implies that if $\mathsf{H}_k(\mathbf{X})$ is restated to refer to the characters appearing in X^*, $\mathsf{H}_k(X^*) = \mathsf{H}_k(\mathbf{X})$. □

Corollary 3. *The minimum number of bits to encode a set* $\mathbf{X}$ *of strings, assuming that each character depends only on the preceding* k *characters, is* $S_{opt}(\mathbf{X}) = T \cdot \mathsf{H}_k(\mathbf{X})$.

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 grow exponentially based on the size of the alphabet for each additional sensor stream. Instead, we use the more local approach of *PartitionCompress*.

We define an *empirically m-local sensor system* to be analogous to the definition of an m-local sensor system, but with an assumption of empirical independence instead of statistical independence. Similarly, an *empirically* (δ, m)*-local sensor system* assumes empirical δ-independence 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 [FM00] as the subroutine that jointly compresses the streams from a single cluster. The Ferragina and Manzini structure [FM00] gives an optimal space bound of $O(T \cdot \mathsf{H}_k(X_i)) + T \cdot o(1)$ where X_i is the merged stream for that single cluster. We are interested in developing a lower bound on the compression that can be achieved using *PartitionCompress* in an empirical setting. 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 $S_{opt}(\mathbf{X}) = T \cdot \sum_{i=1}^{Z} \mathsf{H}_k(X_i)$ space for a single partition. As a direct consequence, we have the following theorem.

Theorem 5.2. *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 5.2 and for Theorems 5.3 and 5.4 grow exponentially in m and the doubling dimension of the space in which the sensors reside. If we consider empirical δ-independence, then the lower bound achieved by the compression algorithm (over Z total clusters in a single partition) remains $O(T \cdot \sum_{i=1}^{Z} \mathsf{H}_k(X_i))$, but $\sum_{i=1}^{Z} \mathsf{H}_k(X_i)$ is not generally equal to $\mathsf{H}_k(\mathbf{X})$, and so an optimal algorithm may be able to reduce the bound due to the δ dependence allowed. By application of Lemma 5.4, an optimal algorithm's bound is $S_{opt}(\mathbf{X}, \delta) = T(1-\delta)\left(\sum_{i=1}^{Z} \mathsf{H}_k(X_i)\right) + T \cdot O(\delta)$. We have the following theorem regarding the compressed size of the sensor outputs.

Theorem 5.3. *Given a set* $\mathbf{X}$ *of sensor outputs from an empirically* (δ, 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 greatest advantage of the dependence between clusters. It would achieve a space bound of

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

for each partition. The *PartitionCompress* algorithm compresses each partition to space

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

The ratio is

$$\rho = \frac{S_{opt}(\mathbf{X})}{S_{opt}(\mathbf{X}, \delta)} = \frac{\sum_{i=1}^{Z} \mathsf{H}_k(X_i)}{(1-\delta)\left[\sum_{i=1}^{Z} \mathsf{H}_k(X_i)\right] - O(\delta)}.$$

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

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

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

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 $O(\delta) \geq \sum_{i=1}^{Z} \mathsf{H}_k(X_i)$.

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

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

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 total space bound is $\max\{O(\delta T), O\left(S_{opt}(\mathbf{X})\right)\}$. □

In this chapter, 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 [KM99] show that LZ78 does not achieve the optimal bound of $T \cdot \mathsf{H}_k(X)$, they show that it uses space at most $T \cdot \mathsf{H}_k(X) + O((T \log\log T)/\log T)$. In our context, this means that each cluster uses space $T \cdot \mathsf{H}_k(X) + O((T \log\log T)/\log T)$.

Theorem 5.4. *Given a set* $\mathbf{X} = \{X_1, X_2, ..., X_Z\}$ *of sensor outputs taken over a sufficiently long time* T *from an empirically* (δ, m)*-local sensor system, for any*

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

$$\begin{aligned}\mathrm{Enc}_{\mathrm{LZ78}}(\mathbf{X}) &= cT \sum_{i=1}^{Z} \left(\mathsf{H}_k(X_i) + O\left(\frac{\log\log T}{\log T} \right) \right) \\ &= O\left(\max\left\{ \delta T, S_{opt}(X, \delta), \frac{T \log\log T}{\log T} \right\} \right) \ \textit{bits.}\end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

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 $O(\delta) \geq \sum_{i=1}^{Z} \mathsf{H}_k(X_i)$.

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

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

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

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

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

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

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

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.

□

As a direct consequence of Lemma 5.2(iv) and Theorem 5.4 we have the following corollary:

Corollary 4. *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 $\mathrm{Enc}_{\mathrm{LZ78}}(\mathbf{X})$ in both statistical and empirical settings (Theorems 5.1 and 5.4 respectively). For future analyses we may 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, $d = O(T/\log T)$ [FM05], 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 $d \log d = \mathrm{Enc}_{\mathrm{LZ78}}(\mathbf{X})$.

5.6 Experimental Results

In order to provide evidence for the effectiveness of our methods, we have implemented our algorithms and provide an empirical analysis of the performance of our algorithms. Although our data sets are rather small, they are large enough to demonstrate the value of local clustering in compression. We consider experimental results on two data sets – a simulated data set consisting of 19 sensors observing equal-sized portions of a circular highway (simulated using the "intelligent driver" traffic model [Tre10, THH00]) and a data set of hallway observations from the Mitsubishi Electronic Research Laboratories (MERL) consisting of 213 sensors [WILW07]. Both of these data sets contain one count per second, representing the number of cars or people, respectively, within the sensor region during that

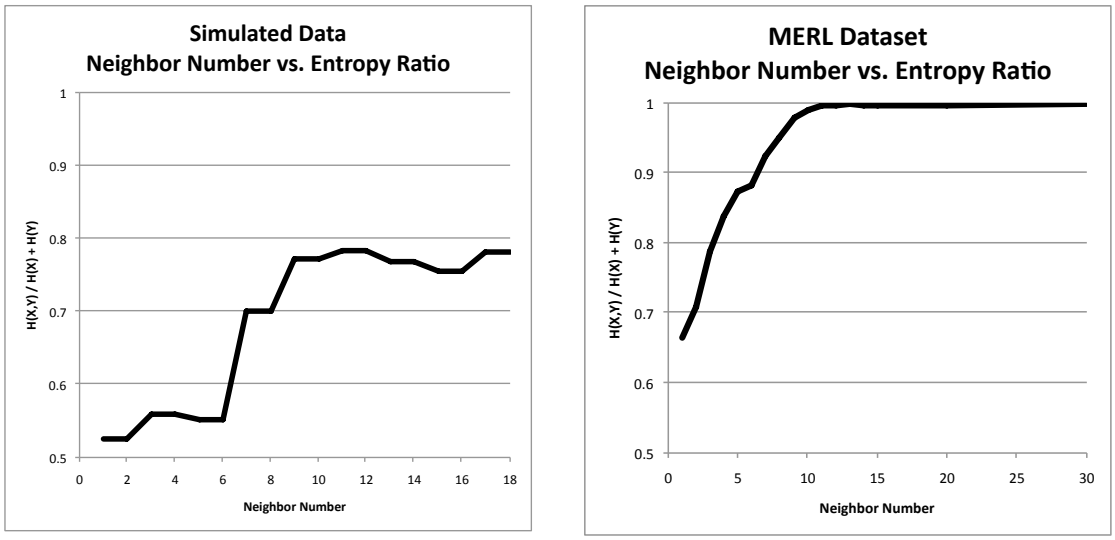

Figure 5.1: A comparison of neighbor numbers and an entropy ratio indicating locality for 10 days of data. *Left*: Simulated data. *Right*: MERL hallway data.

second. For the MERL data set, these counts were derived from activation times (given by epoch time stamps) by considering each sensor activation to contribute a count of one to the associated sensor region for that second. For the rest of this section, we consider the assumption of m-locality and the quality of the compression algorithm presented in Chapter 4 in the context of the simulated data and the collected hallway movement data.

We examine the assumption that the sensor systems are m-local, by considering the entropy relationship of a single sensor stream with that of its mth neighbor, for increasing values of m. Specifically, we consider the *entropy ratio*, the ratio of the pairwise joint empirical entropy (for $k = 4$) of the two sensors to the sum of their individual entropies for 10 days of data. This value can range from 0.5 to 1.0, and is low when two sensor outputs are statistically dependent and high when they

are close to independent. As shown in Figure 5.6, both the simulated data and the MERL data set have clear local neighborhoods where the entropy ratio is low, and as the neighbor number increases so does the entropy ratio. Our results show that, in both data sets, there is strong evidence in support of the underlying hypothesis of our framework, namely that nearby sensors have much lower joint entropy than distant ones. Thus, joint compression of local groups will achieve better compression rates.

Recall that in Chapter 4 we proved that *PartitionCompress* creates $c = O(1 + 12^{O(1)})$ partitions ($c = 1 + 12^d$ for points in $\mathbb{R}^d$) and that *PartitionCompress* compresses the sensor system to within c times the optimal joint entropy bound. This constant is also hidden in the space bounds proven in Section 5.5. For the simulated data, when considered over all possible values of m, c ranged from 1 to 4 with a median value of 3. For the MERL data set, when considered over values of m from 1 to 50, c ranged from 3 to 6 with a median value of 5. In comparison, the worst case bound gives a value of $c = 145$ for two-dimensional space. Thus, c is shown in practice to be much less than the worst case bound, and the compressed size of the data achieved by *PartitionCompress* is correspondingly only a small factor away from the optimal.

Chapter 6

Spatio-temporal Range Searching Over Compressed Kinetic Sensor Data

In Chapters 4 and 5 we introduced an algorithm which, given a set of sensor observations, losslessly compresses these data to a size that is within a constant factor of the asymptotically optimal joint entropy bound. In this chapter 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 joint entropy, queries can be answered in time that is roughly logarithmic in joint 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.

6.1 Introduction

The vast quantities of kinetic sensor data recorded necessitate compression of the sensor observations, yet analyses of these observations is desirable. Ideally, such analyses 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 chapter, we present the first data structure and algorithms for answering range searching queries over compressed data streams arising from large sensor networks.

In Chapters 4 and 5, we presented an algorithm for losslessly compressing kinetic sensor data and a framework for analyzing its performance. 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 [KL04].) 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; the problem considered in Chapter 4 is how to compress all these strings.

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. For example, a spatio-temporal range query might return the sum of the observed object counts within a spherical range of sensors over a given time period. 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 [AE98,Mat94]. One of the fundamental hierarchical structures, on which much other work has been based, is the quadtree, a data structure that divides the space into nested rectangular regions [Sam84]. The net-tree is a data structure similar in spirit, but for which the fundamental unit is the ball instead of the rectangle [HPM06]. (For more information about range searching, see Chapter 2.6.)

We present data structures for storing compressed sensor data and algorithms for performing spatio-temporal range queries over these 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 [Sha48] 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 [KM99, Man01], 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 Chapter 2.4.

Previously, retrieval over compressed text (without relying on decompression) has been studied in the context of strings [ABFC96, FM05, FV07, GN06] and XML files [FLMM06]. For example, Ferragina and Venturini [FV07] show that it is possible to retrieve a substring (indexed by start and end times) in the compressed text with query time equal to $O(1 + \frac{\ell}{\log T})$ where ℓ is the length of the substring and T is the length of the string X. Their space requirement is $T \cdot H_k(X) + o(T)$ bits. Their data structure allows substring queries, which are very different from semigroup range searching queries, which we consider here. More information about compressed text indexing can be found in Chapter 2.5.

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 [CHSV10] and combined with a spatio-temporal structure modified to be based on an I/O-efficient kd-tree [PAAV03, Rob81].

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 6.1: 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.

6.1.1 Contributions

In this chapter 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. [AM00]).

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 are 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 6.1. The

temporal range query data structures and associated bounds are discussed more specifically in Section 6.2 and the spatio-temporal results are discussed in Section 6.3.

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. These improvements increase as the data sets become larger. (See Section 6.4.)

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). The temporal range searching data structure relies on the trie created when compressing the data together with an annotated binary tree. The spatio-temporal range searching data structure relies on modifications to an existing quadtree-based data structure used for answering approximate range search queries as well as the temporal range searching solution. The use of these partition-tree based structures requires storage of an aggregated version of the encoded data at each level of the tree, and we show that this only increases the space used by a factor logarithmic in the number of sensors.

6.2 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, $\mathrm{Enc}(X)$ denotes the length of the compressed encoding of X.

Theorem 6.1. *There exists a temporal range searching data structure, which given string X over a time period of length T, can be built in time $O(\mathrm{Enc}(X))$, achieves query time $O(\log T)$, and uses space $O(\mathrm{Enc}(X))$ bits.*

The remainder of this section is devoted to proving this theorem. In Section 6.2.1 we consider the simpler special case where the semigroup is in fact a group, which means that both addition and subtraction of weights are allowed. In Section 6.2.2, we consider the general semigroup case, where only addition is allowed.

6.2.1 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 the standard accompanying trie (also known as a *dictionary*) containing nodes that represent *words* is created [ZL78]. 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* [FM05]. We will make use of the fact, proved in Chapter 5, 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 are 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. Here, we briefly present an example of this initial preprocessing step for $X =$ "12112312," with aggregation type a sum of counts. An accompanying trie is shown in Figure 6.1. The first word found while scanning through the input and entered into the trie as part of the LZ78 compression algorithm is "1" which has associated anchor point $\$_0$ and stored word count sum of 1. Similarly, the next word discovered is "2" with associated anchor

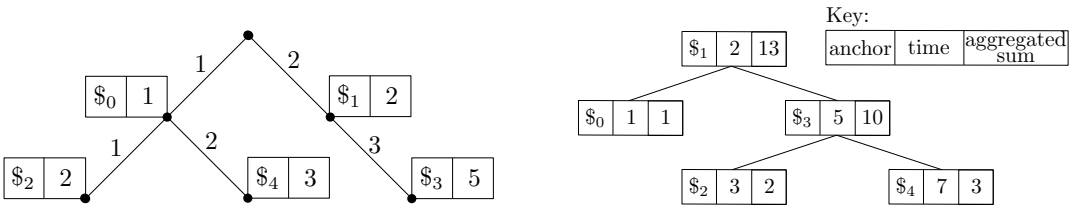

Figure 6.1: Left: LZ78 trie annotated with associated anchor points and word sums for a single sensor with observation string "12112312". Considered inline, the string with anchor points as breaks between the words becomes 1 $\$_0$ 2 $\$_1$ 11 $\$_2$ 23 $\$_3$ 12 $\$_4$. Right: The corresponding binary search tree based on word start times that also contains aggregated sums for the words contained in each node's subtree. The anchor points are also sorted by the word start times (in the order of their indices) and the start times are stored separately with the anchors so that $\$_0$ is associated with start time 1, $\$_1$ is associated with start time 2, $\$_2$ is associated with start time 3, and so on until $\$_4$ is associated with start time 7. Thus, a temporal range $[3, 7]$ would include parts of the words with anchor points $\$_2$, $\$_3$, and $\$_4$.

---

point $\$_1$ and word sum 2. The next word is "11" with anchor point $\$_2$ and word sum $1 + 1 = 2$, and so on until the word "12" ending at anchor point $\$_4$ with word sum $2 + 3 = 5$. LZ78 outputs a space-efficient encoding of this trie, but for our purposes it is not necessary to understand the actual encoding. See Figure 6.1 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. 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 [BFC04].

Lemma 6.1. *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. (Here, we will give some examples. The more precise explanation can be found in the proof of Lemma 6.2.) 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. For example, in Figure 1, consider the query $[5, 5]$, which effectively asks for the value of the fifth character of the string. This query overlaps the first character of the substring "23", and therefore it is an internal query. In this case, the prefix is null (since there are not characters of the substring preceding the query), the query region consists of "2", and the suffix consists of the remainder of the string, namely "3". Overlapping queries consist of a suffix, one or more complete words, and a prefix of the trailing word. For example, in Figure 6.1, the overlapping query $[4, 7]$, corresponding to the substring "1231," consists of the suffix "1" of the word "11," all of the word "23,"

and the prefix "1" of word "12."

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. For example, in Figure 6.1, the aggregate sum of the substring "112312," with temporal range $[3, 8]$ and consisting of the words with anchor points $\$_2$, $\$_3$, and $\$_4$, can be easily found to be 10 by a look-up in the binary search tree. Finally, suffix values can be retrieved by finding the complementary prefix value and subtracting from the total associated with the entire word. For example, the suffix sum of 1 indicated by the temporal range $[4, 4]$ in Figure 6.1 can be retrieved by subtracting the value 1 associated with $\$_0$ from the sum of 2 associated with $\$_2$. 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.

Lemma 6.2. *The query time for temporal range searching in the group setting is* $O(\log d) = O(\log T)$.

Proof. The compressed text is modified to be of the form $W_1\$W_2\$ \dots W_d\$$ where $\{W_1, ..., W_d\}$ 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 [FM05], and though pointers to

the beginning of words would suffice for our application, we use the insertion of \$'s for notational convenience.) Since each dictionary word appears exactly once in the compressed text, each word has a single associated anchor point.

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 [BFC04] for answering level-ancestor queries.

Finally, the prefix of the requested range, which is the suffix of the word w_0 beginning at $\$_0$, 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 [BFC04]. 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. The aggregation tree is a binary tree containing word sums as leaf nodes and aggregate values of all words in the associated subtree for each internal node. Using this data structure, the number of summed subtrees is $O(\log d)$. With this modification, 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. □

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 [BFC04]. 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 6.3. *The total space in bits required for our temporal range structure in the group setting is* $O(d \log d) = O(\mathrm{Enc}(X))$.

6.2.2 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." 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 Sleator and Tarjan link-cut tree [ST83]. They annotate edges along the tree to be either solid or dashed so that

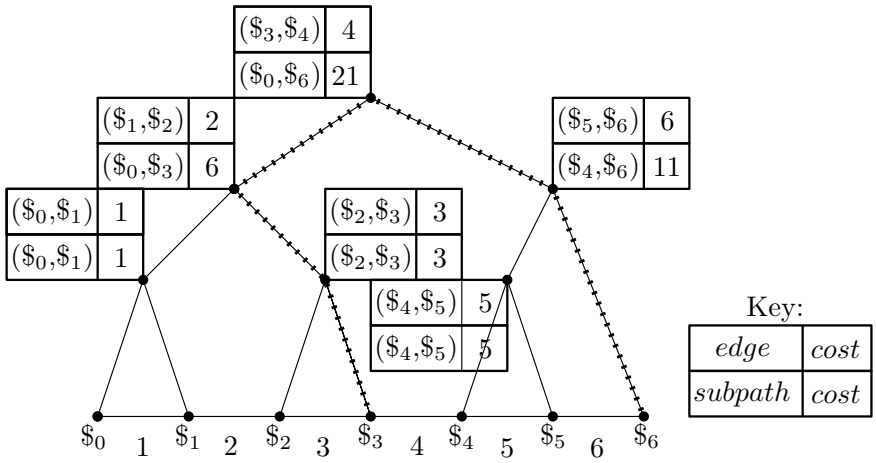

Figure 6.2: A solid path between anchors $\$_0$ and $\$_6$ along sensor output string "123456" with associated binary tree. Each node of the binary tree is annotated with its associated edge cost and its associated subpath cost. The search paths when finding the cost of the subpath between anchors $\$_3$ 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.

---

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 6.2). 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 6.2.

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 [ST83], 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 $t_0 - \$_0$ and $\$_1 - t_1$ since the queries are along a single path that is indicated by $\$_0$ and $\$_1$. 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 6.4. *The query time for temporal range searching in the semigroup setting is* $O(\log d) = O(\log T)$.

6.3 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.

Recall from the previous chapter that $\mathrm{Enc}_{alg}(\mathbf{X})$ denotes the length of the encoded set of sensor outputs $\mathbf{X}$, where alg specifies the string compressor used by the compression algorithm of Chapter 4. Since the LZ78 algorithm will suffice for our purposes, let $\mathrm{Enc}(\mathbf{X}) = \mathrm{Enc}_{\mathrm{LZ78}}(\mathbf{X})$. In Chapter 5, it is shown that $\mathrm{Enc}(\mathbf{X})$

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, is also considered and it is shown that the bounds hold approximately under this weaker definition.)

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.

Theorem 6.2. *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 Chapter 4 groups sensors into clusters, and the sensor outputs within each clus-

ter 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 Chapter 4 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 [AM00] 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. See Figure 6.3 for examples of these cases.

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 Chapter 4, 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

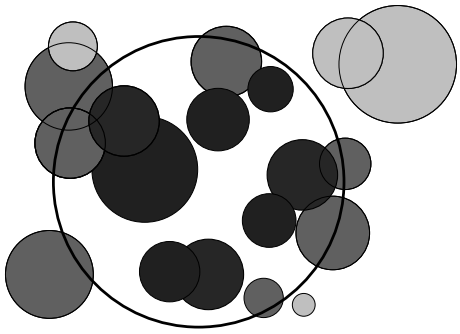

Figure 6.3: A set of clumps and a spherical range. The query range is indicated by the larger disc. The clumps are indicated by smaller balls and are shaded based on their membership in one of three groups; clumps that are outside of the range are light grey, clumps that are stabbed by the range are grey, and clumps that are entirely included in the given range are dark grey.

---

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. The proof of the following lemma 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. See Figure 6.4 for an accompanying diagram.

Lemma 6.5. *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 $P'_i \subseteq b_i$.*

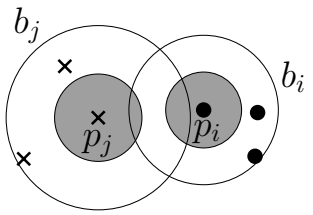

Figure 6.4: Two clumps b_i and b_j from a partition P' generated by *PartitionCompress*. Points represent sensor locations. The clump b_i centered at p_i contains cluster P'_i represented by the solid points. The clump b_j centered at p_j contains cluster P'_j represented by the points marked with a cross. The shaded disc centered at p_i is $\frac{1}{2}b_i$ and the shaded disc centered at p_j is $\frac{1}{2}b_j$.

Proof. 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 $P' = P'_1 \cup \ldots \cup P'_h$. (This produces one group. To form the next group, 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 group. See Chapter 4 for further details.)

We assert that, for $1 \leq i \leq h$, the balls $\frac{1}{2}b_1, \ldots, \frac{1}{2}b_h$ 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 $\|p_ip_j\| \geq 3r_i$. Also, since the ball of radius $\|p_ip_j\| + r_i$ 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 $r_j \leq \|p_ip_j\| + r_i$. Combining these observations we have

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

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. □

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.

Many data structures used in range searching are based on partition trees [AE98]. 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 6.6. *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 $O(h(S) \cdot pp(S) + \mathrm{Enc}(\mathbf{X}))$*, query time* $O(qt(S) \cdot \log T)$*, and space* $O(sp(S) + h(S) \cdot \mathrm{Enc}(\mathbf{X}))$ *bits.*

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 6.5 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 structure 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 of 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 ith element of the vector is the count of the ith 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 $\{p_1, \ldots, p_f\}$ stored in the leaves that are descended from u. Let $\{X_1, \ldots, X_f\}$ 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 be 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 each leaf node of (2), we invoke the associated auxiliary temporal range search structure for time interval $[t_0, t_1]$ 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 can be shown by basic properties of entropy 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 is at most $\text{Enc}(\mathbf{X})$ 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 6.1 to access each auxiliary data structure to answer the temporal range queries. This completes the proof. □

We claim that many standard partition-tree-based methods for approximate range searching can be adapted to perform range searching among clumps. 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 [AM00] 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 $\Omega(\varepsilon \cdot \text{diam}(Q))$ [AM00]. 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. (It may seem odd to consider approximate range searching in light of our emphasis on lossless compression. However, the data structures that we will describe below have the property that the value of ε is specified at query time, and it may even be that $\varepsilon = 0$. Thus, the data represented by the sensors may be extracted to any desired degree of precision.) 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 [AM00].

Many data structures, such as the range searching algorithm appearing in [AM00] 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 6.7. *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.*

Proof. Recall from [AM00] and from Chapter 2.6 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 [AM00] 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 [AM00], 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 b'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 6.5, 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 [AM00], 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 [AM00], it follows that the number of nodes visited by this algorithm is $O((1/\varepsilon^{d-1}) + \log n)$. □

By applying Lemma 6.6 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 $O((S \log^2 S) + \text{Enc}(\mathbf{X}))$, 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 geomet-

ric 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 6.2.

Because the above result relies only on basic packing properties of the BBD-tree data structure, it is easy to see that this result can be applied to other data structures for range searching that rely only on such packing properties. Examples include the BAR-trees [DGK01], the quadtree-based data structure of Chan [Cha98], quadtree-based solutions to absolute range searching [dFM10], and methods based on net-tree decompositions in metric spaces of constant doubling dimension [HPM06, GGN06].

6.4 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) [WILW07] consisting of activation times for sensors located in the hallways of their building, we analyzed two aspects of our data structure's performance; space and time. In short, we found 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. As in the experiments of Chapter 5, the data set considered here is not particularly large. Still, we believe that it is useful for a basic understanding of size and query time

comparisons.

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. Sensors observing adjacent areas of the hallway or hallway areas connected by doors to observed rooms or lounges are considered to be neighboring.

We considered the storage space per sensor data stream for the raw data (consisting of one value per second) versus the temporal range structure (specifically the annotated trie) written to a file. The average 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 6.4. We call the ratio between the space used by the raw data and the space used by the temporal range structure the *improvement ratio*. A graph showing the number of days of data considered versus this improvement ratio is given in Figure 6.4. 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

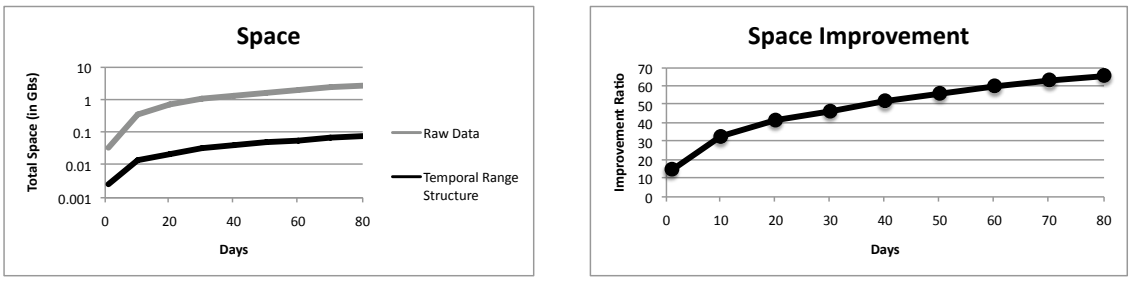

Figure 6.5: *Left:* The space used by the raw data in comparison to that used by the temporal range structure shown for varying numbers of days of data. Note that the size is shown in a logarithmic scale. *Right*: Space savings shown via a comparison of the number of days of data versus the ratio between the raw data and the temporal data structure sizes. As the number of days increase, the space saving increases as well.

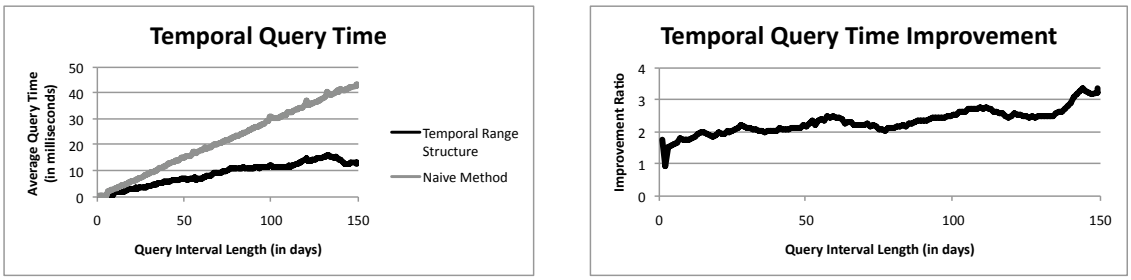

Figure 6.6: *Left*: Average temporal query times for 100 randomly chosen queries for each interval length. Query times using the temporal structure for aggregation over this query interval and using a naive method that simply aggregates one by one are shown. *Right*: The improvement ratio of the temporal query times, the naive time divided by the temporal range structure query time, is shown for each query interval from the corresponding temporal query times graph.

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 6.6. 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 slightly as well. The improvement ratio (the ratio of the naive query time to the temporal range structure time) shows that for most query intervals, the temporal range structure is twice as fast as the naive method (see Figure 6.6).

Chapter 7

Conclusion

In this thesis, I pursued two main avenues of research. First, I explored a problem within the intersection of robust statistics and algorithms for moving points from the point of view of the established KDS framework (see Chapter 3). Second, I developed a new observation-based framework for moving objects and considered compression and retrieval problems and analysis within that framework (see Chapters 4, 5, and 6).

7.1 Robust Kinetic Data Structures

My first work demonstrated the first kinetic robust approximation algorithm for the k-center problem with an approximation ratio of $(3 + \varepsilon)$ in the discrete case and $(4 + \varepsilon)$ in the absolute case for fixed k. In addition, I gave a $(3 + \varepsilon)$-approximation algorithm for the static robust k-center problem for fixed k and a $(4 + \varepsilon)$-approximation algorithm for the kinetic non-robust k-center problem for arbitrary k. My algorithm is one of the first to tackle the conflict between the necessarily local conditions required by the kinetic data structures framework to achieve responsive update rules and globally dependent statistical information.

The KDS algorithm allows for points to change their designation as outliers or

inliers over time. When considering traffic detection, outlier flexibility ensures that cars uncovered by the chosen k centers (cars that are not in traffic) may be covered at a later time (can get stuck in traffic) and vice versa. The robust kinetic k-center algorithm presented here can also be modified to handle k-center queries that have the added anonymity requirement that a given number of points must be served by a center for it to be added to the k-center solution [MK08]. This anonymity constraint can be satisfied by modifying the binary search through the hierarchy to find the appropriate level as presented in Section 3.3.3 so that all $\log \alpha$ levels are searched instead. The set of k centers at the lowest level that satisfy both the current constraints and this added anonymity constraint are output as representative for that spanner. This modification will not increase the KDS quality bounds but will increase the static algorithm runtime to $O((\log n \log \alpha)/\varepsilon^{2d})$.

7.1.1 Open Problems

Open problems in this area include revisions to the algorithm and/or kinetic framework to allow more efficient maintenance of global statistical properties under motion so that algorithms move farther away from their static counterparts and rarely need to invoke the static version of the algorithm. These modifications might additionally allow the k-center problem to be considered for arbitrary k. Currently, the algorithm assumes a fixed k, since allowing k to be arbitrary would yield an update time of $O(k(\log n \log \alpha)/\varepsilon^{2d})$ for the kinetic data structure's responsiveness. This update time is caused by re-running the static algorithm when changes to the

set of centers are suspected. Due to the global nature of the k-center problem and the sequential incremental nature of our algorithm, small changes to the set of centers cannot be fixed by local incremental manipulations of the solution. Removing the restriction on k would require a different, non sequential local solution.

In addition, it would be interesting to remove the dependence on the aspect ratio from the algorithm's time bounds. This dependence is due to the structure of the deformable spanner. Cole, Gottlieb, and Roddity [CG06, GR08] have worked on a dynamic hierarchical spanner that does not depend on the aspect ratio. Use of this spanner does not immediately yield a kinetic algorithm with no aspect ratio dependence. The Cole and Gottlieb [CG06] spanner makes use of path compression so that trivial spanner paths in which a point appears in multiple levels of the hierarchy without any children are represented as only the top and bottom of the path (Cole and Gottlieb call these jumps). However, the robust kinetic k-center algorithm requires a maintained priority queue for each level containing weights for each point at the level. Maintaining these priority queues would negate the space advantages of the spanner's path compression. Thus a significantly different strategy is needed to remove the aspect ratio dependence.

More generally, to my knowledge no other robust kinetic data structures problems have yet been considered. Other robust kinetic clustering problems, such as the robust k-median problem [CKMN01], remain open, as does the classic robust statistical problem of the least median-of-squares regression line estimator [Rou84]. The *least median of squares* estimator fits a line to a set of points by minimizing the median squared distance between any point and the line. This estimator is robust

for any data set containing up to 50 percent outliers [Rou84].

7.2 Observation-based Framework for Objects in Motion

I introduced a sensor-based framework for kinetic data which can handle unrestricted point motion and only relies on past information. I analyzed the framework's encoding size and gave a c-approximation algorithm for compressing point motion as recorded by the framework for a constant c. I also analyzed the framework from a realistic perspective, considering empirical entropy and a limited notion of independence.

Based on this framework and analysis, I presented the first spatio-temporal range searching structure for kinetic sensor data. This structure operates over the compressed version of the data without the need to decompress it. Preprocessing time and the data structure's space, measured in bits, were shown to be on the order of the encoding size. The query time was shown to be logarithmic in the observed length of time. Experimental results showed that the space used was at least an order of magnitude better than the space used by the raw data and that the query time was less than that of a naive method.

7.2.1 Open Problems

There are many open problems relating to the observation-based framework for kinetic data presented here. Some involve further work refining the framework itself, while others consider higher level statistical questions within the given framework.

The framework discussed here does not allow for tracking of individual points through the system; for example, the framework could not be used to determine which point travelled the farthest distance. This is a side-effect of the choice of discrete monitoring and the efforts towards practicality and robustness. While keeping these constraints in mind, it would be interesting to consider a tracking model that allows analysis of individual point properties. It would also be interesting to allow the tracking of some points while still monitoring other points discretely. This would allow for a more practical hybrid approach that could take advantage of all the information available. For example, scientists tagging fish [POS] could track those individual fish while not ignoring the surrounding school.

The theoretical and experimental results presented here were analyzed under main memory assumptions. While I expect much of this analysis to transfer directly to an I/O-efficient model, new data structures may also be required. For future work, it would be interesting to extend these analyses to an I/O-efficient model and conduct experiments using these modified data structures. In addition, the algorithms given on this framework were stated assuming a central processing node with global knowledge. It would be interesting to modify these algorithms to operate in a more distributed manner.

Any statistical analysis considered should operate over the compressed data without the need to decompress it. I suspect that a lossy compression algorithm instead of the lossless compression algorithm given by the preliminary work would greatly increase the efficiency of algorithms based on this framework, so lossy compression algorithms could also be considered and analyzed experimentally. In creat-

ing lossy compression algorithms within this observation-based framework it would be important to consider the domain-specific scientific ramifications of losing some data.

Open questions include solving global statistical questions on kinetic data using the observation-based framework, e.g., answering clustering questions, finding the centerpoint, or finding the point set diameter. More specifically, one interesting open question is developing algorithms to cluster over space and time. I am interested in the following problem definition with the goal of identifying clusters that are mutually predictable over space and time. This definition additionally approximates the location of objects by associating them with the sensor region that observes them and anonymizes objects by assuming no tracking of specific object identities from one sensor observation region to the next. Due to the entropy optimization problem at the core of this definition, this definition captures clusters that are co-located by both space and time (e.g. a flock of birds), clusters that are temporally co-located, but spatially distant (e.g. rush hour traffic patterns), and clusters that are spatially co-located but temporally separated (e.g. a spy and his target). Thus, I believe that this definition has the potential to capture many disparate and interesting spatio-temporal patterns and that it is, in this sense, the "correct" spatio-temporal clustering problem to consider.

Definition 7.2.1 (Spatio-temporal k-Center Problem)**.** *Given a set of S sensors with associated streams $\mathbf{X} = \{X_1, X_2, ..., X_S\}$ of counts over T time steps so that $X_i = X_{i1}, X_{i2}, ..., X_{iT}$, assign counts to k clusters (for each sensor at each time*

step) $C_{ij1}, C_{ij2}, ..., C_{ijk}$ *such that* $\sum_\ell C_{ij\ell} = X_{ij}$ *for all* i *and* j*, so as to minimize the maximum* τ*-order empirical entropy (*$H_\tau(\mathbf{X})$*) over all* k *of the stream sets* $\mathbf{C}_\ell = \{C_{ij\ell}\}_j$.

It would also be interesting to consider a generalization of this definition to include the consideration of outlying observations.

Definition 7.2.2 (Robust Spatio-temporal k-Center Problem)**.** *This problem generalizes the definition of the spatio-temporal* k*-center problem to the robust case where some percentage* q *of the observed counts (or* $Q = q \cdot \sum_{i=1}^{S} \sum_{j=1}^{T} X_{ij}$ *total observations) remain unassigned to any center.*

Solutions to these problems would likely be based on a range searching structure (presented in Chapter 6) that operates without decompressing the data and is built within this framework.

Finally, I believe that it is important to have theoretically sound yet practically reasonable frameworks for our observations of the world around us. Many problems remain to be considered within such a framework and other frameworks could still be created. The framework presented here makes one important step towards observing, collecting, and analyzing the ever present motion of objects.

Bibliography

[AB92] Amihood Amir and Gary Benson. Efficient two-dimensional compressed matching. In James A. Storer and Martin Cohn, editors, *Proceedings of the IEEE Data Compression Conference, DCC*, pages 279–288, Snowbird, Utah, 1992. IEEE Computer Society.

[ABB+04] Arvind Arasu, Brian Babcock, Shivanth Babu, Jon McAlister, and Jennifer Widom. Characterizing memory requirements for queries over continuous data streams. *ACM Transactions on Database Systems*, 29(1):162–194, June 2004.

[ABFC96] 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.

[ACI+00] Neal J. Alewine, James C. Colson, Abraham P. Ittycheriah, Stephane H. Maes, and Paul A. Moskowitz. Automated traffic mapping. U.S. Patent 6150961, filed Nov 24, 1998, and issued Nov 21, 2000.

[AdBG09] Mohammad Ali Abam, Mark de Berg, and Joachim Gudmundsson. A simple and efficient kinetic spanner. *Computational Geometry: Theory and Applications*, pages 306–310, April 2009.

[AE98] Pankaj K. Agarwal and Jeff 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.

[AGE+02] Pankaj K. Agarwal, Leonidas J. Guibas, Herbert Edelsbrunner, Jeff Erickson, Michael Isard, Sariel Har-Peled, John Hershberger, Christian Jensen, Lydia Kavraki, Patrice Koehl, Ming Lin, Dinesh Manocha, Dimitris Metaxas, Brian Mirtich, David M. Mount, S. Muthukrishnan, Dinesh Pai, Elisha Sacks, Jack Snoeyink, Subhash Suri, and Ouri Wolefson. Algorithmic issues in modeling motion. *ACM Computing Surveys*, 34:550–572, December 2002.

[AGG02] Pankaj K. Agarwal, Jie Gao, and Leonidas J. Guibas. Kinetic medians and kd-trees. In Rolf H. Möhring and Rajeev Raman, editors, *Proceedings of the 10th Annual European Symposium on Algorithms*, pages 5–16, Rome, Italy, 2002. Lecture Notes in Computer Science.

[AGMR98] Gerhard Albers, Leonidas J. Guibas, Joseph S. B. Mitchell, and Thomas Roos. Voronoi diagrams of moving points. *International Journal of Computational Geometry Applications*, 8(3):365–380, 1998.

[AM00] Sunil Arya and David M. Mount. Approximate range searching. *Computational Geometry: Theory and Applications*, 17:135–152, 2000.

[AMN+98] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[AP08] Pankaj K. Agarwal and Jeff M. Phillips. An efficient algorithm for 2D Euclidean 2-center with outliers. In Dan Halperin and Kurt Mehlhorn, editors, *Proceedings of the 16th Annual European Symposium on Algorithms*, pages 64–75, Karlsruhe, Germany, 2008. Lecture Notes in Computer Science.

[APKG07] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. Adaptively sampled particle fluids. *ACM Transactions on Graphics*, 26(3):48, 2007.

[ASSC02] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, 2002.

[Ata85] Mikhail J. Atallah. Some dynamic computational geometry poblems. *Computers & Mathematics with Applications*, 11(12):1171–1181, 1985.

[BBD+02] Brian Babcock, Shivanth Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first Symposium on Principles of Database Systems*, pages 1–16, Madison, Wisconsin, USA, 2002. ACM.

[BBKS00] Sergei Bespamyatnikh, Binay K. Bhattacharya, David G. Kirkpatrick, and Michael Segal. Mobile facility location. In *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M)*, pages 46–53, Boston, Massachusetts, USA, 2000. ACM.

[BCMR03] Dania Brunello, Giancarlo Calvagno, Gian A. Mian, and Roberto Rinaldo. Lossless compression of video using temporal information. *IEEE Transactions on Image Processing*, 12(2):132–139, Feb 2003.

[BE97] Marshall Bern and David Eppstein. Approximation algorithms for geometric problems. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, chapter 8. PWS publishing co., Boston, 1997.

[Ben75] Jon Louis Bentley. Multidimensional binary saerch trees used for associative searching. *Communications of the ACM*, 18(9):509–517, Sept. 1975.

[BFC04] Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theoretical Computer Science*, 321:5–12, 2004.

[BG99] Julien Basch and Leonidas J. Guibas. Data structures for mobile data. *Journal of Algorithms*, 31(1):1 – 28, April 1999.

[BGH97] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 747–756, New Orleans, Louisiana, USA, 1997. ACM.

[BGZ97] Julien Basch, Leonidas J. Guibas, and Li Zhang. Proximity problems on moving points. In *Proceedings of the Thirteenth Annual symposium on Computational Geometry*, pages 344–351, Nice, France, 1997. ACM Press.

[BO03] Brian Babcock and Chris Olston. Distributed top-k monitoring. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *Proceedings of the 2003 ACM SIGMOD International conference on management of data*, pages 28–39, San Diego, California, USA, 2003. ACM.

[BW94] M. Burrows and D. J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.

[CG06] Richard Cole and Lee-Ad Gottlieb. Searching dynamic point sets in spaces with bounded doubling dimension. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 574 – 583, Seattle, Washington, USA, 2006. ACM.

[Cha98] Timothy M. Chan. Approximate nearest neighbor queries revisited. *Discrete & Computational Geometry*, 20:359–373, 1998.

[Cha09] Bernard Chazelle. Natural algorithms. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 422–431, New York, NY, USA, 2009. SIAM.

[Che08] Ke Chen. A constant factor approximation algorithm for k-median clustering with outliers. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 826–835, San Francisco, California, USA, 2008. SIAM.

[CHM95] Jose Cuena, Josefa Hernandez, and Martin Molina. Knowledge-based models for adaptive traffic management systems. *Transportation Research Part C: Emerging Technologies*, 3(5):311–337, October 1995.

[CHSV08] Yu-Feng Chien, Wing-Kai Hon, Rahul Shah, and Jeffrey Scott Vitter. Geometric burrows-wheeler transform: Linking range searching and text

indexing. In *Proceedings of the Data Compression Conference*, pages 252–261, Snowbird, Utah, 2008. IEEE Computer Society.

[CHSV10] Sheng-Yuan Chiu, Wing-Kai Hon, Rahul Shah, and Jeffrey Scott Vitter. I/O-efficient compressed text indexes: From theory to practice. In James A. Storer and Michael W. Marcellin, editors, *Proceedings of the Data Compression Conference*, pages 426–434, Snowbird, Utah, 2010. IEEE Computer Society.

[CKMN01] Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms*, pages 642–651, Washington, DC, USA, 2001. ACM / SIAM.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2001.

[CMY08] Graham Cormode, S. Muthukrishnan, and Ke Yi. Algorithms for distributed functional monitoring. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1076–1085, San Francisco, California, USA, 2008. SIAM.

[CMZ07] Graham Cormode, S. Muthukrishnan, and Wei Zhuang. Conquering the divide: Continuous clustering of distributed data streams. In *Proceedings of the 23rd International Conference on Data Engineering*, pages 1036–1045, Istanbul, Turkey, 2007. IEEE Computer Society.

[CT06] T. M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-IEEE, second edition, 2006.

[CTS97] Michael M. Chang, A. Murat Tekalp, and M. Ibrahim Sezan. Simultaneous motion estimation and segmentation. *IEEE Transactions on Image Processing*, 6(9):1326–1333, 1997.

[DFHT05] Erik D. Demaine, Fedor V. Fomin, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. Fixed-parameter algorithms for the (k, r)-center in planar graphs and map graphs. *ACM Transactions on Algorithms*, 1(1):33–47, July 2005.

[dFM10] Guilherme D. da Fonseca and David M. Mount. Approximate range searching: The absolute model. *Computational Geometry: Theory and Applications*, 43:434–444, 2010.

[DGL10] Bastian Degener, Joachim Gehweiler, and Christiane Lammersen. The kinetic facility location problem. *Algorithmica*, 57(3):562–584, July 2010.

[DKR06] Antonios Deligiannakis, Yannis Kotidis, and Nick Roussopoulos. Processing approximate aggregate queries in wireless sensor networks. *Inf. Syst.*, 31(8):770–792, 2006.

[DKR07] Antonios Deligiannakis, Yannis Kotidis, and Nick Roussopoulos. Dissemination of compressed historical information in sensor networks. *The VLDB Journal*, 16(4):439–461, 2007.

[DM98] Yining Deng and B. S. Manjunath. NeTra-V: Toward an object-based video representation. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(5):616–627, 1998.

[EL75] Paul Erdös and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. *Infinite and Finite Sets*, Colloquia Mathematica Societatis János Bolyai 10:609–627, 1975.

[FG88] Thomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 434–444, Chicago, Illinois, 1988. ACM Press.

[FLMM05] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 184–196, 2005.

[FLMM06] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and searching XML data via two zips. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *Proceedings of the 15th International Conference on World Wide Web*, pages 751–760, Edingburgh, Scotland, UK, 2006. ACM.

[FM00] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science, FOCS*, pages 390–398, Redondo Beach, California, USA, 2000. IEEE Computer Society.

[FM05] Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, July 2005.

[FM09] Sorelle A. Friedler and David M. Mount. Compressing kinetic data from sensor networks. In Shlomi Dolev, editor, *Proceedings of the Fifth International Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors)*, volume 5804, pages 191 – 202, Rhodes, Greece, 2009. Lecture Notes in Computer Science.

[FM10a] Sorelle A. Friedler and David M. Mount. Approximation algorithm for the kinetic robust k-center problem. *Computational Geometry: Theory and Applications*, 43:572–586, 2010. doi: 10.1016/j.comgeo.2010.01.001.

[FM10b] Sorelle A. Friedler and David M. Mount. Spatio-temporal range searching over compressed kinetic sensor data. In Mark de Berg and Ulrich Meyer, editors, *Proceedings of the 18th Annual European Symposium on Algorithms*, volume 6346, pages 386–397, Liverpool, UK, Sept. 2010. Lecture Notes in Computer Science.

[FV07] Paolo Ferragina and Rossano Venturini. A simple storage scheme for strings achieving entropy bounds. *Theoretical Computer Science*, 372(1):115–121, March 2007.

[Gal91] Didier Le Gall. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, 1991.

[GGN06] Jie Gao, Leonidas J. Guibas, and An Nguyen. Deformable spanners and applications. *Computational Geometry: Theory and Applications*, 35(1):2–19, 2006.

[Gil06] Ralph Gillmann. Accuracy assessment of traffic monitoring devices vehicle by vehicle. *Transportation Research Record: Journal of the Transportation Research Board*, 1945:56–60, 2006.

[GJS96] Prosenjit Gupta, Ravi Janardan, and Michiel Smid. Fast algorithms for collision and proximity problems involving moving geometric objects. *Computational Geometry: Theory and Applications*, 6:371–391, 1996.

[GKS08] Sorabh Gandhi, Rajesh Kumar, and Subhash Suri. Target counting under minimal sensing: Complexity and approximations. *Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors)*, pages 30–42, 2008.

[GN06] Rodrigo González and Gonzalo Navarro. Statistical encoding of succinct data structures. *Combinatorial Pattern Matching*, pages 294–305, 2006.

[GNSL09] Sorabh Gandhi, Suman Nath, Subhash Suri, and Jie Liu. GAMPS: Compressing multi sensor data by grouping and amplitude scaling. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 771–784, Providence, Rhode Island, USA, 2009. ACM.

[Gol99] Andrew R. Golding. Automobile navigation system with dynamic traffic data. U.S. Patent 5933100, filed Dec 27, 1995, and issued Aug 3, 1999.

[Gon85] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[GR08] Lee-Ad Gottlieb and Liam Roditty. An optimal dynamic spanner for doubling metric spaces. In *Proceedings of the 16th Annual European Symposium on Algorithms*, volume 5193, pages 478–489, Karlsruhe, Germany, 2008. Lecture Notes in Computer Science.

[Gri98] A. P. Gribbon. Field test of nonintrusive traffic detection technologies. *Mathematical and Computer Modelling*, 27(9-11):349–352, 1998.

[GRS83] Leonidas J. Guibas, Kyle Ramshaw, and Jorge Stolfi. A kinetic framework for computational geometry. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 100–111, Tucson, Arizona, 1983. IEEE Computer Society.

[GTH08] Alexandre Guitton, Niki Trigoni, and Sven Helmer. Fault-tolerant compression algorithms for sensor networks with unreliable links. Technical Report BBKCS-08-01, Birkbeck, University of London, 2008.

[Gui98] Leonidas J. Guibas. Kinetic data structures: A state of the art report. In *Proceedings of the third workshop on the Algorithmic Foundations of Robotics*, pages 191–209, Houston, Texas, 1998.

[Gui02] Leonidas J. Guibas. Sensing, tracking and reasoning with relations. *IEEE Signal Processing Magazine*, 19(2):73–85, Mar 2002.

[Gui04] Leonidas J. Guibas. Kinetic data structures. In D. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, pages 23–1–23–18. Chapman and Hall/CRC, 2004.

[Ham71] Frank R. Hampel. A general qualitative definition of robustness. *The Annals of Mathematical Statistics*, 42(6):1887–1896, Dec 1971.

[Hoc95] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS publishing Co., Boston, 1995.

[HP04] Sariel Har-Peled. Clustering motion. *Discrete & Computational Geometry*, 31(4):545–565, 2004.

[HPM06] Sariel Har-Peled and Manor Mendel. Fast construction of nets in low dimensional metrics and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

[HS85] Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[HSV10] Wing-Kai Hon, Rahul Shah, and Jeffrey Scott Vitter. Compression, indexing, and retrieval for massive string data. In Amihood Amir and Laxmi Parida, editors, *Proceedings of the 21st Annual Symposium on Combinatorial Pattern Matching*, volume 6129, pages 260–274, New York, NY, USA, 2010. Lecture Notes in Computer Science.

[Huf52] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, Sept. 1952.

[JN06] Colette Johnen and Le Huy Nguyen. Self-stabilizing weight-based clustering algorithm for ad hoc sensor networks. *Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors)*, pages 83–94, 2006.

[Kah91] Simon Kahan. A model for data in motion. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 267–277, New Orleans, Louisiana, USA, 1991. ACM.

[KEK+98] Sung-Wook Kim, Yongsoon Eun, Hyungjin Kim, Jaein Ko, Wook-Jin Jung, Young Kyu Choi, Young Gil Cho, and Dong-Il Cho. Performance comparison of loop/piezo and ultrasonic sensor-based traffic detection systems for collecting individual vehicle information. In *Proceedings of 6th World Congress on Intelligent Transport Systems*, 1998.

[KH79] O. Kariv and S.L. Hakimi. An algorithmic approach to network location problems. Part 1: The p-centers. *SIAM Journal on Applied Mathematics*, 37:513–538, 1979.

[KL04] Robert Krauthgamer and James R. Lee. Navigating nets: Simple algorithms for proximity search. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 798 – 807, New Orleans, Louisiana, USA, 2004. SIAM.

[KM99] Rao S. Kosaraju and Giovanni Manzini. Compression of low entropy strings with Lempel–Ziv algorithms. *SIAM Journal on Computing*, 29(3):893–911, 1999.

[Man01] Giovanni Manzini. An analysis of the Burrows–Wheeler transform. *Journal of the ACM*, 48(3):407–430, May 2001.

[Mat94] Jirka Matousek. Geometric range searching. *Computing Surveys*, 26(4):422–461, 1994.

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

[MK08] Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Proceedings of Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 5171, pages 165–178, Boston, Massachusetts, USA, 2008. Lecture Notes in Computer Science.

[MNP+04] David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and A. Y. Wu. A computational framework for incremental motion. In Jack Snoeyink and Jean-Daniel Boissonnat, editors, *Proceedings of the 20th ACM Symposium on Computational Geometry*, pages 200–209, Brooklyn, New York, USA, 2004. ACM.

[Mon05] J. J. Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68:1703–1759, 2005.

[NM07] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):2, 2007.

[NS08] Sotiris Nikoletseas and Paul G. Spirakis. Efficient sensor network design for continuous monitoring of moving objects. *Theoretical Computer Science*, 402(1):56–66, 2008.

[PAAV03] Octavian Procopiuc, Pankaj K. Agarwal, Lars Arge, and Jeffrey Scott Vitter. A dynamic scalable kd-tree. In *Proceedings of the International Symposium on Spatial and Temporal Databases*, volume LNCS 2750, 2003.

[Ple80] J. Plesnik. On the computational complexity of centers locating in a graph. *Applications of Mathematics*, 25(6):445–452, 1980.

[Ple87] J. Plesnik. A heuristic for the p-center problem in graphs. *Discrete Applied Mathematics*, 17(3):263–268, 1987.

[POR] PORTAL. Portland oregon regional transportation archive listing. `http://portal.its.pdx.edu/`.

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

[Ris76] Jorma Rissanen. Generalized Kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20(3):198–203, May 1976.

[RL87] Peter J. Rousseeuw and Annick M. Leroy. *Robust Regression and Outlier Detection*. New York: Wiley, 1987.

[Rob81] John T. Robinson. The K-D-B-Tree: A search structure for large multidimensional dynamic indexes. In Y. Edmund Lien, editor, *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pages 10–18, Ann Arbor, Michigan, 1981. ACM Press.

[Rou84] Peter J. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, 79(388):871–880, 1984.

[SA95] Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, 1995.

[Sam84] Hanan Samet. The quadtree and related hierarchical data structures. *Computing Surveys*, 16(2):187–260, June 1984.

[SG06] Kunihiko Sadakane and Roberto Grossi. Squeezing succinct data structures into entropy bounds. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1230–1239, Miami, Florida, USA, 2006. ACM Press.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.

[Sha94] Micha Sharir. Almost tight upper bounds for lower envelops in higher dimensions. *Discrete and Computational Geometry*, 12(1):327–345, Dec 1994.

[SM06] Christopher M. Sadler and Margaret Martonosi. Data compression algorithms for energy-constrained devices in delay tolerant networks. In Andrew T. Campbell, Philippe Bonnet, and John S. Heidemann, editors, *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, pages 265–278, Boulder, Colorado, USA, November 2006. ACM.

[SS07] Nicolas Saunier and Tarek Sayed. Automated analysis of road safety with video data. *Transportation Research Record: Journal of the Transportation Research Board*, 2019:57–64, 2007.

[ST83] Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[ST95] Elmar Schomer and Christian Thiel. Efficient collision detection for moving polyhedra. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, pages 51–60, Vancouver, B.C., Canada, 1995. ACM Press.

[ST96] Elmar Schomer and Christian Thiel. Subquadratic algorithms for the general collision detection problem. In *Abstracts 12th European Workshop on Computational Geometry*, pages 95–101, Münster, Germany, 1996.

[SWP08] Emad Soroush, Kui Wu, and Jian Pei. Fast and quality-guaranteed data streaming in resource-constrained sensor networks. In Xiaohua Jia, Ness B. Shroff, and Peng-Jun Wan, editors, *Proceedings of the Ninth ACM International Symposium on Mobile ad hoc networking and computing*, pages 391–400, Hong Kong, China, 2008. ACM.

[Tel] TeleAtlas. Dynamic content - real time traffic information. `http://www.teleatlas.com/OurProducts/MapEnhancementProducts/DynamicContent/index.htm`.

[THH00] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62:1805–1824, 2000.

[Tre10] Martin Treiber. Dynamic traffic simulation. `http://www.traffic-simulation.de/`, Jan 2010.

[Wan98] Demin Wang. Unsupervised video segmentation based on watersheds and temporal tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(5):539–546, 1998.

[WER07] Zhimin Wang, Sebastian Elbaum, and David S. Rosenblum. Automated generation of context-aware tests. In *Proceedings of the 29th International Conference on Software Engineering*, pages 406–415, Minneapolis, MN, USA, 2007. IEEE Computer Society.

[WILW07] Christopher R. Wren, Yuri A. Ivanov, Darren Leigh, and Jonathan Westbues. The MERL motion detector dataset: 2007 workshop on massive datasets. Technical Report TR2007-069, Mitsubishi Electric Research Laboratories, Cambridge, MA, USA, August 2007.

[Wol02] Ouri Wolfson. Moving objects information management: The database challenge (vision paper). In Alon Y. Halevy and Avigdor Gal, editors, *Proceedings of the 5th International Workshop on Next Generation Information Technologies and Systems*, pages 75–89, Ceasarea, Israel, 2002. Lecture Notes in Computer Science.

[WZ94] Aaron D. Wyner and Jacob Ziv. The sliding-window Lempel-Ziv algorithm is asymptotically optimal. *Proceedings of the IEEE*, 82(6):872–877, Jun 1994.

[YZ09] Ke Yi and Qin Zhang. Multi-dimensional online tracking. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1098–1107, New York, NY, USA, 2009. SIAM.

[ZL77] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–342, May 1977.

[ZL78] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

Index