Machine Learning and Coresets for Automated Real-Time Data Segmentation and Summarization

by

Mikhail Volkov
M.Sc., Massachusetts Institute of Technology (2013),
B.A., B.A.I., University of Dublin, Trinity College (2010)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016

Author ..... Signature redacted .....
Department of Electrical Engineering and Computer Science
August 31, 2016

Certified by.. Signature redacted .....
Daniela Rus
Andrew and Erna Viterbi Professor, EECS, MIT
Thesis Supervisor

Accepted by... Signature redacted .....
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Students

Machine Learning and Coresets for Automated Real-Time Data Segmentation and Summarization

by

Mikhail Volkov

M.Sc., Massachusetts Institute of Technology (2013),

B.A., B.A.I., University of Dublin, Trinity College (2010)

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

In this thesis, we develop a family of real-time data reduction algorithms for large data streams, by computing a compact and meaningful representation of the data called a coreset. This representation can then be used to enable efficient analysis such as segmentation, summarization, classification, and prediction. Our proposed algorithms support large streams and datasets that are too large to store in memory, allow easy parallelization, and generalize to different data types and analyses.

We discuss some of the challenges that arise when dealing with real Big Data systems. Such systems are designed to routinely process unseen, possibly unbounded, data streams; are expected to perform reliably, online, in real-time, in the presence of noise, and under many performance and bandwidth limitations; and are required to produce results that are provably close to optimal. We will motivate the need for new data reduction techniques, in the form of theoretical and practical open problems in computer science, robotics, and medicine, and show how coresets can help to overcome these challenges and enable us to build several practical systems that meet these specifications.

We propose a theoretical framework for constructing several coreset algorithms that efficiently compress the data while preserving its semantic content. We provide an efficient construction of our algorithms and present several systems that are capable of handling unbounded, real-time data streams, and are easily scalable and parallelizable. Finally, we demonstrate the performance of our systems with numerous experimental results on a variety of data sources, from financial price data to laparoscopic surgery video.

Thesis Supervisor: Daniela Rus
Title: Andrew and Erna Viterbi Professor, EECS, MIT

Reader 1: Polina Golland
Title: Professor, MIT

Reader 2: Dan Feldman
Title: Professor, University of Haifa

Acknowledgments

Deus é fiel

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

- AWS: Amazon Web Services
- BOW: bags of words
- CNN: convolutional neural network
- DCT: discrete cosine transform
- DR: dead reckoning
- DTW: dynamic time warping
- EC2: Elastic Compute Cloud
- FAB-MAP: Fast Appearance-based Mapping algorithm
- FIFO: first in, first out
- FPS: farthest point sampling
- GPS: global positioning system
- HMM: Hidden Markov Model
- HOG: histogram of oriented gradients
- HSV: hue, saturation, value
- KL: KullbackLeibler
- LDA: latent Dirichlet allocation

- LSA: latent semantic analysis
- LSG: laparoscopic sleeve gastrectomy
- NNMF: non-negative matrix factorization
- PCA: principal component analysis
- PDF: probability density function
- RDP: Ramer-Douglas-Peucker
- RGB: red, green, blue
- RMIS: robot-assisted minimally invasive surgery
- SFTA: Segmentation-based Fractal Texture Analysis
- SURF: Speeded Up Robust Features
- SVD: singular value decomposition
- SVM: Support Vector Machine
- WM: working memory

Chapter 1

Introduction

"Where is the Life we have lost in living?
Where is the wisdom we have lost in knowledge?
Where is the knowledge we have lost in the information?"
— T. S. Eliot, *Choruses from The Rock* (1934)

1.1 Prelude

Our world is changing. Around the year 380 BC, Plato argued that a man should not begin his study of philosophy until the age of thirty. His reasoning was that he is not ready to think critically before such time, the implication being that he will still have more than enough time to learn everything there is to know about the world after such time. Following the Dark Ages, the Doctor of Philosophy title was established in European universities since the twelfth century AD, meant to be a "terminal degree" to confirm the completion of one's knowledge of the world. Our understanding of the world progressed and the invention of the printing press heralded the birth of modern education. Still, even as late as the seventeenth century, the father of modern Western philosophy, René Descartes – a philosopher, mathematician, and scientist by today's academic canon – could be considered to have known all there was to know about the world during his time. To use MIT's own metaphor, knowledge and the information contained therein, was still just a water fountain and one could drink most of it in

the seventeenth century if one was thirsty enough.

As information was passed down through generations, living standards improved, education became the norm among young adults, and human knowledge continued to increase. As we grew to understand more about the laws of physics and the natural world, more academic disciplines were established. New knowledge was being derived from existing knowledge, leading to increasingly abstract academic pursuits and causing a second order increase in the rate of accumulation of knowledge. And although life expectancy also increased, we could not keep up with exponential increase of information. And so it was that the human race inevitably crossed an "epistemological event horizon", a point in time whereupon it was no longer possible for one man to know everything that was known before him. The water fountain burst its pipes, turning into the proverbial fire hydrant, and the world could no longer be conquered by one man.

1.1.1 What Is Data?

"Distinguishing the signal from the noise requires both scientific knowledge and self-knowledge: the serenity to accept the things we cannot predict, the courage to predict the things we can, and the wisdom to know the difference."

— Nate Silver, *The Signal and the Noise* (2012) [122]

There are a many elementary and uncontroversial sources that will answer this question quite concisely, and it is not my intention to engage the reader in a différance of scientific terminology. But I would simply like to draw one's attention to the large number of different academic and practical problem domains for which the term is always reliably well-defined, and always slightly nuanced.

Our scientific evolution as a species, from Plato to Popper, has heralded moments of tremendous historical significance. And over the past 20 years, the advent of the Internet has without a doubt been the most significant to date. However we choose to define data, the Internet has fundamentally changed the way we produce, consume, and interact with it.

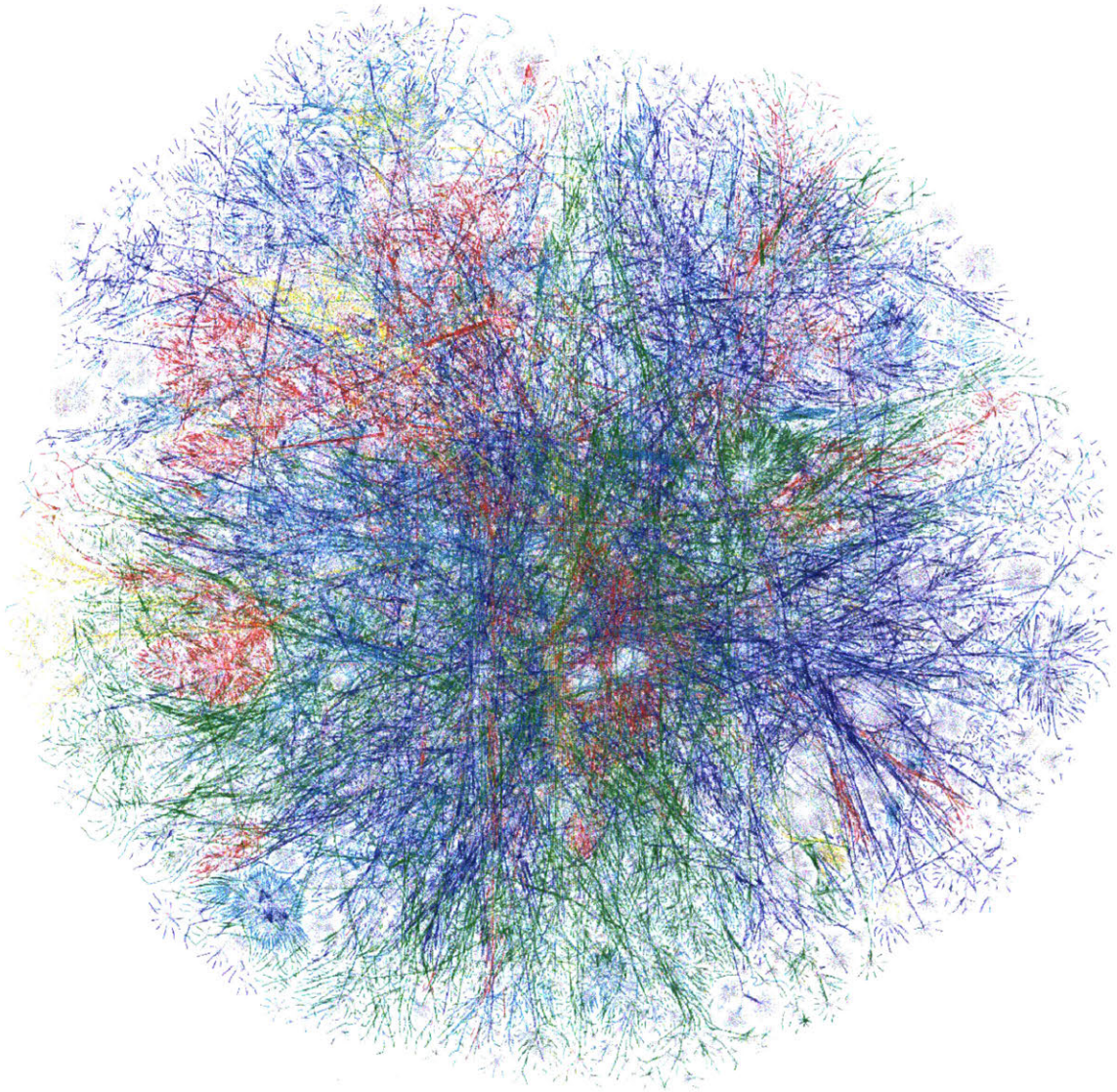
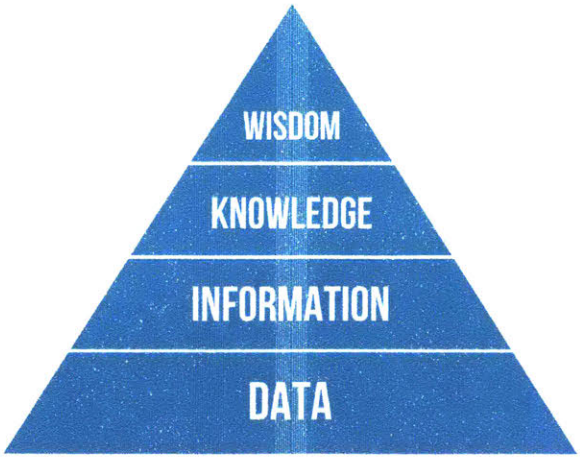


Figure 1-1: The Internet



The Data-Information-Knowledge-Wisdom Pyramid [137].

Figure 1-2: DIKW Pyramid

In terms of its impact on society, the closest historical parallel that we can draw would probably be the invention of the printing press. And with this comparison I would offer the reader an opinion that is at the heart of how I see the central theme of my of my work—I believe that there is a growing dichotomy between *information* and *data*, and that at the time that the printing press was invented in the fifteenth century these ideas were synonymous—all data was once information.

Today we live in a different time. To understand what data really entails today, and how it is different from knowledge and information, I believe that it is helpful to consider just how much data is there.

1.1.2 How Much Data Is There?

"There was 5 exabytes of information created between the dawn of civilization through 2003, but that much information is now created every 2 days, and the pace is increasing."

— Eric Schmidt, *Techonomy Conference* (2010)

Any discussion about data inevitably leads to the question of just how much data is out there? The study that Eric Schmidt cited [92] attempted to add up all the data stored since the dawn of civilization until 2003. And the number they came up

At 7.22 billion, the number of mobile phones has surpassed our own population.

Figure 1-3: Protests in Hong Kong, 2014

with is around 5 exabytes. To put that into some kind of context: 1 gigabyte is a small library of approximately 1000 books; 5 exabytes is 5 billion gigabytes, which is approximately one such small library for every single person on the planet. To generate this volume of data every 2 days is a fairly mind-blowing statistic, and it pushes Big Data to the boundaries of our intuition, where we can still understand such numbers on paper, but probably find it difficult to relate them to reality.

And indeed, according to Eric Schmidt, this pace is only increasing. At 7.22 billion, the number of active mobile phones has recently surpassed the number of humans alive today [128] (Fig. 1-3). Almost every song, movie and piece of multimedia is available instantly, anywhere in the first first world. Online education is now abundant and free, and social media is ubiquitous. More weather alerts, more traffic information, more social media feeds, and more news reports – ostensibly, more data is a good thing. Almost any factual claim can be instantly checked, verified, and cross-referenced against an archive of relevant information, making fact-based disagreements a thing

of the past. Expanding on Eric Schmidt's prediction, Bruce Schneier writes, *"All of our data exhaust adds up ... By 2015, 76 exabytes of data will travel across the Internet every year"* [118].

But just as as the Industrial Revolution saw an unprecedented rise in the levels of pollution, so is the Information Revolution accompanied by a rise in data exhaust that has not yet given us the the benefit of hindsight. "Disruptive technologies" are taking over Silicon Valley just like the coal mines and steel plants once redecorated our skies and landscapes. The Data Rush is as real as the Gold Rush. Data is no longer just information—it is a raw material and a commodity, it is mined, generated, processed, bought, sold, and stolen.

Hedge funds are becoming powerhouses of top engineering talent and finding alphas through sentiment analysis of Twitter feeds; ultra-high-frequency proprietary trading firms are literally manipulating the laws of physics to get their data faster than you; and some of the biggest tech companies that you have never heard of are data brokers.

The rate at which data is being produced is increasing exponentially. It is becoming increasingly difficult to distinguish the signal from the noise, and to separate data from data exhaust.

1.1.3 Why Do We Need Data Reduction?

"One can prove that the better you can compress, the better you can predict; and being able to predict well is key for being able to act well."

— Marcus Hutter, *Human Knowledge Compression Contest* (2009) [72]

Big Data is a term for data sets that are so large and complex that traditional techniques are inadequate to process the data. This complexity is characterized not only by an increase in the volume of data, but also the speed of data acquisition, varying quality and consistency of data sources. This thesis introduces new theory, algorithms and systems for data reduction of real Big Data problems.

We motivate the central contribution of this work with a real example from the

The Kepler space observatory, used to locate likely planets orbiting stars beyond the sun (artist's rendition).

Figure 1-4: The Kepler space observatory

cutting edge of modern technology. The Kepler space observatory was launched in 2009 to discover Earth-size planets orbiting other stars [12] (Fig. 1-4). The spacecraft is equipped with a telescope and the largest camera system ever launched into space, capable of recording 30 minutes of footage at a 95 megapixel resolution [102]. Unfortunately, the resulting sum of all 95 million pixels constitute more data than can be stored and sent back to Earth.

The problem that Kepler faced was that despite recording a high amount of data, only some fixed fraction of it contained any useful information. The solution that NASA employed was to compress the data to about 6 percent of the raw pixels, selecting only those pixels associated with specific targets of interest. The data from these pixels is then re-quantized, compressed, and stored along with other auxiliary data to be later sent back to Earth [100, 101].

What we are really interested in is not the data itself, but the information contained in it that enables us to do something useful. This intuition was formalized by Claude Shannon with the birth of information theory. Shannon's entropy measures the information contained in a stream of data, and puts a numerical bound on the theoretical performance of any form of lossless compression [119].

There is a close equivalence between compression and machine learning. From the machine learning point of view, a system that predicts posterior probabilities of a sequence given its entire history can be used for optimal data compression; while an optimal lossless compression algorithm can be used for classification and prediction (by finding the symbol in the sequence that yields the best compression ratio, given the previous history [120, 15].

Indeed, the problem of compression is so general that it is considered to be equivalent to general AI. This is exemplified well in the field of decision theory, which formally poses the problem of a rational agent interacting with the world in uncertain conditions. In 2000, Marcus Hutter proved that finding the optimal behavior of a rational agent is equivalent to compressing its observations [70, 71]. This result essentially proved Occam's Razor, the idea that the simplest satisfactory explanation is usually the correct one [96]. For a mode in-depth discussion on the subject see [96], and the references therein.

1.1.4 Why Do We Need Coresets?

"Science may be described as the art of systematic over-simplification – the art of discerning what we may with advantage omit."

— Karl Popper, *The Open Universe: An Argument for Indeterminism* (1992) [110]

In the Kepler example, the data reduction techniques were traditional lossy compression or sub-sampling of the original data. Another example of lossy compression is the MP3 standard for digital audio files. The degree of data reduction is expressed as a fraction of the size of the compressed data relative to the original data. Both are examples of lossy compression because some of the information contained in the

data is irreversibly destroyed in the process.

However, in both examples one property remains preserved – in data compression, the aim is to reduce size of the data while in some sense preserving the original character properties of the data; the compressed data is still the same type of data structure as the original data. An image of deep space is recovered although some of the pixels are missing; the mp3 track can still be played back, albeit at a lower sound quality than the original.

But what if this key requirement is relaxed, so that we no longer require an approximation of the original data, but merely that we can approximate some indirect property with respect to the original data? As it turns out, this relaxation is enough to enable a whole family of data reduction constructs called coresets that allow us to solve a wide range of problems that are intractable to solve on the entire dataset and do not become easier with a simple compression of the data. In this context, the best definition of a coreset is a *problem-dependent* compression of the data. The key difference between coresets and compression is the additional level of indirection in the problem that we are trying to solve – compression aims to approximate the data itself, while coresets aim to approximate some derived property of the data.

With coresets we may indeed "with advantage omit" a lot of redundant data, allowing us to efficiently solve Big Data problems on small but provably discerned subsets. If indeed the rationale for data reduction being representative of general AI is true, or if at least it is a sound benchmark, then it suggests that there is great power in the simplicity of representation and parsimony of expressiveness that coresets provide. It is my aim to convince the reader that this is the case in the rest of this thesis.

1.1.5 The Big Picture

"We demand rigidly defined areas of doubt and uncertainty!"

Douglas Adams, *The Hitchhiker's Guide to the Galaxy* (1979)

The Hutter Prize for Lossless Compression of Human Knowledge was estab-

lished to promote the advancement of AI through a contest to compress the English Wikipedia [72]. Since Wikipedia is very sparse, it makes for an excellent test case for compression algorithms. In this thesis we set ourselves a similar challenge, though with distinctly different motivations and technical approach.

However, it is no coincidence that we also chose to use Wikipedia. It is perhaps the most influential and ubiquitous term-document corpus in the Age of Information. Much like *The Hitchhiker's Guide to the Galaxy* [3] was designed to serve as "the standard repository for all knowledge and wisdom", Wikipedia may be the first reasonable approximation to the sum total of all human knowledge. It is profound to consider that so much data, information, and knowledge can be contained in just under 12GB of data, and stored on a microSD card the size of a fingernail.

1.2 Thesis Overview

In this thesis, we develop a family of real-time data reduction algorithms for large streams, by computing a compact and meaningful representation of the data called a *coreset*. This representation can then be used to enable efficient analysis such as segmentation, summarization, state estimation, and prediction. Our proposed algorithms support large streams and datasets that are too large to store in memory, allow easy parallelization, and generalize to different data types and analyses.

In this section we will discuss some of the challenges that arise when dealing with real Big Data systems. Such systems are designed to routinely process unseen, possibly unbounded data streams and are expected to perform reliably, online, in real-time, in the presence of noise and under many performance and bandwidth limitations, while still producing results that are close to optimal. We will motivate the need for data reduction, in form of both theoretical and practical open problems in computer science, robotics, and medicine, and show how coresets can help overcome all of these challenges and enable us to build a number of practical systems in these areas.

1.2.1 Motivation and Open Problems

There is an increasing demand for systems that process long-term, high-dimensional data streams over practically unbounded time. Examples include life-logging video streams, financial ticker data, and Twitter feeds. In these examples the data is represented as a high-dimensional feature at discrete points in time – for example location vectors, stock prices, or image content feature histograms. In other examples the data may have no inherent temporal structure – for example a collection of documents written in the English language.

Segmentation and Summarization Analyzing many of these data streams, we often observe a partition into time intervals that are governed by different temporal models. Detecting these segments and the models describing each of them is known as temporal segmentation. Such a segmentation can be used to find important transition points in the data that are semantically informative with respect to our choice of representation. The segmentation itself may be the ultimate goal of the system, such as automatic phase detection in laparoscopic surgical video. In other cases the transition points can enable further analysis of the data such as efficient summarization, state estimation and retrieval algorithms that would otherwise be impossible to run on the full dataset. For example, the summarization of wearable video data can be used to efficiently detect different scenes and important events, while collecting GPS data for citywide drivers can be used to learn weekly transportation patterns and characterize driver behavior.

Currently, segmentation of such data streams over long periods of time is limited. One of the reasons for this is that current optimally-guaranteed algorithms for data segmentation and summarization can handle only relatively small datasets that consist of only few segments, and low-dimensional signals such as GPS data. Larger data streams are usually handled by running computation only on small parts of the streams, or resorting to a non-optimal on-line approach for summarization [130].

State Estimation and Retrieval In continuously operating robotic systems, where life-long video streams are a crucial source of information, efficient representation of the captured video is crucial. Many uses of visual sensors involve retrieval tasks on the collected video with various contexts. Robots operating in a persistent manner are faced with several difficulties due to the variety of tasks and scenes, and the complexity of the data arriving from the sensors. While for some tasks we can define simplified world representations that form approximate sufficient statistics, real-world scenarios challenge these approximations.

For example, one important task is loop closure detection for robotic navigation and localization. Loop closure involves the recognition of mutually-visible parts of the world from two or more frames, allowing correction of reconstruction and localization errors due to noisy measurements, as well as initialization and recalibration of pose estimation processes.

Scene assumptions are also critical. Under the assumption of a static environment, a location-based loop closure can be defined in terms of locations and their visual appearance model. Clearly such a model would not be sufficient for a human operator trying to find a specific observed event in the robots history. Breaking the assumption of a static environment would turn the relatively easily defined location-based mapping problem into the full life-long localization and reconstruction problems that are still open lines of research. On the other hand, keeping the full visual history of the robot in random access memory is also suboptimal and often is impossible. A more condensed form, summarizing the video and allowing various tasks to be performed is an important tool for persistent operation.

More generally, various retrieval tasks can be defined directly on the visual history, searching for places, objects, people, and other queries. For example, an interactive retrieval task might involve simplifying the users search in a large video for a specific subsequence. While for specific tasks and contexts, we can describe specific representations that gather sufficient statistics from a life-long video, for a more complete set of tasks, going back to the visual history may be required, at least in some condensed form of it.

Classification and Prediction Video segmentation and summarization systems are also important in the medical field. Video-based coaching and debriefing of laparoscopic and robot-assisted surgery has been demonstrated to contribute to enhanced surgical performance. A context-aware summarization and retrieval system for surgical video can facilitate education, time-critical consultation, and can improve perioperative workflow efficiency of operating room assignment and turnover.

While recording laparoscopic and robotic surgical procedures is quick and easy, reviewing and analyzing video remains a time-consuming process. Videos are not segmented automatically and must be viewed in their entirety or by skipping time segments in trial-and-error fashion to identify different phases of an operation. Modern pressures on training and productivity preclude spending hours viewing and editing surgical video for the purpose of routine video-based coaching. Moreover, hospital policies and legacy infrastructure are often prohibitive of recording and storing large amounts of video data as a matter of routine.

These procedures are typically carried out in a stepwise fashion by identifying distinct steps or phases or an operation. Therefore a visual segmentation and summarization of the surgical video is an obvious solution towards an automatic system for identification of the surgical phases. However, the challenge lies not only in computing a segmentation but in being able to identify and predict the correct segments, based on the surgeons' own knowledge of the criteria for identifying the segments manually.

This is a typical problem of classification in machine learning. Training a system to recognize and correctly classify visual categories such as surgical phases, as well as a multitude of other medical imaging problems has been the subject of research in computational biology and medical engineering for decades. The biggest challenge in building reliable machine learning systems is understanding the data and finding the right feature space. A visual segmentation and summarization can serve as a pre-processing step for producing high-level semantic features that are then used to train the phase classification and prediction system.

This is as much a problem of human learning as it is machine learning, as the

surgeons need to explicitly elaborate on how they arrive at critical decisions during the operation – this is not always an easy task, as a lot of it comes down to years of experience, and can be hard to explain.

Dimensionality Reduction Finally, we consider the case of non-temporal or *stationary* data that has no inherent dimension of time. Much of the large scale stationary data sets available today (such as image corpora, collections of documents, etc.) are sparse. As motivated in the previous section, Wikipedia is perhaps the most important sparse dataset that is universally available to anyone with an internet connection. It is arguably the best snapshot of all the world's knowledge, and we are spoiled with no shortage of open research questions.

For example, in machine learning and natural language processing, a topic model is a type of statistical model for discovering the abstract "topics" that occur in a collection of documents. This is a canonical dimensionality reduction problem in information retrieval. Applying this definition to Wikipedia it is tempting to consider if we can compute say 100 such abstract topics. Essentially, we are asking the question "can we represent the sum total of human knowledge with 100 high-dimensional vectors, and what would that mean?"

The English Wikipedia consists of approximately 4.4 million articles, and every article can be considered simply as a collection of words from the English language. We can associate a matrix with the entire English Wikipedia, where the (approximately 1.4 million) English words define the columns and the individual documents define the rows. This matrix will be very large and very sparse because most English words do not appear in most documents.

A topic model of a sparse matrix immediately implies sparse SVD. Even if we could compute the exact SVD of a large document-term matrix such as Wikipedia, each of the corresponding k topics will be a linear combination of all the 1.4 million distinct terms in Wikipedia. These combinations may be too large to fit into RAM and will be difficult to interpret. In-fact, the matrix is so large that no existing dimensionality reduction algorithm can compute its eigenvectors. To this point, running the state

of the art singular vector decomposition (SVD) implementation on the Wikipedia document-term matrix crashes the computer very quickly after applying its step of random projection on the first few thousand documents. This is because such dense vectors, each of length 1.4 million, use all of the computer's RAM capacity.

Algorithms for dimensionality reduction usually aim to project an input set of d-dimensional vectors (documents) onto a $k \leq d-1$ dimensional affine subspace that minimizes the sum of squared distances to these vectors, under some constraints. Special cases include linear regression ($k = d-1$), low-rank approximation (k-SVD), latent semantic analysis (LSA), principle component analysis (PCA), latent Dirichlet allocation (LDA), and non-negative matrix factorization (NNMF). Learning algorithms such as k-means clustering can then be applied on the low-dimensional data to obtain fast approximations. There are no algorithms or coreset constructions with performance guarantees for computing the PCA of sparse $n \times n$ matrices in the streaming model, meaning that they use memory that is poly-logarithmic in n.

1.2.2 Main Results

In this thesis we present theory and develop systems for analyzing very large datasets by compressing the data into a compact and meaningful representation called a *coreset* [4, 46]. This representation can then be used to enable efficient computation of inefficient algorithms. Our proposed algorithms support streams and datasets that are too large to store in memory, allow "embarrassingly parallel" implementations, and are data-agnostic in the sense that they generalize to different types of data and analytical methods.

Informally, coresets are approximation algorithms for specific problems that offer linear construction complexity, and sublinear memory requirements. Running an algorithm for a specific type of coreset (such as segmentation or clustering) approximates the cost obtained by that algorithm on the original data, with a provably small error.

More rigorously, a coreset C is a problem dependent compression of the data A, such that running an algorithm f on the coreset yields a result $f(C)$ that provably

approximates the result $f(A)$ of running the algorithm on the original data. If the coreset is small and its construction is fast, then computing $f(C)$ is fast even if computing $f(A)$ on the original data is intractable.

In the following chapters of the thesis we detail how we use coresets to enable efficient solutions to the problems of segmentation, summarization, state estimation, retrieval, classification, and prediction. This thesis is organized into ten chapters, as follows.

Segmentation In Chapter 4 we consider the problem of computing an optimal segmentation of a signal by a k-piecewise linear function, using only one pass over the data. We present a coreset computation for the signal that enables an efficient approximation to the k-segment mean problem. Our results rely on a novel reduction of statistical estimations to problems in computational geometry.

Specifically, we show that the segmentation problem admits coresets of cardinality only linear in the number of segments k and data points n and independent of dimension d of the signal. Further, we provide a construction for such a representation of size $O(k/\varepsilon^2)$ that provides a $(1+\varepsilon)$-approximation for the sum of squared distances to any given k-piecewise linear function. Our coreset construction can be streamed or parallelized.

In Chapter 5 we present a system for efficient online segmentation of large data streams, such as real-time video streams. Our system implements the algorithms presented in the previous chapter. We empirically evaluate our algorithms on very large synthetic and real datasets including GPS data, financial ticker data, and real-time video feeds from mobile phones and wearable devices. Finally, we show how our system can be scaled up by parallelizing our experiments across 255 machines on Amazon EC2.

Summarization, State Estimation, and Retrieval In Chapter 6 we consider the segmentation and summarization of life-long video streams in continuously operating robotic systems. Using the coresets developed in the previous chapters, we

formulate a new method for hierarchical retrieval of frames from large video streams that are collected online by a moving robot. We demonstrate how to utilize the resulting structure for efficient loop closure by a novel sampling approach that is adaptive to the structure of the video. The same structure also allows us to create a highly-effective search tool for large-scale videos.

We define a data-adaptive structure called a *coreset tree* that consists of a summary of representative *keyframes* from the video. We show how the coreset tree can be used as a tool to enable efficient localization by providing useful candidate keyframes for loop closure algorithms. We present an algorithm for sampling the coreset tree to find such candidate keyframes over a video history of arbitrarily large size, while providing guarantees for the retrieval given enough computation time. The collection of the data requires a linear-size memory, and allows logarithmic-time retrieval, providing an approximate sufficient statistic for a large variety of tasks involving the visual history. The tasks we demonstrate in this chapter can be considered representatives of the full range of possible tasks for a robotic system involving its visual history.

We present a user interface that allows intuitive human retrieval of keyframes from a coreset summary of a processed or online video stream. Finally, we demonstrate the efficiency and versatility of the coreset tree by conducting a variety of summarization, retrieval and localization experiments on standard datasets, on a large-scale video of a city tour from a wearable camera, and on remote-controlled quad-rotor robots mounted with a wireless camera.

Classification and Prediction In Chapter 7 we consider the problem of context-aware segmentation of laparoscopic and robot assisted surgical video. As motivated in the previous section, such a system would be invaluable to surgeons, to potentially improve performance and workflow efficiency, and as a tool for education and time-critical expert consultation. The stakes for developing such a system could not be higher – when a human life is one of the variables that depends on the correct operation of a system, the margin for error is close to zero.

In this part of the thesis we present a system that automatically generates a

video segmentation of laparoscopic and robot-assisted procedures according to their underlying surgical phases. We develop a visual feature space, tailor-made visual for the laparoscopic surgery domain, that captures the intuition of the world's best laparoscopic surgeons. We use the k-segment coreset algorithms presented in Chapter 4 as a backbone for our system, and we use the coreset tree presented in Chapter 6 as a preprocessing step, allowing our system to produce results of approximately equal quality, in real-time, and using only a fraction of the computational resources. Finally, we employ an SVM and HMM combination as the top layer of our system enabling the system to automatically learn the phase transitions from training data and segment unseen the surgical videos according to their phases. We evaluate our system with cross-validation experiments on real video of laparoscopic vertical sleeve gastrectomy (LSG) procedures, and propose a blueprint for piloting such a system in a real operating room environment with minimal risk factors.

Dimensionality Reduction In Chapter 8 we consider a long-open research question of whether we can compute a coreset for principal component analysis (PCA) that is both small in size and a subset of the original data. In this chapter we answer this question affirmatively and provide an efficient construction.

Our main result is the first algorithm for computing a coreset of size independent of both n and d, for any given $n \times d$ input matrix. The algorithm takes as input a finite set of d-dimensional vectors and computes a coreset that is a weighted subset of k^2/ε^2 such vectors. This coreset can be used to approximate the sum of squared distances from a matrix whose rows are the n vectors seen so far, to any k-dimensional affine subspace, up to a factor of $1 \pm \varepsilon$. For a (possibly unbounded) stream of such input vectors the coreset can be maintained at the cost of an additional factor of $\log^2 n$.

The polynomial dependency on d of the cardinality of previous coresets made them impractical for fat or square input matrices, such as the Wikipedia document-term matrix, images in a sparse feature space representation, or an adjacency matrix of a graph. If each row of an input matrix has $O(\text{nnz})$ non-zero entries, then the

update time per insertion, the overall memory that is used by our algorithm, and the low-rank approximation of our coreset is independent of n and d.

In Chapter 9, we provide an efficient implementation our coreset for dimensionality reduction presented in the previous chapter. Our system can run on a standard laptop. Since our algorithm affords an embarrassingly parallel architecture, we implement our system on Amazon EC2, and receive a significantly better running time and accuracy compared to existing heuristics that yield non-sparse solutions. We evaluate our system on synthetic data to provide a ground-truth for the quality, efficiency, and scalability of our system. Finally, as motivated in the previous section, we consider the grand challenge of computing a topic model for the entire English Wikipedia. We apply our algorithm to compute the principal component analysis (PCA) of the Wikipedia document-term matrix, and use this to compute a topic model of $k = 100$ topics.

1.2.3 Key Contributions

This thesis makes the following contributions to the state of the art.

- A coreset for the k-segment mean problem, of size $O(k/\varepsilon^2)$ that provides a $(1+\varepsilon)$-approximation for the sum of squared distances to any given k-piecewise linear function (Chapter 4).

- A system for approximating the k-segmentation of streaming data. Experimental results with video streams, GPS data, and financial ticker data (Chapter 5).

- Algorithms for semantic summarization and retrieval of video frames from unbounded life-long video streams (Chapter 6).

- A new mechanism for computing an adaptive, semantic summary of the video by means of a coreset tree (Chapter 6).

- A system for efficient loop closure detection by novel sampling approach that

is adaptive to the structure of the video. Experimental results with real video data of a tour of Boston collected from a wearable device (Chapter 6).

- A system for automatically identifying the phases of laparoscopic and robot-assisted surgical procedures and segmenting them in real-time. Experimental results with real surgical data. (Chapter 7).

- A dimensionality reduction algorithm for computing a (k, ε)-coreset of size independent of both n and d, for any given $n \times d$ input matrix (Chapter 8).

- A system for computing an efficient low-rank approximation of a matrix. Experiments to to compute the principal component analysis (PCA) and derive a topic model for the entire English Wikipedia (Chapter 9).

Chapter 2

Notation

This thesis uses the following notation and conventions. Where the nature of a term or variable is not obvious from the context, it is always made clear explicitly.

- $\mathbb{N}, \mathbb{Z}, \mathbb{R}$ denote the set of natural numbers, the set of integers, and the real numbers, respectively. A superscripted $\mathbb{R}^d$ denotes the real coordinate space of d dimensions.
- i, j, k usually denote integers, such as indices of a closed form expression or iterations in an algorithm.
- u, v, w usually denote vectors, and subscripted letters denote their elements, i.e. $u = [u_1, u_2, \ldots]$.
- The notation $[n]$ denotes the vector of integers $[1, \ldots, n]$.
- The notation $j{:}k$ denotes the set of indices $\{j, j+1, \ldots, k\}$ for integers $k \geq j$.
- X, or any other italicized letter denotes a matrix. (Note that in this thesis we use the convention that vectors are the rows of a matrix, not its columns. Although this is unconventional with respect to most mathematical literature, it is much more intuitive for our work and it is the convention that was used in all our publications.)
- The notation $X_{i,:} \in \mathbb{R}^{1 \times d}$ denotes the i-th row of a matrix X.

- The notation $X_{:,j} \in \mathbb{R}^{n \times 1}$ denotes the j-th column of a matrix X.
- n denotes the number of rows of a matrix.
- d denotes the dimension of the matrix X, that is the number of its columns and the number of elements of its row vectors.
- t denotes the time dimension, which is a non-negative integer.
- $\text{proj}_v u$ is the projection of u on v.
- $\text{comp}_v u$ is the component of u in the direction of v.
- trace denotes the trace of a matrix.
- $\Pr(\cdot)$ denotes the probability of an event.
- mean denotes the mean or expected value of a set of numbers.
- var denotes the variance of a set of numbers.
- A string of non-italicized letters such as KSEGMENTCORESET denotes the name of an algorithm.
- $O(\cdot)$ denotes the asymptotic function used to express the upper bound on the growth rate of the function (big O notation).
- A denotes a non-temporal input data matrix.
- P denotes a temporal signal, which is defined as a set vectors $P \in \mathbb{R}^d + 1, P = \{(1, p_1), \ldots, (n, p_n)\}$, where $p_j \in \mathbb{R}^d$ is the point at time index j for every $j \in [n]$.
- ℓ (`\ell`) denotes a vector norm. In particular, ℓ_2 denotes the Euclidean norm.
- $\|X\|^2 = \sum_{ij} (X_{ij})^2$ is the sum of squared entries of a matrix or a vector X (the Frobenius norm for a matrix or the ℓ_2 Euclidean norm for a vector). We emphasize this as $\|X\|_F^2$ where necessary, to differentiate between the induced 2-norm $\|X\|_2^2$ of a matrix.

- C, D denote coresets. C usually denotes a simple coreset that is a subset of a rows of a matrix or signal. D usually denotes a compound coreset construction such as a tuple of mathematical objects.

- U, D, V usually denote the singular value decomposition (SVD) of a matrix $A = UDV^T$. (D is used over Σ to avoid confusion with summation, but Σ is used where it is clearer to avoid confusion with the coreset D. This is always defined explicitly.)

- k denotes the number of segments in the k-segment mean problem or the rank of the subspace in the low rank approximation (k-SVD) problem.

- ε denotes a positive real number that represents an error measure, specifically in the definition of (k, ε)-coreset.

- α, β usually denote positive real numbers that represent some approximation criteria, such as the bi-criteria or (α, β)-approximation.

- $\text{cost}(\cdot)$ is always defined as some fitting function of a matrix or signal to a subspace or set of points. This is specified explicitly when used.

- $\text{dist}(p, l)$ is the regression distance between a point p and its projection on l.

- l denotes a line in $\mathbb{R}^d$.

- f usually denotes a k-segment, that is a piecewise linear function $f : \mathbb{R} \to \mathbb{R}^d$ that maps every time $i \in \mathbb{R}$ to a point $f(i)$ in $\mathbb{R}^d$.

- g denotes the 1-segment mean approximation (SVD) for a subset of a signal.

- σ usually denotes the fitting cost of a matrix or signal.

- b, e denote the beginning and end of the time interval $[b, e]$ associated with a coreset segment.

- h denotes the (α, β)-approximation for the k-segment mean of a signal.

- Q denotes a coreset segment tuple (C, g, b, e).
- $f^\star$ denotes the relevance score for keyframe selection.
- $\hat{d}$ denotes the ℓ_2 distance d combined with the relevance score $f^\star$ used computed by the modified FPS algorithm.
- L denotes the projection of a coreset C on its 1-segment mean approximation g.
- w denotes the non-negative weight vector $w = (w_1, \cdots, w_n) \in [0, \infty)^n$.
- W denotes the diagonal matrix with the non-negative weight vector along its diagonal.
- M denotes the lookup table of inner products maintained for the coreset for sum of vectors.
- J denotes the set of recorded indices $j \in [n]$.

Please see page 12 for a list of algorithms and page 13 for a list of abbreviations used in this thesis.

Chapter 3

Related Work

Our work builds on several important contributions in coresets, k-segmentations, and video summarization. This thesis we refer to work ranging from decades ago to very recent results. In this chapter we detail some of the key results.

3.1 k-segmentation

The k-segment mean problem is defined in Definition 1 in Section 4.1. The k-segment mean problem can be solved exactly using dynamic programming [14]. However, this takes $O(dn^2k)$ time and $O(dn^2)$ memory, which is impractical for streaming data. In [62, Theorem 8] a $(1+\varepsilon)$-approximation was suggested using $O(n(dk)^4 \log n/\varepsilon)$ time. While the algorithm in [62] support efficient streaming, it is not parallel. Since it returns a k-segmentation and not a coreset, it cannot be used to solve other optimization problems with additional priors or constraints. In [51] an improved algorithm that takes $O(nd^2k + ndk^3)$ time was suggested. The algorithm is based on a coreset of size $O(dk^3/\varepsilon^3)$. Unlike the coreset in this work, the running time of [51] is cubic in both d and k.

The result in [51] is the last in a line of research for the k-segment mean problem and its variations; see survey in [50, 62, 58]. The application was segmentation of 3-dimensional GPS signal (time, latitude, longitude). The coreset construction in [51] and previous papers takes time and memory that is quadratic in the dimension d and

cubic in the number of segments k. Conversely, our coreset construction takes time only linear in both k and d.

3.1.1 Approximation Algorithms

One of the main challenges in providing provable guarantees for segmentation with respect to segmentation size and quality is global optimization. Current provable algorithms for data segmentation are cubic-time in the number of desired segments, quadratic in the dimension of the signal, and cannot handle both parallel and streaming computation as desired for big data. The closest work that provides provable approximations is that of [51].

Several works attempt to summarize high-dimensional data streams in various application domains. For example, [107] describe the video stream as a high-dimensional stream and run approximated clustering algorithms such as k-center on the points of the stream; see [59] for surveys on stream summarization in robotics. The resulting k-centers of the clusters comprise the video summarization. The main disadvantages of these techniques are

(i) They partition the data stream into k clusters that do not provide k-segmentation over time. In this sense, k-segmentation can be considered as k-clustering where the assignment of points to centers is based on their context instead of only similarity of images.

(ii) Computing the k-center takes time exponential in both d and k [69]. In [107] heuristics were used for dimension reduction, and in [59] a 2-approximation was suggested for the off-line case, which was replaced by a heuristic for streaming.

(iii) In the context of analysis of video streams, they use a feature space that is often simplistic and does not utilize the large data available efficiently. In our work the feature space can be updated on-line using a coreset for k-means clustering of the features seen so far.

3.2 Video Summarization

One motivating application for us is online video summarization, where input video stream can be represented by a set of points over time in an appropriate feature space. Every point in the feature space represents the frame, and we aim to produce a compact approximation of the video in terms of this space and its Euclidean norm. In the context of video streams analysis, we build upon a multitude of prior works on for activity understanding of both robotic systems [22] and human users [9]. Application-aware summarization and analysis of ad-hoc video streams is a difficult task with many attempts aimed at tackling it from various perspectives [22, 44, 91, 9, 17]. The problem is highly related to video action classification, scene classification, and object segmentation [91]. Applications where life-long video stream analysis is crucial include mapping and navigation, medical imaging, assistive technology, and augmented-reality applications, among others. Our goal differs from video compression in that compression is geared towards preserving image quality for all frames, and therefore stores semantically redundant content. Instead, we seek a summarization approach that allows us to represent the video content by a set of key segments, for a given feature space.

The features used for this task vary from low level brightness-, gradient- and optical flow-based descriptors to quite elaborate descriptions of scene structure and content. Significant attention has been given to low-level feature extraction from video sequences for these vision tasks (see [136] for example). Maekawa et al. [94] used an HSV color codebook in order to represent activities from a wrist-based camera. Bandla and Grauman [9] use spatio-temporal features based on optical flow and HoG freatures, and Koppula et al. [77] combine descriptors and local transformation cues. Lu and Grauman [91] define the relative strength of inter-frame connections based on object cooccurrence. Directly learning features has also been attempted by deep learning techniques, see for example [8]. Developing scalable analysis schemes for summarizing large quantities of visual data is still lacking, especially as far as provably efficient and accurate algorithms are concerned.

3.2.1 Medical Video

Video-based coaching and debriefing of laparoscopic and robot-assisted minimally invasive surgery (RMIS) has been demonstrated to contribute to enhanced surgical performance [18, 123]. These procedures are typically taught in a stepwise fashion by identifying distinct steps or *phases* or an operation [76]. Context-aware segmentation of surgical video can facilitate education, time-critical consultation, and can improve perioperative workflow efficiency of operating room assignment and turnover [97, 43]. Recent studies such as [98] conclude that phase recognition is crucial for skill evaluation in robot-assisted surgery, which is still a relatively new discipline. Research in computer vision has addressed the general problem of automated video segmentation (see [114], and references therein). There has been a lot of research on surgical phase recognition [79, 16, 90, 127, 104], but this work has been mostly limited to offline video of entire procedures.

3.3 Localization and Loop Closure

Large-scale place recognition in online robotic systems relates to several active fields of research. Studied intensely between the vision [138, 125, 84, 117] and the robotics [67, 31, 85] communities, attempts have been made to allow faster loop closure processing of larger datasets [32] and to utilize 3D information in order to increase specificity [108]. Maddern et al. [93] propose a way of minimizing the effects of life-long collection of images in the mapping phase, especially with respect to location model creation.

We note that a key assumption of such algorithms is that loop closure and location pruning are achieved at full video frame rate, thereby inducing both high computational costs and the need for significant retrieval efficiencies. When 3D information is unreliable (e.g. indoor localization), when there is significant location uncertainty, or when loop closure information is scarce, the number of locations may grow with the size of the video stream. In this sense, dealing with the inherent complexity of life-long videos in the area of localization is still quite limited.

Several works in the robotics community (see for example [31, 74, 6]) attempt

to define the problem as large scale inference problem over a graphical model of observations conditioned on locations. One advantage of such methods is their natural integration with local filtering approaches for localization [36] and handling of outlier matches.

In the vision community, localization is highly related to place recognition works such as [84]. However in large scale location recognition, 2D-to-3D matching approaches (such as [86, 116]) attempt to address the retrieval problem associated with collections of large images and multiple locations. The main emphasis in these approaches is on obtaining high specificity and reducing false alarm rate – this is partially due to the fact that such systems are inherently looking for maximum probability solutions and hence attempt to prune multiple alternatives to the correct location. Several of these works obtain high efficiency by 3D-aided pruning, since they utilize the reconstructed map as well.

The use of coresets of data approximation has been considered previously with the work of [107] and [114] being the most closely related. While [107] considers the use of coresets for appearance-based mapping, navigation, and localization they do not exploit temporal consistency across frames as we do here. Additionally, the computational complexity and associated memory requirements of the proposed approach represent an exponential improvement when compared to that work. Lastly, the proposed coreset formulation allows for an unbounded set of locations while supporting location retrieval and loop closure. While in [114] we adopt a descriptor vector representation of frames and demonstrate segmentation using a derived coreset, we do not consider localization or loop closure. Furthermore, in contrast to the current formulation, the number of segments is assumed to be known *a priori*.

In the field of data approximation by coresets, the closest work to ours is that of Paul et al. [107]. Our work on visual summarization improves on the coreset of [107] in the following ways. First, unlike the algorithm in [107], the new coreset algorithm uses the temporal consistency across adjacent frames to compute better summaries. Second, the computational complexity and memory requirements are exponentially better over [107]. Third, the new coreset can handle an unbounded set of locations,

supporting location retrieval and loop closures.

It differs significantly from ours, however, in several aspects, since the coreset used differs significantly, and temporal consistency in the data is not exploited. Moreover, the complexity requirements for the coreset defined in [107] differ from ours – for example its memory complexity depends exponentially on the dimension, which is impractical for large datasets. Furthermore, the method proposed in [107] did not include a method of retrieving location from an unbounded, ever-growing, set of locations, which is key to the proposed approach. The work of [114] associates frames with descriptor vectors and perform segmentation based on the coreset. No attempt, however, is made for localization and loop closure or retrieval, and the segmentation is limited to k segments, for which a coreset approximation is constructed.

3.4 Dimensionality Reduction

In [131] it was recently proved that an (k, ε)-coreset of size $|C| = O(dk^3/\varepsilon^2)$ exists for every input matrix, and distances to the power of $z \geq 1$ where z is constant. The proof is based on a general framework for constructing different kinds of coresets, and is known as *sensitivity* [46, 83]. This coreset is efficient for tall matrices, since its cardinality is independent of n. However, it is useless for "fat" or square matrices (such as the Wikipedia matrix above), where d is in the order of n, which is the main motivation for our work. In [23], the Frank-Wolfe algorithm was used to construct different types of coresets than ours, and for different problems. Our approach is based on a solution that we give to an open problem in [23], however we can see how it can be used to compute the coresets in [23] and vice versa. For the special case $z = 2$ (sum of squared distances), a coreset of size $O(k/\varepsilon^2)$ was suggested in [27] with a randomized version in [26] for a stream of n points that, unlike the standard approach of using merge-and-reduce trees, returns a coreset of size independent of n with a constant probability. These result minimizes the $\|\cdot\|_2$ error, while our result minimizes the Frobenius norm, which is always higher, and may be higher by a factor of d. After appropriate weighting, we can apply the uniform sampling of size $O(k/\varepsilon^2)$

to get a coreset with a small Frobenius error [73], as in our work. However, in this case the probability of success is only constant. Since in the streaming case we compute roughly n coresets (formally, $O(n/m)$ coresets, where m is the size of the coreset) the probability that all these coresets constructions will succeed is close to zero (roughly $1/n$). Since the probability of failure in [73] reduces linearly with the size of the coreset, getting a constant probability of success in the streaming model for $O(n)$ coresets would require to take coresets of size that is no smaller than the input size.

There are many papers, especially in recent years, regarding data compression for computing the SVD of large matrices. None of these works addresses the fundamental problem of computing a sparse approximated PCA for a large matrix (in both rows and columns), such as Wikipedia. The reason is that current results use sketches which do no preserve the sparsity of the data (e.g. because of using random projections). Hence, neither the sketch nor the PCA computed on the sketch is sparse. On the other side, we define coreset as a small weighted subset of rows, which is thus sparse if the input is sparse. Moreover, the low rank approximation of a coreset is sparse, since each of its right singular vectors is a sum of a small set of sparse vectors. While there are coresets constructions as defined in this work, all of them have cardinality of at least d points, which makes them impractical for large data matrices, where $d \geq n$. In what follows we describe these recent results in details.

The recent results in [27, 26] suggest coresets that are similar to our definition of coresets (i.e., weighted subsets), and do preserve sparsity. However, as mentioned above they minimize the 2-norm error and not the larger Frobenius error, and maybe more important, they provide coresets for k-SVD (i.e., k-dimensional subspaces) and not for PCA (k-dimensional affine subspaces that might not intersect the origin). In addition [26] works with constant probability, while our algorithm is deterministic (works with probability 1). Some recent papers can be considered as applications and extensions of our papers. We expect that there will be many more such related results and applications in the future, e.g. a coreset for an SVM classifier can also be reduced to a sum of vectors [23]. Our recent arXiv publication [53] was already cited and inspired other papers, including one from Google [5].

3.4.1 Coresets

Following a decade of research (e.g. [47, 45, 64, 39, 41, 40, 46]), it was recently proven that an (ε, k)-coreset for low rank approximation of size $|C| = O(dk^3/\varepsilon^2)$ exists for every input matrix [131]. The proof is based on a general framework for constructing different kinds of coresets, and is known as *sensitivity* [46, 83]. This coreset is efficient for tall matrices, since its cardinality is independent of n. However, it is useless for "fat" or square matrices (such as the Wikipedia matrix above), where d is in the order of n, which is the main motivation for our work. In [23], the Frank-Wolfe algorithm was used to construct different types of coresets than ours, and for different problems. Our approach is based on a solution that we give to an open problem in [23].

3.4.2 Sketches

A *sketch* in the context of matrices is a set of vectors $u_1, \cdots, u_s$ in $\mathbb{R}^d$ such that the sum of squared distances $\sum_{i=1}^{n}(\text{dist}(a_i, S))^2$ from the input n points to *every* k-dimensional subspace S in $\mathbb{R}^d$, can be approximated by $\sum_{i=1}^{n}(\text{dist}(u_i, S))^2$ up to a multiplicative factor of $1 \pm \varepsilon$. Note that even if the input vectors $a_1, \cdots, a_n$ are sparse, the sketched vectors $u_1, \cdots, u_s$ in general are not sparse, unlike the case of coresets. A sketch of cardinality d can be constructed with no approximation error ($\varepsilon = 0$), by defining $u_1, \cdots, u_d$ to be the d rows of the matrix DV^T where $UDV^T = A$ is the SVD of A. It was proved in [48] that taking the first $O(k/\varepsilon)$ rows of DV^T yields such a sketch, i.e. of size independent of n and d.

Unlike the other coresets, this sketch is of cardinality independent of both n and d. Using the merge-and-reduce technique, this sketch can be computed in the streaming and parallel computation models. The final coreset size will be increased by a factor of $O(\log n)$ in this case. For the streaming case, it was shown in [87, 56] that such a strong sketch of the same size, $O(k/\varepsilon)$, can be maintained without the additional $O(\log n)$ factor. In this case, the running time is longer by a factor of the coreset size $|C|$, since the merge-and-reduce tree is no longer used.

The first sketch for sparse matrices was suggested in [24], but like more recent

results, it assumes that the complete matrix fits in memory. Other sketching methods that usually do not support streaming include random projections [7, 2, 42] and randomly combined rows [106, 132, 115, 88].

The Lanczoz method [105] and its variant [75] multiply a large matrix by a vector for a few iterations to get its largest eigenvector v_1. Then the computation is done recursively after projecting the matrix on the hyperplane that is orthogonal to v_1. Multiplying a matrix by a vector can be done easily in the streaming mode without having all the matrix in memory, although it requires multiple passes over the data compared to the one-pass model that we consider. If A is sparse then the computation of Ax_i for $1 < i < m$ takes time that depends only on the non-zeroes-entries of A. However, v_1 is in general not sparse even A is sparse. Hence, when we project A on the orthogonal subspace to v_1, the resulting matrix is dense for the rest of the computations ($k > 1$). Indeed, our experimental results show that the MATLAB `svds` function which uses this method runs faster than the exact SVD, but crashes on large input, even for small k. This thesis builds on this extensive body of prior work in dimensionality reduction, and our approach uses coresets to solve the time and space challenges.

3.4.3 Lower bounds

Recently, [56, 87] proved a lower bound of $O(k/\varepsilon)$ for the cardinality of a strong sketch. A lower bound of $O(k/\varepsilon^2)$ for strong coreset was proved for the special case $k = d - 1$ in [10]. Our algorithm is more efficient and numerically stable than the approach in [10], since, for example, it doesn't need to compute the inverse of the input matrix. Also, the error in [10] that corresponds to the error matrix is measured with respect to the 2-norm, while we bound the (always higher) Frobenius norm of this matrix.

3.4.4 Software

Popular software for computing SVD such as GenSim [113], redsvd [61] or the MATLAB sparse SVD function (`svds`) use sketches and crash for inputs of a few thousand of documents and a dimensionality reduction (approximation rank) $k < 100$ on a regular laptop, as expected from the analysis of their algorithms. This is why existing implementations (including Gensim) extract topics from large matrices (e.g. Wikipedia), based on low-rank approximation of only small subset of few thousands of selected words (matrix columns), and not the complete Wikipedia matrix. Even for $k = 3$, running the implementation of sparse SVD in Hadoop [121] took several days [63]. Next we give a broad overview of the very latest state of the dimensionality reduction methods, such as the Lanczoz algorithm [82] for large matrices, that such systems employ under the hood.

Chapter 4

Coresets for Segmentation of Temporal Data

In this chapter[1] we present the k-segment coreset, and develop the theoretical framework to enable fast content-based segmentation of data streams. We propose a new coreset for the k-segmentation problem (see Definition 1 in Section 4.1) that can be computed in one pass over the streaming data, with insertion time and space that are dependent poly-logarithmically in n. Unlike previous results, the insertion time per new observation and required memory is only linear in both the dimension d and the number k of segments. This result is formalized in Theorem 2. Our algorithm is scalable, parallelizable, and provides a provable approximation of the cost function. Using this coreset construction we present a system for compression of large-scale data. Our approach allows real-time segmentation of video streams such as video in a way that preserves the semantic content of the aggregated video sequences, and is easily extendable.

Chapters 4 and 5 are organized as follows. We begin by describing the k-segmentation problem in Section 4.1. We present our proposed coresets, describe their construction, and detail their properties in Section 4.2. In the next chapter we describe our system implementation and present experimental results. In Section 5.1 we perform several

[1]Some of the content in this chapter was published in [114].

experiments in order to validate our proposed approach on real data.

4.1 Problem Formulation

The k-segment mean problem optimally fits a given discrete time signal of n points by a set of k linear segments over time, where $k \geq 1$ is a given integer. That is, we wish to partition the signal into k consecutive time intervals such that the points in each time interval are lying on a single line; see Fig. 4-1(left) and the following formal definition.

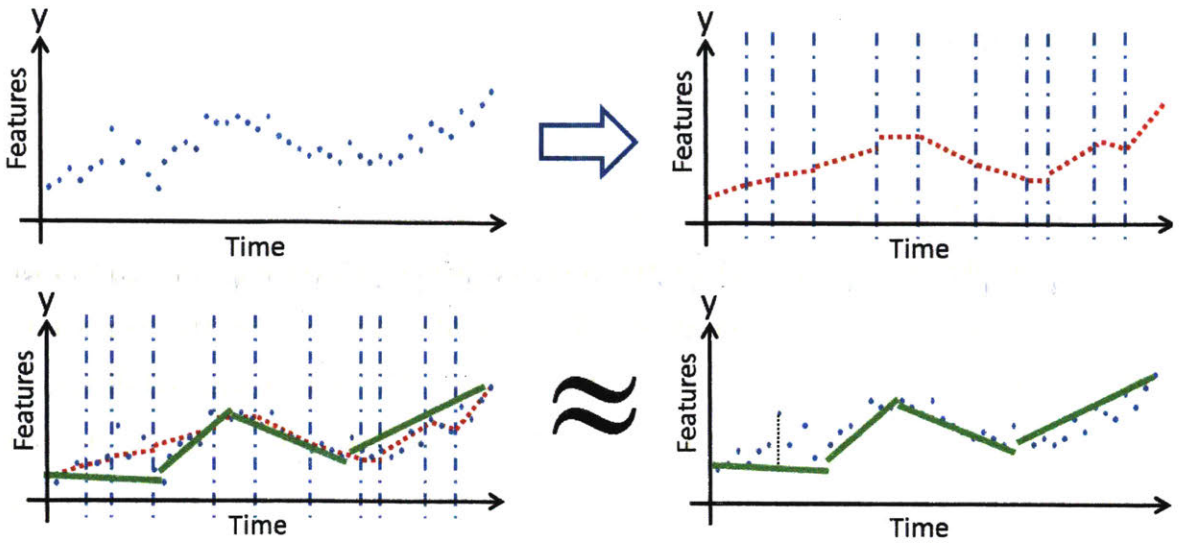
We make the following assumptions with respect to the data: (a) We assume the data is represented by a feature space that suitably represents its underlying structure; (b) The content of the data includes at most k segments that we wish to detect automatically; An example for this are scenes in a video, phases in the market as seen by stock behaviour, etc. and (c) The dimensionality of the feature space is often quite large (from tens to thousands of features), with the specific features depending on the application – several examples are given in Section 5.1. This motivates the following problem definition.

Definition 1 (k-segment mean). *A set P in $\mathbb{R}^{d+1}$ is a* signal *if $P = \{(1, p_1), \cdots, (n, p_n)\}$ where $p_j \in \mathbb{R}^d$ is the point at time index j for every $j \in [n] = \{1, \cdots, n\}$. For an integer $k \geq 1$, a* k-segment *is a k-piecewise linear function $f : \mathbb{R} \to \mathbb{R}^d$ that maps every time $t \in \mathbb{R}$ to a point $f(t)$ in $\mathbb{R}^d$. The* fitting error *at time j is the squared distance between p_j and its corresponding projected point $f(i)$ on the k-segments. The* fitting cost *of f to P is the sum of these squared distances,*

$$\text{cost}(P, f) = \sum_{j=1}^{n} \|p_j - f(j)\|_2^2, \tag{4.1}$$

where $\|\cdot\|$ denotes the Euclidean distance. The function f is a k-segment mean *of P if it minimizes* $\text{cost}(P, f)$.

For the case $k = 1$ the 1-segment mean is the solution to the linear regression problem. If we restrict each of the k-segments to be a horizontal segment, then each



For every k-segment f, the cost of input points (blue) is approximated by the cost of the coreset. Left to right, top to bottom: (a) An input signal; (b) The coreset consists of the partition of the input into a few segments (dashed blue vertical lines), with approximate per-segment representation of the data (dashed red lines). The cost of f is the sum of these squared distances from all the input points; (c) Given any k-segment, (colored in green) model, we can compute an ℓ_2-fitting cost using the coreset; (d) The fitting cost should approximate the ℓ_2 fitting cost on the original data points.

Figure 4-1: k-segment coreset illustration

segment will be the mean height of the corresponding input points. The resulting problem is similar to the k-mean problem, except each of the voronoi cells is forced to be a single region in time, instead of nearest center assignment, i.e. the regions are contiguous.

In this chapter we seek a compact representation D that approximates $\text{cost}(P, f)$ for every k-segment f using the above definition of $\text{cost}'(D, f)$. We denote a set D as a (k, ε)-coreset according to the following definition,

Definition 2 ((k, ε)-coreset). *Let $P \subseteq \mathbb{R}^{d+1}$, $k \geq 1$ be an integer, for some small $\varepsilon > 0$. A set D, with a cost function $\text{cost}'(\cdot)$ is a (k, ε)-coreset for P if for every k-segment f we have*

$$(1 - \varepsilon)\text{cost}(P, f) \leq \text{cost}'(D, f) \leq (1 + \varepsilon)\text{cost}(P, f).$$

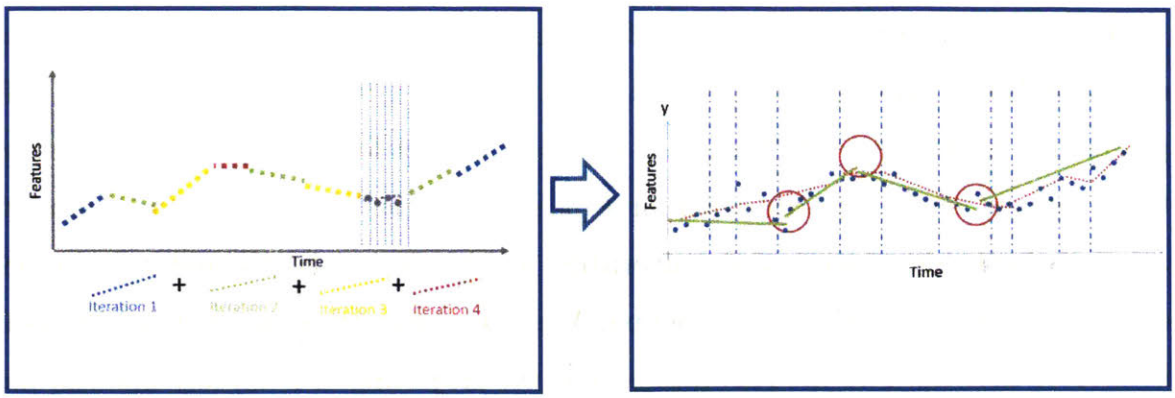
We present a new coreset construction with provable approximations for a family of natural k-segmentation optimization problems. This is the first such construction whose running time is linear in both the number of data points n, their dimensionality d, and the number k of desired segments. The resulting coreset consists of $O(dk/\varepsilon^2)$ vectors that allow us to approximate the sum of square distances for *any* k-piecewise linear function (k segments over time). In particular, we can use this coreset to compute the k-piecewise linear function that minimize the sum of squared distances to the input points, given arbitrary constraints or weights (priors) on the desired segmentation. Such a generalization is useful, for example, when we are already given a set of candidate segments (e.g. maps or distribution of images) and wish to choose the right k segments that approximate the input signal.

Previous results on coresets for k-segmentation achieved running time or coreset size that are at least quadratic in d and cubic in k [51, 50]. As such, they are less suitable for large-scale data such as long streaming video data which is inherently high-dimensional and contains a large number of scenes. This prior work is based on some non-uniform sampling of the input data. In order to achieve our results, we had to replace the sampling approach by a new set of deterministic algorithms that carefully select the coreset segments and their internal representation.

4.2 A Novel Coreset for k-segment Mean

We now describe the construction of the coreset. We detail the main theorems and proofs, after giving the intuition behind the construction, which is illustrated in Figure 4-2.

The key insights for constructing the k-segment coreset are: i) We observe that for the case $k = 1$, a 1-segment coreset can be easily obtained using SVD. ii) For the general case, $k \geq 2$ we can partition the signal into a suitable number of intervals, and compute a 1-segment coreset for each such interval. If the number of intervals and their lengths are carefully chosen, most of them will be well approximated for every k-segmentation, and the remaining intervals will not incur a large error con-



Flowchart of our algorithm: (a) Use the BICRITERIA algorithm to estimate the signal complexity. (b) Use the BALANCEDPARTITION algorithm to compute the coreset.

Figure 4-2: k-segment coreset flowchart

tribution. Based on these observations, we propose the following construction. 1) Estimate the signal's complexity, i.e., the approximated fitting cost to its k-segment mean. We denote this step as a call to the algorithm BICRITERIA. 2) Given an complexity measure for the data, approximate the data by a set of segments with auxiliary information, which is the proposed coreset, denoted as the output of algorithm BALANCEDPARTITION.

We then prove that the resulting coreset allows us to approximate with guarantees the fitting cost for any k-segmentation over the data, as well as compute an optimal k-segmentation. We state the main result in Theorem 2, and describe the proposed algorithms as Algorithms 2 and 1, respectively.

4.2.1 Computing a k-segment Coreset – Overview

We would like to compute a (k, ε)-coreset for our data. A (k, ε)-coreset D for a set P approximates the fitting cost of any query k-segment to P up to a small multiplicative error of $1 \pm \varepsilon$. We note that a $(1, 0)$-coreset can be computed using SVD; See Appendix 4.8.1 for details and proof. However, for $k > 2$, we cannot approximate the data by a representative point set (as shown in Appendix 4.8.1, Corollary 2). Instead, we define a data structure D as our proposed coreset, and define a new cost function $\text{cost}'(D, f)$ that approximates the cost of P to any k-segment f.

The proposed coreset D consists of tuples of the type (C_i, g_i, b_i, e_i). Each tuple corresponds to a different time interval $[b_i, e_i]$ in $\mathbb{R}$ and represents the set $P(b_i, e_i)$ of points of P in this interval. The set C_i is a $(1, \varepsilon)$-coreset for $P(b_i, e_i)$, and g_i denotes the 1-segment mean approximation for $P(b_i, e_i)$. We refer to these tuples as *coreset segments* in the description of the algorithm. For brevity's sake, the index i may be omitted where it is clear from the context. We assume that the time (first coordinate) is discrete between 1 to n. This means that the projecting of P on any line can be described exactly in $O(d)$ space using only the first and last projected points, which motivates the structure of D, and the element g_i.

We note the following:

(1) If all the points of the k-segment f are on the same segment in this time interval, i.e, $\{f(t) \mid b \leq t \leq e\}$ is a linear segment, then the cost from $P(b, e)$ to f can be approximated well by C, up to $(1 + \varepsilon)$ multiplicative error (see Appendix 4.8.1).

(2) If we project the points of $P(b, e)$ on g, then the projected set L of points will approximate well the cost of $P(b, e)$ to f, even if f corresponds to more than one segment in the time interval $[b, e]$. Unlike the previous case, the error here is additive.

(3) Since f is a k-segment there will be at most $k-1$ coreset segments whose time interval intersects more than two segments of f, so the overall additive error is small. This motivates the following definition of D and cost$'$.

Definition 3 (cost$'(D, f)$). *Let $D = \{(C_i, g_i, b_i, e_i)\}_{i=1}^{m}$ where for every $i \in [m]$ we have $C_i \subseteq \mathbb{R}^{d+1}$, $g_i : \mathbb{R} \to \mathbb{R}^d$ and $b_i \leq e_i \in \mathbb{R}$. For a k-segment $f : \mathbb{R} \to \mathbb{R}^d$ and $i \in [m]$ we say that C_i is* served by one segment *of f if $\{f(t) \mid b_i \leq t \leq e_i\}$ is a linear segment. We denote by* $\text{Good}(D, f) \subseteq [m]$ *the union of indexes i such that C_i is served by one segment of f. We also define $L_i = \{g_i(t) \mid b_i \leq t \leq e_i\}$, the projection of C_i on g_i. We define* cost$'(D, f)$ *as* $\sum_{i \in \text{Good}(D,f)} \text{cost}(C_i, f) + \sum_{i \in [m] \setminus \text{Good}(D,f)} \text{cost}(L_i, f)$.

Our coreset construction for general $k > 1$ is based on a data-dependent input parameter $\sigma > 0$ such that for an appropriate σ the output is a (k, ε)-coreset. For the purpose of constructing our coreset we will require the definition of an (α, β)-approximation,

Definition 4 ((α,β)-approximation). *For $\alpha,\beta>0$, an (α,β)-approximation for the k-segment mean of P is a $(k\cdot\beta)$-segment g such that* $\text{cost}(P,g)\leq\alpha\cdot\text{cost}(P,f^*)$.

Specifically, we show that Algorithm 2 constructs an (α,β)-approximation, for α,β small enough so as to estimate the complexity of the data. We show that using the minimal value $\text{cost}(P,g)$ even without knowing g, suffices to get a (k,ε)-coreset, using the BALANCEDPARTITION(P,ε,σ) algorithm, given as Algorithm 1.

4.3 k-segment mean

Let $P=\{(t_1,p_1),\cdots,(t_n,p_n)\}$ be a subset of $\mathbb{R}^{d+1}$ where $t_i\in\mathbb{R}$ and $p_j\in\mathbb{R}^d$ for every $j\in[n]=\{1,\cdots,n\}$. The *fitting cost* (henceforth simply "cost") from P to a k-segment f is the sum of squared distances

$$\text{cost}\,(P,f)=\sum_{(t,p)\in P}\|(p-f(t))\|^2, \tag{4.2}$$

where $\|X\|^2=\sum_{ij}(X_{ij})^2$ is the sum of squared entries of a matrix or a vector X (known as the Frobenius norm for a matrix or the ℓ_2 Euclidean norm for a vector).

A *k-segment mean* of P is a k-segment $f^*:\mathbb{R}\to\mathbb{R}^d$ that minimizes $\text{cost}(P,f)$ over every k-segment $f:\mathbb{R}\to\mathbb{R}^d$. For $\alpha\geq1$, an *α-approximation for the k-segment mean of P* is a k-segment f such that $\text{cost}(P,f)\leq\alpha\cdot\text{cost}(P,f^*)$. For $\alpha,\beta>0$, an (α,β)-approximation for the k-segment mean of P is a $(k\cdot\beta)$-segment g such that $\text{cost}(P,g)\leq\alpha\cdot\text{cost}(P,f^*)$.

One of our main tool for computing approximations to the k-segment mean is the *singular value decomposition* (SVD) which is defined as follows. For integers $n,d\geq1$ we denote by $\mathbb{R}^{n\times d}$ the set $n\times d$ matrices whose entries are in $\mathbb{R}$. A *unitary* matrix is a matrix whose columns are orthonormal vectors. The thin SVD of a matrix $X\in\mathbb{R}^{n\times d}$ is $X=UDV^T$ where both $U\in\mathbb{R}^{n\times d}$ and $V\in\mathbb{R}^{d\times d}$ are unitary matrices, and $D\in\mathbb{R}^{d\times d}$ is a diagonal matrix of non-negative and non-increasing diagonal entries. As we show in Appendix 4.8.1, we can use the SVD to get a $(1,0)$-coreset for 1-segments.

4.4 Balanced Partition

We now proceed to describe our coreset construction, according to the outline in Section 4.2.1.

We will compute such a small structure D that approximates $\text{cost}(P, f)$ for every k-segment f using the above definition of $\text{cost}'(D, f)$. Such a set D will be called a (k, ε)-coreset as follows.

Definition 5 ((k, ε)-coreset). *Let $P \subseteq \mathbb{R}^{d+1}$, $k \geq 1$ be an integer, and let $\varepsilon \in (0, 1/10)$. The set D is a (k, ε)-*coreset *for P if for every k-segment f we have*

$$(1 - \varepsilon)\text{cost}(P, f) \leq \text{cost}'(D, f) \leq (1 + \varepsilon)\text{cost}(P, f).$$

Our coreset construction is based on an input parameter $\sigma > 0$ such that for an appropriate σ the output is a (k, ε)-coreset. Recall that for $\alpha, \beta > 0$, an (α, β)-approximation for the k-segment mean of P is a $(k \cdot \beta)$-segment g such that $\text{cost}(P, g) \leq \alpha \cdot \text{cost}(P, f^*)$. We show that using the value $\text{cost}(P, g)$ of such an approximation, even without knowing g, suffices to get a (k, ε)-coreset. In the next section we will compute such an (α, β)-approximation for small α and β.

The size of the resulting coreset depends on α and β. In particular, for $\alpha = \beta = 1$ the following lemma implies that there exists a (k, ε)-coreset of size $O(k/\varepsilon^2)$ for every input set P.

Lemma 1. *Let $P = \{(1, p_1), \cdots, (n, p_n)\}$ such that $p_j \in \mathbb{R}^d$ for every $j \in [n]$. Suppose that $h : \mathbb{R} \to \mathbb{R}^d$ is an (α, β)-approximation for the k-segment mean of P, and let*

$$\sigma = \frac{\varepsilon^2 \text{cost}(P, h)}{100k\alpha}.$$

Let D be the output of a call to BalancedPartition(P, ε, σ)*; See Algorithm 1.*

Then D is a (k, ε)-coreset for P of size

$$|D| = O(k) \cdot \left(\frac{\alpha}{\varepsilon^2} + \beta\right),$$

Algorithm 1 BALANCEDPARTITION(P, ε, σ)

Input: A set $P = \{(1, p_1), \cdots, (n, p_n)\}$ in $\mathbb{R}^{d+1}$, an error parameters $\varepsilon \in (0, 1/10)$ and $\sigma > 0$.

Output: A set D that satisfies Lemma 1.

```
Q := ∅; D = ∅
p_{n+1} := an arbitrary point in R^d
for j := 1 to n + 1 do
    Q := Q ∪ {(j, p_j)}
    f* := a 2-approximation to the 1-segment mean of Q.  // See Corollary 2
    λ := cost(Q, f*)
    if λ > σ or j = n + 1 then
        T := Q \ {(j, p_j)}  // Define the new coreset segment data up to j
        C := a (1, ε/4)-coreset for T  // See Claim 1
        g := a 2-approximation to the 1-segment mean of T  // See Corollary 2
        b := j − |T|
        e := j − 1
        D := D ∪ {(C, g, b, e)} // Add a new coreset segment
        Q := {(j, p_j)} // Start aggregating a new coreset segment
    end if
end for
return D
```

and can be computed in $O(dn/\varepsilon^4)$ time.

Proof. Let $m = |D|$ and f be a k-segment. We denote the ith coreset segment in D by (C_i, g_i, b_i, e_i) for every $i \in [m]$. For every $i \in [m]$ we have that C_i is a $(1, \varepsilon/4)$-coreset for a corresponding subset $T = T_i$ of P. By the construction of D we also have $P = T_1 \cup \cdots \cup T_m$.

Using Definition 3 of $\text{cost}'(D, f)$, $\text{Good}(D, f)$ and L_i, we thus have

$$
\begin{aligned}
&|\text{cost}(P, f) - \text{cost}'(D, f)| \\
&= |\sum_{i=1}^{m} \text{cost}(T_i, f) - \left(\sum_{i \in \text{Good}(D,f)} \text{cost}(C_i, f) + \sum_{i \in [m] \setminus \text{Good}(D,f)} \text{cost}(L_i, f) \right)| \\
&= |\sum_{i \in \text{Good}(D,f)} (\text{cost}(T_i, f) - \text{cost}(C_i, f)) + \sum_{i \in [m] \setminus \text{Good}(D,f)} (\text{cost}(T_i, f) - \text{cost}(L_i, f)) \\
&\leq \sum_{i \in \text{Good}(D,f)} |\text{cost}(T_i, f) - \text{cost}(C_i, f)| + \sum_{i \in [m] \setminus \text{Good}(D,f)} |\text{cost}(T_i, f) - \text{cost}(L_i, f)|,
\end{aligned}
\tag{4.3}
$$

where the last inequality is due to the triangle inequality. We now bound each term in the right hand side.

For every $i \in \text{Good}(D, f)$ we have that C_i is a $(1, \varepsilon/4)$-coreset for T_i, so

$$|\text{cost}(T_i, f) - \text{cost}(C_i, f)| \leq \frac{\varepsilon \text{cost}(T_i, f)}{4}. \tag{4.4}$$

For every $i \in [m] \setminus \text{Good}(D, f)$, we have

$$\begin{aligned}
|\text{cost}(T_i, f) - \text{cost}(L_i, f)| &= \left| \sum_{(p,t) \in T_i} \|p - f(t)\|^2 - \sum_{t=b_i}^{e_i} \|g_i(t) - f(t)\|^2 \right| \\
&= \left| \sum_{(p,t) \in T_i} \left(\|p - f(t)\|^2 - \|g_i(t) - f(t)\|^2 \right) \right| \qquad (4.5) \\
&\leq \sum_{(p,t) \in T_i} \left| \|p - f(t)\|^2 - \|g_i(t) - f(t)\|^2 \right| \qquad (4.6) \\
&\leq \sum_{(p,t) \in T_i} \left(\frac{12 \|g_i(t) - p\|^2}{\varepsilon} + \frac{\varepsilon \|p - f(t)\|^2}{2} \right) \qquad (4.7) \\
&= \frac{12\text{cost}(T_i, g_i)}{\varepsilon} + \frac{\varepsilon \text{cost}(T_i, f)}{2} \leq \frac{24\sigma}{\varepsilon} + \frac{\varepsilon \text{cost}(T_i, f)}{2}, \qquad (4.8)
\end{aligned}$$

where (4.6) is by the triangle inequality, and (4.7) is by the weak triangle inequality (see [49, Lemma 7.1]). The inequality in (4.8) is because by construction $\text{cost}(T, f^*) \leq \sigma$ for some 2-approximation f^* of the 1-segment mean of T. Hence, $\text{cost}(T, g_i) \leq 2\text{cost}(T, f^*) \leq 2\sigma$.

Plugging (4.8) and (4.4) in (4.3) yields

$$\begin{aligned}
|\text{cost}(P, f) - \text{cost}'(D, f)| &\leq \sum_{i \in \text{Good}(D,f)} \frac{\varepsilon \text{cost}(T_i, f)}{4} + \sum_{i \in [m] \setminus \text{Good}(D,f)} \left(\frac{24\sigma}{\varepsilon} + \frac{\varepsilon}{2} \text{cost}(T_i, f) \right) \\
&\leq \left(\frac{\varepsilon}{4} + \frac{\varepsilon}{2} \right) \text{cost}(P, f) + \frac{24k\sigma}{\varepsilon},
\end{aligned}$$

where in the last inequality we used that fact that $|[m] \setminus \text{Good}(D, f)| \leq k - 1 < k$

since f is a k-segment. Substituting σ yields

$$|\text{cost}(P,f) - \text{cost}'(D,f)| \leq \frac{3\varepsilon}{4}\text{cost}(P,f) + \frac{\varepsilon\text{cost}(P,h)}{4\alpha} \leq \frac{3\varepsilon}{4}\text{cost}(P,f) + \frac{\varepsilon\text{cost}(P,f)}{4} = \varepsilon\text{cost}(P,f).$$

Bound on $|D|$: Let $j \in [m-1]$, consider the values of T, Q and λ during the execution of Line 8 when $T = T_j$ is constructed. Let $Q_j = Q$ and $\lambda_j = \lambda$. The cost of the 1-segment mean of Q_j is at least $\lambda_j/2 > \sigma/2 > 0$, which implies that $|Q_j| \geq 3$ and thus $|T_j| \geq 1$. Since Q_{j-1} is the union of T_{j-1} with the first point of T_j we have $Q_{j-1} \subseteq T_{j-1} \cup T_j$. By letting g^* denote a 1-segment mean of $T_{j-1} \cup T_j$ we have

$$\text{cost}(T_{j-1} \cup T_j, g^*) \geq \text{cost}(Q_{j-1}, g^*) \geq \lambda_j/2 > \sigma/2.$$

Suppose that for our choice of $j \in [m-1]$, the points in $T_{j-1} \cup T_j$ are served by a single segment of h, i.e, $\{h(t) \mid b_{j-1} \leq t \leq e_j\}$ is a linear segment. Then

$$\text{cost}(T_{j-1}, h) + \text{cost}(T_j, h) = \text{cost}(T_{j-1} \cup T_j, h) \geq \text{cost}(T_{j-1} \cup T_j, g^*) > \sigma/2. \quad (4.9)$$

Let $G \subseteq [m-1]$ denote the union over all values $j \in [m-1]$ such that j is both even and satisfies (4.9). Summing (4.9) over G yields

$$\text{cost}(P,h) = \sum_{j \in [m]} \text{cost}(T_i, h) \geq \sum_{j \in G} (\text{cost}(T_{j-1}, h) + \text{cost}(T_j, h)) \geq |G|\sigma/2. \quad (4.10)$$

Since h is a (βk)-segment, at most $(\beta k) - 1$ sets among $T_1, \cdots, T_m$ are not served by a single segment of h, so $|G| \geq (m - \beta k)/2$. Plugging this in (4.10) yields $\text{cost}(P,h) \geq (m - \beta k)\sigma/4$. Rearranging,

$$m \leq \frac{4\text{cost}(P,h)}{\sigma} + \beta k = O\left(\frac{k\alpha}{\varepsilon^2}\right) + \beta k. \quad (4.11)$$

Running time: In Theorem 6 it was shown how to compute a $(1,\varepsilon)$-coreset C in time $O(dn/\varepsilon^4)$ for n points using the algorithm in [57]. This algorithm is dynamic and supports insertion of a new point in $O(d/\varepsilon^4)$ time. Therefore, updating the 1-

Algorithm 2 BICRITERIA(P, k)

Input: A set $P \subseteq \mathbb{R}^{d+1}$ and an integer $k \geq 1$

Output: h, an $(O(\log n), O(\log n))$-approximation to the k-segment mean of P.

1: **if** $n \leq 2k+1$ **then**
2: Set $f \leftarrow$ a 1-segment mean of P
3: **return** f
4: **end if**
5: Set $t_1 \leq \cdots \leq t_n$ and $p_1, \cdots, p_n \in \mathbb{R}^d$ such that $P = \{(t_1, p_1), \cdots, (t_n, p_n)\}$
6: Set $m \leftarrow \{t \in \mathbb{R} \mid (t,p) \in P\}$
7: Partition P into $4k$ sets $P_1, \cdots, P_{4k} \subseteq P$ such that for every $i \in [4k-1]$:
(i) $|\{t \mid (t,p) \in P_i\}| = \left\lfloor \frac{m}{4k} \right\rfloor$, and (ii) if $(t,p) \in P_i$ and $(t',p') \in P_{i+1}$ then $t < t'$.
8: **for** $i = 1$ to $4k$ **do**
9: Compute a 2-approximation g_i to the 1-segment mean of P_i
10: **end for**
11: Set $Q \leftarrow$ the union of $k+1$ signals P_i with the smallest value $\text{cost}(P_i, g_i)$ among $i \in [2k]$.
12: Set $h \leftarrow$ BICRITERIA$(P \setminus Q, k)$; *// Repartition the segments that did not have a good approximation*
13: Set

$$f(t) := \begin{cases} g_i(t) & \exists (t,p) \in P_i \text{ such that } P_i \subseteq Q \\ h(t) & \text{otherwise} \end{cases}.$$

14: **return** f

segment mean f^* and the coreset C can be done in $O(d/\varepsilon^4)$ time per point, and the overall running time is $O(nd/\varepsilon^4)$. □

4.5 (α, β)-Approximation and the Bicriteria Algorithm

We now describe the BICRITERIA(P, k) algorithm. This algorithm allows us to estimate the minimal k-segmentation cost for the signal, which is required for a balanced-complexity coreset. It is described as Algorithm 2.

Theorem 1. *Let $f : \mathbb{R} \to \mathbb{R}^d$ be the output of a call to* BICRITERIA(P, k). *Then*

(i) *f is a (βk)-segment for some $\beta = O(\log n)$.*

(ii) $\text{cost}(P, f) \leq \alpha \text{cost}(P, f^*)$, *where $\alpha = \log_2 n$, and f^* is a k-segment mean of P.*

(iii) *f can be computed in $O(dn)$ time.*

Proof. ***(i)*** In every recursive iteration of the algorithm we remove $(k-1)$ subsets of P, whose overall size is

$$|Q| \geq (k-1) \cdot \left\lfloor \frac{n}{4k} \right\rfloor \geq (k-1) \cdot \left(\frac{n}{4k} - 1\right) = \frac{n}{4} - \frac{n}{4k} - (k-1) \geq \frac{n}{4} - \frac{n}{8} - \frac{n}{12} = \frac{n}{24}.$$

where in the last inequality we used the assumption $k \in [2, n/12 + 1]$. Hence, the size of P reduced by a constant fraction in each recursive iteration and we have $O(\log n)$ iterations.

Each subset P_i in Q contributes at most 2 segments to f, so the number of segments in f increases by $O(k)$ in each of the $O(\log n)$ recursive iterations. Hence, the final output f has $O(k \log n)$ segments.

(ii) Consider the value of P during one of the recursive iterations. Since f^* is a k-segment, every set in $P_1, \cdots, P_{4k}$ is served by one segment of f^*, except at most $k-1$ such subsets. Let $M \subseteq [4k]$ denote the indexes of these (at most $k-1$) subsets, and let $W = [4k] \setminus M$ denote the rest, such that $Q = \bigcup_{i \in W} P_i$. Hence,

$$\text{cost}(P, f^*) \geq \sum_{i \in W} \text{cost}(P_i, f^*) \geq \sum_{i \in W} \min_g \text{cost}(P_i, g) \geq \frac{1}{2} \sum_{i \in [4k] \setminus M} \text{cost}(P_i, g_i), \quad (4.12)$$

where the minimum is over every 1-segment $g : \mathbb{R} \to \mathbb{R}^d$. Since

$$|[4k] \setminus M| = 4k - |M| \geq 4k - (k-1) = 3k + 1,$$

we have

$$\sum_{i \in [4k] \setminus M} \text{cost}(P_i, g_i) \geq \sum_{i \in W} \text{cost}(P_i, g_i) = \sum_{i \in W} \text{cost}(P_i, f) = \text{cost}(Q, f).$$

Plugging the last inequality in (4.12) yields $\text{cost}(Q, f) \leq 2\text{cost}(P, f^*)$. Summing over all iterations proves the claim.

(iii) In each recursive iteration, the dominated running time is in the "for" loop in

Algorithm 3 CORESET(P, k, ε)

Input: A set $P = \{(1, p_1), \cdots, (n, p_n)\}$ in $\mathbb{R}^{d+1}$.
Output: A (k, ε)-coreset (C, w) that satisfies Theorem 2.

1: Compute $h \leftarrow$ BICRITERIA(P, k) ; See Algorithm 2
2: Set $\sigma \leftarrow \frac{\varepsilon^2 \text{cost}(P,h)}{100k \log_2 n}$
3: Set $D \leftarrow$ BALANCEDPARTITION(P, ε, σ) *//See Algorithm 1*
4: **return** D

Lines 8–9. We compute a 2-approximation g_i for the 1-segment mean of a set P_i of m points in $O(md)$ time using Corollary 2. Hence, the overall time to compute Lines 8–9 is $O(nd)$. Since the size of P reduced by a constant fraction in each recursive iteration, the overall running time is dominated by the first iteration which takes $O(nd)$ time. □

4.6 (k, ε)-Coreset

We now define the k-segment coreset construction algorithm and prove bounds on its desisred tradeoff of size, construction complexity, and accuracy of representation.

Theorem 2. *Let $P = \{(1, p_1), \cdots, (n, p_n)\}$ such that $p_j \in \mathbb{R}^d$ for every $j \in [n]$. Let D be the output of a call to* CORESET(P, k, ε)*; see Algorithm 3.*
Then D is a (k, ε)-coreset for P of size

$$|D| = O(k) \cdot \left(\frac{\log n}{\varepsilon^2}\right),$$

and can be computed in $O(dn/\varepsilon^4)$ time.

Proof. By Theorem 1, h is an (α, β)-approximation for the k-segment mean of P for $\alpha = \log_2 n$ and $\beta = O(\log n)$. Theorem 2 then follows by substituting α and β in Theorem 1. □

4.7 Efficient k-segmentation via Coresets

We recall one of the useful aims for a k-segment mean coreset is fast k-segmentation. When computing an optimal k-segmentation for our data, we are bounded by the scale of the data in yet another aspect – the number of possible locations for each segment endpoint is $O(N)$. This means we cannot run algorithms with a linear complexity in the data size, let alone a quadratic one, as the original method of [14]. While the coreset we propose handles gracefully k-segmentations with any endpoints, sweeping through all possible endpoints as in the original algorithm by Bellman [14] is computationally prohibitive. In order to avoid the problems associated with such a limitation, we propose two approaches. Alternatively, the approach presented in Subsection 4.7.2 incorporates per coreset segment an weak coreset in the form of a weighted set of points. Computing the cost using these points for shattered coreset segments allows us to bound the error with respect to the k-segment mean computed with the full set of possible endpoints. In Subsection 4.7.1 we show the endpoint-limited case, where we limit ourself to a subset of the possible endpoint based on the coreset segments. In the endpoint limited case, we use the same dynamic programming framework suggested by Bellman in [14], and as demonstrated in Section 5.1, obtain state-of-the-art results on signals from various domains.

4.7.1 Endpoint-constrained k-Segment Mean Computation

One approach for fast segmentation uses the coreset from Algorithm 1 directly, while constraining the possible segment endpoints. We add the additional constraints that there cannot be more than one k-segment endpoint inside each coreset-segment. This allows us to use the coreset obtained by Algorithm 1 for cost computations, and perform the computation of all linear segment costs required in [14] on a sublinear number of sampling points, reducing overall algorithm complexity from $O(kN^2)$ to $O(k^3 log^2 N)$. By construction of the piecewise coresets, and the segments C_i computed in Algorithm 1, the cost computed with these limitations on the endpoints is an ε approximation of the cost of our solution on the real data. Specifically, our solution

Algorithm 4 MODIFIEDBELLMAN(D)

Input: D, a set that satisfies Lemma 1.
Output: f, an ε-approximation of the constrained k-segment mean

1: **for** $b = 1, 2, \ldots, m$ **do**
2: Update the 1-segment solution for each subsegment starting at $t = 1$
3: $f_1(b) = h(1, b)$
4: **end for**
5: **for** $k' = 1, 2, \ldots, K$ **do**
6: **for** $b = 1, 2, \ldots, m$ **do**
7: **for** $u_{k'} = 1, 2, \ldots, b$ **do**
8: Update the cost $f_{k'}$ based on the $(k'-1)$-segment solution $f_{k'-1}$
9: Update f, the k'-segment solution by
10: $f_{k'}(b) = \min_{1 \geq u_{k'} \geq b} [h(u_{k'}, b) + f_{k'-1}(u_{k'})]$, where
$h(u_{k'}, b)$ is computed using the appropriate matrix $C_{(u_{k'}, b)}$.
11: **end for**
12: **end for**
13: **end for**

is an ε-approximation to the real optimal solution from among those of constrained end-points. We note the cost difference between the constrained and unconstrained solutions can be bounded using the Lipschitz constant of the signal.

The modifications required to the algorithm of [14] in this case are as follows

- During the search over $u_{k'}$, $u_{k'}$ is allowed only to be at locations which are part of the piecewise coreset of some segment in D.
- For each line segment $(u_{k'}, b)$, its fitting solution and cost is obtained by concatenating row-wise the matrices C_i from each segment i of D completely contained inside $(u_{k'}, b)$, along with the sampling points inside $(u_{k'}, b)$ from partially contained segments of D, into a single matrix $C_{(u_{k'}, b)}$, and solving for the linear segment using $C_{(u_{k'}, b)}$.
- $h(u_{k'}, u_{k'})$ is defined to be infinite if two segment endpoints are inside a coreset segment.

The Modified Bellman algorithm for computing k-segmentation using a coreset is described as Algorithm 4.

Let $L_{coreset}$ denote the maximum number of inner-points per segment obtained from Algorithm 5. The number of segment fitting cost computations done is

$$O\left(\left(\frac{(k\alpha)}{\varepsilon^2}+\beta k\right)^2 L_{coreset}^2 k\right). \tag{4.13}$$

We denote the minimal fitting k-segment such that its endpoints are constrained to be on the boundaries of one of the coreset segments produced by Algorithm 1.

Theorem 3. *Given a coreset D as described in Algorithm 1, Algorithm 4 finds an ε-approximation of the constrained k-segment mean in time $O(\text{polylog}(n)\,\text{poly}(k))$*

Proof. Computation time is determined by the number of end points over the whole signal. Following the proof from [14], the modified algorithm finds the optimal constrained k-segment.

According to Equation 4.11, we have overall $O(\left(\frac{(k\alpha)}{\varepsilon^2}+\beta k\right))$ segment endpoints, which is $O(\log n)$. Therefore our algorithm requires $O\left((\log n)^2 k\right)$ estimations of linear segment fittings. Each line segment estimation involves constructing a matrix composed out of $O\left(\frac{(k\alpha)}{\varepsilon^2}+\beta k\right)$ complete segments, and inverting it. We note that each segment (partial or full) contributes $O(\log n)$ rows to the matrix, and that its width is $O(d)$. Hence, inverting it is $O(\text{polylog}(n)\,\text{poly}(k))$, therefore the algorithm takes $O(\text{polylog}(n)\,\text{poly}(k))$ to complete. The approximation property of the algorithms comes from the approximation of the coresets □

Incorporating the construction time of the coreset, we have the following.

Theorem 4. *Let P be a d-dimensional signal. A $(1+\varepsilon)$ approximation to the k-segment mean of P can be computed in $O\left(ndk/\varepsilon + d(klog(n)/\varepsilon)^{O(1)})\right)$ time .*

4.7.2 Weak (k,ε)-Coreset for Efficient Segmentation

As an alternative that avoids constraints on the segment endpoints, we describe an additional new approximation tool for computing efficient k-segmentation. Instead of going through all n time points of the signal for the $k+1$-segments induction

Algorithm 5 PIECEWISECORESET(n, s, ε)

Input: An integer $n \geq 1$, a function $s : [n] \to (0, \infty)$ and an error parameter $\varepsilon > 0$.
Output: A vector $w = (w_1, \cdots, w_n)$ that satisfies Lemma 2.

Set $t \leftarrow \sum_{j=1}^{n} s_j$ and $B \leftarrow \emptyset$
for $i = 1$ to n **do**
  Set $b_i \leftarrow \left\lceil \frac{\sum_{j=1}^{i} s_i}{\varepsilon t} \right\rceil$ *//Hence, $b_i \leq \lceil 1/\varepsilon \rceil$*
  if $b_i \notin \{b_j \mid j \in B\}$ **then**
    $B \leftarrow B \cup \{i\}$
  end if
end for
for each $j \in B$ **do**
  Set $I_j \leftarrow \{i \in [n] \mid b_i = b_j\}$
end for
$w_j \leftarrow \begin{cases} \frac{1}{s_j} \sum_{i \in I_j} s_i & j \in B \\ 0 & \text{otherwise.} \end{cases}$
return $(w_1, \cdots, w_n)$

step, or constrain the minimization problem as in Subsection 4.7.1, we propose an approximate set of points to describe the time domain of each coreset segment that is shattered into 2 or more segments. Aggregating the points over all coreset segments provides us with a set of control points over which we can, for example, run the original dynamic algorithm of [14] with bounded error as we now describe.

For an integer $n \geq 1$ we denote $[n] = \{1, \cdots, n\}$. Let $k, n \geq 1$ be a pair of integers. A function $f : \mathbb{R} \to [0, \infty)$ is *non-decreasing* over $[n]$ if $f(i) \leq f(j)$ for every $i \leq j$ in $[n]$, and *non-increasing* if $f(i) \geq f(j)$ for every $i \leq j$ in $[n]$. A function is *monotonic* if it is either non-increasing or non-decreasing. A function $g : \mathbb{R} \to [0, \infty)$ is *k-piecewise monotonic* if $[n]$ can be partitioned into k consecutive sub-intervals $[n] = [i_1] \cup ([i_2] \setminus [i_1]) \cdots \cup ([n] \setminus [i_{k-1}])$ such that g is monotonic over each one of them.

Lemma 2. *Let $k, n \geq 1$ be a pair of integers, $\varepsilon > 0$ and let $f, s : [n] \to (0, \infty)$ be a pair of functions, such that f, s are k-piecewise rational functions, where each interval is of the form $s_i = (a_j + ib_j)^{d_j}$, for each point i in segment j. (d_j can be any integer. Note: this includes k-segments) Let $w = (w_1, \cdots, w_n) \in \mathbb{R}^n$ denote the output of a call to* PIECEWISECORESET$(n, s, \varepsilon/(2\tilde{c}k \sum_{i=1}^{n} s_i))$*; see Algorithm 5. If for*

every $i \in [n]$

$$f(i) \leq s_i \sum_{j=1}^{n} f(j) \tag{4.14}$$

then

$$(1-\varepsilon)\sum_{i=1}^{n} f(i) \leq \sum_{i=1}^{n} w_i f(i) \leq (1+\varepsilon)\sum_{i=1}^{n} f(i).$$

Proof. For every $i \in [n]$ let

$$h_i = \frac{f(i)}{s_i \sum_{j=1}^{n} f(j)}.$$

We will prove that for a vector w that is returned from a call to PIECEWISECORESET(n, s, ε) we have

$$\left| \sum_i \frac{s_i}{t} \cdot h_i - \sum_{j \in B} \frac{w_j s_j}{t} \cdot h_j \right| \leq 2\tilde{c}\varepsilon k, \tag{4.15}$$

Multiplying this by $t \sum_{i=1}^{n} f(i)$ yields

$$\left| \sum_{i=1}^{n} f(i) - \sum_{j \in B} w_j f(j) \right| = \left| \sum_{i=1}^{n} f(i) - \sum_{j=1}^{n} w_j f(j) \right| \leq 2\tilde{c}kt\varepsilon \sum_i f(i).$$

Replacing ε by $\varepsilon/(16kt)$ will prove Lemma 2.

Due to the definitions of f,s, h is $\tilde{c}k$-monotonic, where $\tilde{c}$ is a finite small integer constant depending on the maximum $|d_j|$. This comes from the $2k$ transitions of segments from f, s, and the number of poles/zeros for the derivative of the product function $f_i s_i$. Hence, there is a partition $\Pi = \{[i_1], [i_2] \setminus [i_1], \cdots, [n] \setminus [i_{\tilde{c}k-1}]\}$ of $[n]$ into consecutive $\tilde{c}k$ intervals such that h_i is monotonic over each of these intervals.

Let $I_j = \{i \in [n] : b_i = b_j\}$ for every $j \in B$. For every $I \in \Pi$ we define Good(I) =

$\{j \in B \mid I_j \subseteq I\}$. Their union is denoted by $\text{Good} = \sum_{I\in\Pi} \text{Good}(I)$. Hence,

$$\left|\sum_{i\in[n]} \frac{s_i}{t}\cdot h_i - \sum_{j\in B}\frac{w_j s_j}{t}\cdot h_j\right| = \left|\sum_{i\in[n]} \frac{s_i}{t}\cdot h_i - \sum_{j\in B}\frac{\sum_{i\in I_j} s_i}{t}\cdot h_j\right| = \sum_{j\in B}\sum_{i\in I_j}\frac{s_i}{t}\cdot(h_i - h_j)$$

$$\leq \left|\sum_{j\in B\setminus\text{Good}}\sum_{i\in I_j}\frac{s_i}{t}\cdot(h_i-h_j)\right| \tag{4.16}$$

$$+\sum_{I\in\Pi}\left|\sum_{j\in\text{Good}(I)}\sum_{i\in I_j}\frac{s_i}{t}\cdot(h_i-h_j)\right|. \tag{4.17}$$

We now bound (4.16) and (4.17). Put $j \in B$. By Line 5 of the algorithm we have $|I_j \cap B| = 1$ and $\sum_{i\in I_j} s_i/t \leq \varepsilon$. Hence,

$$\left|\sum_{i\in I_j}\frac{s_i}{t}\cdot(h_i-h_j)\right| \leq \varepsilon(\max_{i\in I_j} h_i - \min_{i\in I_j} h_i) \leq \varepsilon, \tag{4.18}$$

where the last inequality holds since $h_i \leq 1$ for every $i \in [n]$, by (4.14). Since each set $I \in \Pi$ contains consecutive numbers, we have $|B\setminus\text{Good}| \leq 2k$. Using this and (4.18), we bound (4.16) by

$$\left|\sum_{j\in B\setminus\text{Good}}\sum_{i\in I_j}\frac{s_i}{t}\cdot(h_i-h_j)\right| \leq |B\setminus\text{Good}|\cdot\varepsilon \leq |\Pi|\varepsilon \leq \tilde{c}\varepsilon k. \tag{4.19}$$

Put $I \in \Pi$ and denote the numbers in $\text{Good}(I)$ by $\text{Good}(I) = \{k, k+1, \cdots, l\}$. Recall that h is monotonic on I. Without loss of generality, assume that h is non-decreasing on I. Therefore, summing (4.18) over $\text{Good}(I)$ yields

$$\left|\sum_{j\in\text{Good}(I)}\sum_{i\in I_j}\frac{s_i}{t}(h_i-h_j)\right| \leq \sum_{j=k}^{l}\left|\sum_{i\in I_j}\frac{s_i}{t}(h_i-h_j)\right| \leq \sum_{j=k}^{l}\varepsilon(\max_{i\in I_j} h_i - \min_{i\in I_j} h_i)$$

$$\leq \varepsilon\sum_{j=k}^{l-1}(\min_{i\in I_{j+1}} h_i - \min_{i\in I_j} h_i) = \varepsilon(\min_{i\in I_l} h_i - \min_{i\in I_1} h_i) \leq \varepsilon,$$

where in the last derivation we used the fact that $h_i \leq 1$ for every $i \in [n]$. Summing

over every $I \in \Pi$ bounds (4.17) as,

$$\sum_{I \in \Pi} \left| \sum_{j \in \text{Good}(I)} \sum_{i \in I_j} \frac{s_i}{t} \cdot (h_i - h_j) \right| \leq |\Pi| \cdot \varepsilon \leq \tilde{c}\varepsilon k.$$

Plugging (4.19) and the last inequality in (4.16) and (4.17) respectively proves (4.15) as

$$\left| \sum_{i \in [n]} \frac{s_i}{t} \cdot h_i - \sum_{j \in B} \frac{w_j s_j}{t} h_j \right| \leq 2\tilde{c}\varepsilon k$$

□

For every $p, q \in \mathbb{R}^d$ we denote $D(p,q) = \|p - q\|^2$, where $\|p - q\|$ is the Euclidean distance between p and q.

Lemma 3. *Let $p_1, \cdots, p_n$ be a set of points on a line in $\mathbb{R}^d$ such that $\|p_1 - p_2\| = \cdots = \|p_{n-1} - p_n\| = \Delta$ for some $\Delta \geq 0$ and the first coordinate of p_i is i for every $i \in [n]$. Let $l : \mathbb{R} \to \mathbb{R}^d$ be a function such that $\{(x, l(x)) \mid x \in \mathbb{R}\}$ is a line in $\mathbb{R}^{d+1}$. Then for every $i \in [n]$*

$$\|p_i - l(i)\|_2^2 \leq \frac{4 \sum_{j \in [i]} \|p_i - l(i)\|_2^2}{i}.$$

Proof. Since P is contained in a line, it can be shown [45] that there is a point $q \in \mathbb{R}^d$ and a positive number $w > 0$ such that for every $i \in [n]$

$$\|p_i - l(i)\|_2 = w \|p_i - q\|_2 . \tag{4.20}$$

Let $\tilde{D} : [0, \infty) \to [0, \infty)$ be a monotone non-decreasing function and $r \in [0, \infty)$ such that $D(xe^{\delta}) \leq e^{r\delta} D(x)$ for every $x, \delta > 0$. It can be shown that for $\rho = \max\{2^{r-1}, 1\}$ and every $a, b, c \in M$ in a metric space (M, dist) we have

$$\tilde{D}(\text{dist}(a,c)) \leq \rho(\tilde{D}(\text{dist}(a,b)) + \tilde{D}(\text{dist}(b,c)));$$

See [52], Lemma 2.1. In particular, for the case $M = \mathbb{R}^d$, $\text{dist}(a,b) = w\|a - b\|$, we

denote $D(a,b) = \tilde{D}(w\,\|a-b\|)$ to obtain

$$D(a,c) \le \rho(D(a,b) + D(b,c)). \tag{4.21}$$

Let $m = \frac{1}{i}\sum_{j\in[i]} D(p_j, q)$ and $i \in [n]$. We will prove that

$$D(p_i, q) \le 4m\rho^2. \tag{4.22}$$

In particular, for $\tilde{D}(x) = x^2$ we have $r = 2$, $\rho = 1$ and

$$\begin{aligned} \|p_i - l(i)\|_2^2 = \tilde{D}(\|p_i - l(i)\|) = \tilde{D}(w\,\|p_i - q\|) = D(p_i, q) \\ \le 4m = \frac{4\sum_{j\in[i]} \|p_i - l(i)\|_2^2}{i}, \end{aligned} \tag{4.23}$$

where the second equality is by (4.20), and (4.23) is by (4.22).

Indeed, let $Q = \{j \in [i] \mid D(p_j, q) \le 2m\}$. By Markov's inequality,

$$|Q| \ge \frac{i}{2}. \tag{4.24}$$

Hence, there are $p_s, p_t \in Q$ such that $s - t \ge i/2$. Using this and (4.21)

$$D(p_s, p_t) \le \rho(D(p_s, q) + D(q, p_t)) \le 2\rho m. \tag{4.25}$$

Since $s - t \ge i/2$,

$$\Delta\,\|p_i - p_s\| = \Delta(i - s) \le \Delta(i - i/2) = \Delta i/2 \le \Delta(s - t) = \Delta\,\|p_s - p_t\|\,.$$

Since $\tilde{D}$ is non-decreasing, the last equation implies $D(p_i, p_s) \le D(p_s, p_t)$. Together with (4.25) we get $D(p_i, p_s) \le 2\rho m$. Using the last inequality and the fact that $p_s \in Q$ proves (4.22) as

$$D(p_i, q) \le \rho(D(p_i, p_s) + D(p_s, q)) \le \rho(2\rho m + 2m) \le 4m\rho^2.$$

□

A function $g : \mathbb{R} \to \mathbb{R}^d$ is a 2-*piecewise linear function* if the set $\{(x, g(x)) \mid x \in \mathbb{R}\}$ is the union of two linear segments in $\mathbb{R}^{d+1}$.

Corollary 1. *Let* $(w_1, \cdots, w_n) \in \mathbb{R}^n$ *be the output of a call to* PIECEWISECORESET$(n, s, \frac{c\varepsilon}{\log n})$ *where* c *is a sufficiently small universal constant,* $n \geq 1$, $\varepsilon > 0$ *and* s *is the function that maps every* $i \in [n]$ *to* $s_i = \max\left\{\frac{4}{i}, \frac{4}{n-i+1}\right\}$.

Then for every set $(1, p_1), \cdots, (n, p_n)$ *of* n *points that is contained in a line in* $\mathbb{R}^{d+1}$ *and every 2-piecewise linear function* $g : \mathbb{R} \to \mathbb{R}^d$ *the following hold:*

(i) w *has* $\|w\|_0 = O\left(\frac{\log n}{\varepsilon}\right)$ *non-zeroes entries.*

(ii) w *can be computed in* $O(\log n) \cdot \|w\|_0$ *time and* $\|w\|_0$ *space.*

(iii) The following bound holds:

$$(1-\varepsilon)\sum_{i=1}^{n} \|g(i) - p_i\|^2 \leq \sum_{i=1}^{n} w_i^2 \|g(i) - p_i\|^2 \leq (1+\varepsilon)\sum_{i=1}^{n} \|g(i) - p_i\|^2 .$$

Proof. (i) Put $\varepsilon' = c\varepsilon \log n$. By Line 11 of the algorithm, $\|w\|_0 = |B|$. Since B consists of distinct integers $b_i \in [1, 1/\varepsilon' + 1]$ we have $\|w\|_0 = |B| = O(1/\varepsilon') = O(\log(n)/\varepsilon)$. *(ii)* Since b_i is monotonic over $i \in [n]$, we can use binary search on $[n]$ to compute the smallest $i \in [n]$ such that $b_i \notin B$. In each of the $O(\log n)$ iterations we compute b_j for some $j \in [n]$. Since $\sum_{j=1}^{i} s_i$ is a sum of two harmonic series, b_j can be computed in $O(1)$ time. As explained in (i), $|B| = O(\log(n)/\varepsilon)$ so the overall time is $O(\log(n)/\varepsilon') = O(\log^2 n/\varepsilon)$. We only need to store w during this recursion, which takes $\|w\|_0$ space. *(iii)* Put $i \in [n]$ and let $f(i) = \|p_i - g(i)\|^2$. Since $(1, p_1), \cdots, (n, p_n)$ are on a line, we have that $\|p_1 - p_2\| = \cdots = \|p_{n-1} - p_n\| = \Delta$ for some $\Delta \geq 0$. Since g is 2-piecewise linear function, there is a line $\{x, l(x)\}$ for some $l : \mathbb{R} \to \mathbb{R}^d$ such that $l(j) = g(j)$ for every $j \in [i]$ or every $j \in \{i, i+1, \cdots, n\}$. Without loss of generality, we assume the

first case. By Lemma 3,

$$f(i) = \|p_i - g(i)\|_2^2 = \|p_i - l(i)\|_2^2 \leq \frac{4\sum_{j\in[i]} \|p_i - l(i)\|_2^2}{i} \leq s_i \sum_{j\in[i]} \|p_i - l(i)\|_2^2 = s_i \sum_{j\in[i]} f(j). \tag{4.26}$$

Since g is 2-piecewise linear and $p_1, \cdots, p_n$ are points on a line, we have that f is 4-monotonic over $[n]$. The function s is 2-monotonic. We also have that

$$\frac{c\varepsilon}{\log n} \leq \frac{\varepsilon}{2\tilde{c}k\sum_{i=1}^{n} s_i}$$

for a sufficiently small constant c. Plugging this and (4.26) in Lemma 2 then proves the theorem as

$$(1-\varepsilon)\sum_{i=1}^{n} f(i) \leq \sum_{i=1}^{n} w_i f(i) \leq (1+\varepsilon)\sum_{i=1}^{n} f(i).$$

□

We show how to compute a $(1+\varepsilon)$-approximation to the k-segment mean of the original signal P using the coresets constructed in Algorithms 3 and 5. The technique can be used to solve any other optimization problem over k-segments, assuming that we have an existing algorithm for a weighted signal. For example, if priors are given (weights for each segment) or we want to minimize the cost over some subset of k-segments (e.g., (k,m)-segment mean).

We assume that we are given a possibly inefficient algorithm SLOWSEGMENTATION ("black box") that will be used to compute the k-segment mean of a small set that is based on the coreset. The algorithm SLOWSEGMENTATION gets a set Q of pairs $((t,p),w)$ where $t \in \mathbb{R}$, (t,p) is a point in $\mathbb{R}^{d+1}$, and $w > 0$ denote its weight. The algorithm then returns the k-segment mean of Q, i.e., the k-piecewise linear function that minimizes the *weighted cost*, $\text{cost}_W(Q,f) \leftarrow \sum_{((t,p),w)\in Q} w\|(p - f(t))\|^2$. We will run this algorithm only on a small set Q, whose size is roughly the size of the coreset. In what follows we describe the algorithm FASTSEGMENTATION that uses the coreset and SLOWSEGMENTATION to get a fast approximation of the k-segment mean of the

Algorithm 6 FASTSEGMENTATION$(P, k, \varepsilon, \mathrm{S})$

Input: $P = \{(1, p_1), \cdots, (n, p_n)\}$ in $\mathbb{R}^{d+1}$
Input: $k \geq 1$, an integer
Input: $\varepsilon > 0$, an error parameter
Input: S = SLOWSEGMENTATION(Q, k), an algorithm that computes the k-segment mean of a given weighted set Q.
Output: f, a $(1+\varepsilon)$-approximation f to the k-segment mean of P.

1: $D \leftarrow$ CORESET(P, k, ε); See Algorithm 3.
2: Identify $D = \{(C_1, g_1, b_1, e_1), .., (C_m, g_m, b_m, e_m)\}$
3: $Q \leftarrow \emptyset$
4: **for** $i \leftarrow 1$ to m **do**
5: $\quad P_i \leftarrow \{(b_i, g_i(b_i)), \cdots, (e_i, g_i(e_i))\}$
6: $\quad (w_1, \cdots, w_n) \leftarrow$ PIECEWISECORESET$(|P_i|, s, c\varepsilon/\log(n))$, where c and s are defined in Corollary 1.
7: $\quad Q \leftarrow Q \cup \left\{(t, p), w_j^2) \mid (t, p)\right.$ is the jth point of the signal $\left.P_i\right\}$
8: **end for**
9: $h \leftarrow$ SLOWSEGMENTATION(Q, k)
10: **for** $i \leftarrow 1$ to m **do**
11: $\quad T_i \leftarrow \{b_i, \cdots, e_i\}$
12: $\quad$ **if** $\{h(t) \mid t \in T_i\}$ consists of at most 2-segments **then**
13: $\quad\quad f(t) \leftarrow h(t)$ for every $t \in T_i$
14: $\quad$ **else**
15: $\quad\quad f(t) \leftarrow g_i(t)$ for every $t \in T_i$
16: $\quad$ **end if**
17: **end for**
18: **return** f

original set P.

Algorithm overview The input to the algorithm FASTSEGMENTATION is a signal P of n points in $\mathbb{R}^d$, an error parameter $\varepsilon > 0$, and an integer $k \geq 1$. In addition, the algorithm gets a pointer to the algorithm SLOWSEGMENTATION.

Recall that for a k-segment $f : \mathbb{R} \to \mathbb{R}^d$ and $i \in [m]$ we say that C_i is served by one segment of f if $\{f(t) \mid b_i \leq t \leq e_i\}$ is a linear segment. The next lemma states the weighted set Q that is computed in Line 7 of Algorithm 6 is a weak coreset in the following sense. For every k-segment f such that each cell C_i is served by at most two segments of f, the cost of P and the weighted cost of Q to f are approximately the same. In Theorem 1 we prove that a k-segment mean has this property, and thus

can be computed from this coreset Q.

Lemma 4 (Weak coreset). *Let f be a k-segment such that C_i is served by at most two segments of f, for every $i \in [m]$. Then*

$$\min_f \text{cost}(P, f) \leq \min_f \text{cost}_W(Q, f) \leq (1+\varepsilon) \min_f \text{cost}(P, f).$$

Proof. Put $i \in [m]$ and $P_i = \{(t_1, p_1), \cdots, (t_{|P_i|}, p_{|P_i|})\}$. Since C_i is served by at most two segments of f, then P_i is also served by at most two segments of f. By Corollary 1,

$$(1-\varepsilon) \sum_{j=1}^{|P_i|} w_j \|f(t_j) - p_j\|^2 \leq \sum_{j=1}^{|P_i|} w_j^2 \|f(t_j) - p_j\|^2 \leq (1+\varepsilon) \sum_{j=1}^{|P_i|} \|f(t_j) - p_j\|^2.$$

Hence, letting $Q_i = \{(t_j, p_j), w_j^2) \mid w_j > 0, j \in [|P_i|]\}$, by Line 7 of Algorithm 6 we obtain

$$\begin{aligned} |\text{cost}(P_i, f) - \text{cost}_W(Q_i, f)| &= \left| \sum_{j=1}^{|P_i|} \|f(t_j) - p_j\|^2 - \sum_{j=1}^{|P_i|} w_j^2 \|f(t_j) - p_j\|^2 \right| \\ &\leq \varepsilon \sum_{j=1}^{|P_i|} \|f(t_j) - p_j\|^2 = \varepsilon \text{cost}(P_i, f). \end{aligned}$$

Summing over every $i \in [m]$ yields

$$|\text{cost}(P, f) - \text{cost}_W(Q, f)| \leq \varepsilon \text{cost}(P, f).$$

□

Theorem 5. *Let $P = \{(1, p_1), \cdots, (n, p_n)\}$ be a set in $\mathbb{R}^{d+1}$, $\varepsilon \in (0, 1/2)$, and $k \geq 1$ be an integer. Let $f : \mathbb{R} \to \mathbb{R}^d$ be the output of a call to* FastSegmentation$(P, k, \varepsilon,$ SegAlg$)$. *Then f is a $(1+\varepsilon)$-approximation to the k-segment mean of P, i.e.,*

$$\text{cost}(P, f) \leq (1+\varepsilon) \min_{f'} \text{cost}(P, f'),$$

where the minimum is over every k-segment $g : \mathbb{R} \to \mathbb{R}^d$.

Proof. Let h be the k-segment that is computed in Line 9 of the algorithm FASTSEGMENTATION(P, k) and let $i \in [m]$. We first prove that $\text{cost}(P_i, f) \leq \text{cost}(P_i, h)$ by case analysis: (i) $f(t) = h_i(t)$ for every $t \in T_i$, and (ii) $f(t) = g_i(t)$ for every $t \in T_i$.
Case (i): In this case $\text{cost}(P_i, f) = \text{cost}(P_i, h)$ by definition of P_i.
Case (ii): In this case $\text{cost}(P_i, f) = \text{cost}(P_i, g_i)$. By its construction, g_i is a 2-approximation for the 1-segment mean of P_i. Since the points of P_i lie on a line, we thus have $\text{cost}(P_i, g_i) = 0$. Hence,

$$\text{cost}(P_i, f) = \text{cost}(P_i, g_i) = 0 \leq \text{cost}(P_i, h).$$

Summing $\text{cost}(P_i, f) \leq \text{cost}(P_i, h)$ over $i \in [m]$ yields

$$\text{cost}(P, f) \leq \text{cost}(P, h). \tag{4.27}$$

Suppose that h^* minimizes $\text{cost}(P, f')$ over every k-segment $f' : \mathbb{R} \to \mathbb{R}^d$. Similarly to (4.27), it can be shown that there is a k-segment f^* such that

$$\text{cost}(P, f^*) \leq \text{cost}(P, h^*),$$

and C_i is served by at most two segments of f^*, for every $i \in [m]$. We then have

$$\begin{aligned}
\text{cost}(P, f) &\leq \frac{\text{cost}_W(Q, f)}{1+\varepsilon} && (4.28)\\
&\leq \frac{\text{cost}_W(Q, h)}{1+\varepsilon} && (4.29)\\
&\leq \frac{\text{cost}_W(Q, f^*)}{1+\varepsilon} && (4.30)\\
&\leq \frac{(1-\varepsilon)\text{cost}(P, f^*)}{1+\varepsilon} && (4.31)\\
&\leq (1+10\varepsilon)\text{cost}(P, f^*), && (4.32)
\end{aligned}$$

where (4.29) holds by (4.27), Eq. (4.28) and (4.31) hold by Lemma 4, Eq. (4.30) is by the optimality of h, and (4.32) holds since $\varepsilon \leq 1/2$. Replacing ε with $\varepsilon/10$ proves

this theorem. □

4.8 Parallel and Streaming Implementation

One major advantage of coresets is that they can be constructed in parallel as well as in a streaming setting. The main observation is that the union of coresets is a coreset — if a data set is split into subsets, and we compute a coreset for every subset, then the union of the coresets is a coreset of the whole data set. This allows us to have each machine separately compute a coreset for a part of the data, with a central node which approximately solves the optimization problem; see [49, Theorem 10.1] for more details and a formal proof.

When discussing streaming coresets, one must define the merging and reduction operations used in streaming, and show that the coresets create are still efficient and accurate. We build our merge and reduce operations as a modification of the coreset algorithm as given in Algorithm 3, so that it compacts coreset segments rather than signal points. For this we modify Algorithms 1,2 as we now describe.

First, we look at Algorithm 2, we modify it in the following way. In line 6 of the algorithm, the original segments from both child coresets are taken. Partitioning is done by unifying existing coreset segments into sections P_i. Iterating over Theorem 1, we note that part i is kept by the reduction of parts at each turn. Looking at the proof of part ii we note that we only use the coreset segments' cost as represented for 1-segments, and this can be computed by the C matrices, starting from the end of Equation 4.12. This holds also for the joined and compacted matrices.

Next, we look at Algorithm 1, and modify it to utilize the child coresets' coreset segments in order to construct a new set of coreset segments for the combined span. This requires several modification the the algorithm – notibly, the accumulation of a new coreset segment Q is done solely in terms of adding new child coreset segments. We note that f^* and λ in lines 5 and 6 respectively can be computed for concatenations of coreset segments, in terms of their $(1, \varepsilon/4)$-coresets. We note that C,g can be computed using the C matrices of the child coresets. We do so by concatenating the

C child matrices, and recomputing the SVD for the concatenated matrix.

Specifically, for the matrix-based approach (of claim 1), let (U_1, S_1, V_1) and (U_2, S_2, V_2) be the SVD of matrices $P_1, P, 2$ corresponding to the coreset segments creation. It's easy to show that

$$\left\| (S_1 V_1^T) \begin{pmatrix} a \\ b \\ -I \end{pmatrix} \right\|_F^2 + \left\| (S_2 V_2^T) \begin{pmatrix} a \\ b \\ -I \end{pmatrix} \right\|_F^2 =$$
$$\left\| \begin{pmatrix} S_1 V_1^T \\ S_2 V_2^T \end{pmatrix} \begin{pmatrix} a \\ b \\ -I \end{pmatrix} \right\|_F^2 = \left\| U_J^T \begin{pmatrix} S_1 V_1^T \\ S_2 V_2^T \end{pmatrix} \begin{pmatrix} a \\ b \\ -I \end{pmatrix} \right\|_F^2 = \tag{4.33}$$
$$\left\| S_J V_J^T \begin{pmatrix} a \\ b \\ -I \end{pmatrix} \right\|_F^2,$$

where (U_J, S_J, V_J) is the SVD of $\begin{pmatrix} S_1 V_1^T \\ S_2 V_2^T \end{pmatrix}$, and we used the properties of the Frobenius norm, the isometry properties of unitary matrices, and properties of (U_J, S_J, V_J), respectively. Once C_J is computed g_J can be computed easily. We note that similar to Theorem 6, an approximate solution can be computed using the approach of [57].

Looking at the proof of Lemma 1, we note that the treatment of good coreset segments remains the same. The coreset segments that do not belong to $Good(D, f)$ still amount to the same cost bounds, due to the construction of g_J.

4.8.1 1-Segment Coreset

A $(1, \varepsilon)$-coreset approximates $\text{cost}(P, f)$ for every 1-segment f up to a factor of $1 \pm \varepsilon$,

Definition 6 ($(1, \varepsilon)$-coreset). *Let P and C be two sets in $\mathbb{R}^{d+1}$ and let $\varepsilon, w > 0$. The*

Algorithm 7 1-SEGMENTCORESET(P)

Input: A set $P = \{(t_1, p_1), \cdots, (t_n, p_n)\}$ in $\mathbb{R}^{d+1}$.
Output: $(1, 0)$-coreset (C, w) that satisfies Claim 1.

1: Set $X \in \mathbb{R}^{n \times (d+2)}$ to be matrix whose ith row is $(1, t_i, p_i)$ for every $i \in [n]$.
2: Compute the thin SVD $X = UDV^T$ of X.
3: Set $u \in \mathbb{R}^{d+2}$ to be the leftmost column of DV^T.
4: Set $w \leftarrow \frac{\|u\|^2}{d+2}$. *//$w > 0$ since $\|D\| = \|X\| > 0$*
5: Set $Q, Y \in \mathbb{R}^{(d+2)\times(d+2)}$ to be unitary matrices whose leftmost columns are $u/\|u\|$ and $(\sqrt{w}, \cdots, \sqrt{w})/\|u\|$ respectively.
6: Set $B \in \mathbb{R}^{(d+2)\times(d+1)}$ to be the $(d+1)$ rightmost columns of $YQ^TDV^T/\sqrt{w}$.
7: Set $C \subseteq \mathbb{R}^{d+1}$ to be the union of the rows in B
8: **return** (C, w).

pair (C, w) is a $(1, \varepsilon)$-coreset *for P, if for every 1-segment $f : \mathbb{R} \to \mathbb{R}^d$ we have*

$$(1 - \varepsilon)\text{cost}(P, f) \leq w \cdot \text{cost}(C, f) \leq (1 + \varepsilon)\text{cost}(P, f).$$

For example, (P, w) is a $(1, \varepsilon)$-coreset for P with $\varepsilon = 0$ and $w = 1$. However, a coreset is efficient if its size $|C|$ is much smaller than P.

It is easy to compute $\text{cost}(P, f)$ exactly by a matrix DV^T of $(d+2)$ rows using SVD, as shown in Algorithm 7. In our coreset construction we use additional matrices Q and Y to turn this matrix into a subset C of $\mathbb{R}^{d+1}$ so that the cost $\text{cost}(P, f) = \text{cost}(C, f)$ is still a point-wise cost, although a weighted one. This allows us to improve the result later in this section, to get a less trivial coreset C of only $O(1/\varepsilon^2)$ rows.

Claim 1. *Let P be a set of n points in $\mathbb{R}^{d+1}$. Let (C, w) be the output of a call to* 1-SEGMENTCORESET(P)*; see Algorithm 7. Then (C, w) is a $(1, 0)$-coreset for P of size $|C| = d + 1$. Moreover, C and w can be computed in $O(nd^2)$ time.*

Proof. Let $f : \mathbb{R} \to \mathbb{R}^d$ be a 1-segment. Hence, there are row vectors $a, b \in \mathbb{R}^d$ such that $f(t) = a + bt$, for every $t \in \mathbb{R}$. By definition of Q and Y we have $YQ^Tu/\|u\| = (\sqrt{w}, \cdots, \sqrt{w})^T/\|u\|$. The leftmost column of YQ^TDV^T is thus

$YQ^T u = (\sqrt{w}, \cdots, \sqrt{w})^T$. Therefore,

$$\begin{aligned}
\text{cost}\,(P,f) &= \sum_{(t,p)\in P} \|f(t)-p\|^2 = \sum_{(t,p)\in P} \|a+bt-p\|^2 = \sum_{(t,p)\in P} \left\| \begin{bmatrix} 1 & t \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} - p \right\|^2 \\
&= \left\| X \begin{bmatrix} a \\ b \\ -I \end{bmatrix} \right\|^2 = \left\| UDV^T \begin{bmatrix} a \\ b \\ -I \end{bmatrix} \right\|^2 = \left\| YQ^T DV^T \begin{bmatrix} a \\ b \\ -I \end{bmatrix} \right\|^2 \\
&= \left\| \begin{bmatrix} \sqrt{w} & \\ \vdots & \sqrt{w}B \\ \sqrt{w} & \end{bmatrix} \begin{bmatrix} a \\ b \\ -I \end{bmatrix} \right\|^2 = w \left\| \begin{bmatrix} 1 & \\ \vdots & B \\ 1 & \end{bmatrix} \begin{bmatrix} a \\ b \\ -I \end{bmatrix} \right\|^2 \\
&= w \sum_{(t,p)\in B} \|(a+bt-p)\|^2 = w \cdot \text{cost}(C,f).
\end{aligned}$$

Construction Time. The matrices Q and Y can be computed in $O(dn^2)$ time using the QR decomposition of $\begin{bmatrix}(1,\cdots,1)^T & I\end{bmatrix}$ and $\begin{bmatrix}u & I\end{bmatrix}$. Computing the thin SVD of an $n \times d$ matrix X also takes $O(nd^2)$ time. Hence, the overall running time is $O(nd^2)$ [109]. □

The size $d+1$ and running time of the above $(1,0)$-coreset C might be too large, for example when d is in the order of n, or we are dealing with high dimensional space such as images or text. On the other side, in the rest of the work the construction of $(1,\varepsilon)$-coresets suffices. Using recent results from [49] and [57], the following theorem yields faster and smaller coreset constructions when $d \gg 1/\varepsilon$.

Theorem 6. *Let $P \subseteq \mathbb{R}^{d+1}$ and let $\varepsilon > 0$. A $(1,\varepsilon)$-coreset $C \subseteq \mathbb{R}^{d+1}$ for P of size $|C| = O(1/\varepsilon^2)$ can be computed in $O(nd/\varepsilon^4)$ time.*

Proof. It was proven in [49] that a coreset for P and a family of query shapes, where each shape is spanned by $O(1)$ vectors in $\mathbb{R}^d$, can be computed by projecting P on a $(1/\varepsilon^2)$ dimensional subspace S that minimizes the sum of squared distances to P up to a $(1+\varepsilon)$ factor. The resulting coreset approximates the sum of squared distances to every such shape up to a factor of $(1+\varepsilon)$. The size of this coreset is n, the

same as the input size, however the coreset is contained in an $O(1/\varepsilon^2)$ dimensional subspace. We then compute a $(1,0)$-coreset C for this low dimensional set of n points in $s = O(1/\varepsilon^2)$ space using Claim 1. This will take additional $O(ns^2)$ time and the resulting coreset will be of size $O(s)$.

The subspace S can be computed deterministically in $O(nd/\varepsilon^4)$ using a recent result of [57]. □

As proven below, the 1-segment mean of C is an approximation to the 1-segment mean of P. So, using C we can compute a fast approximation for the 1-segment mean of P.

Corollary 2. *Let $\varepsilon \in (0,1)$. A $(1+\varepsilon)$-approximation to the 1-segment of P can be computed in $O(nd/\varepsilon^4)$ time.*

Proof. Using Theorem 6 we compute a $(1,\varepsilon)$-coreset C of size $|C| = O(1/\varepsilon^2)$ in $O(nd/\varepsilon^4)$ time. Then, using the singular value decomposition it is easy to compute a 1-segment mean f of C in $O(d \cdot |C|^2) = O(d/\varepsilon^4)$ time. Hence, the overall running time is $O(nd/\varepsilon^4)$.

Let f^* be a 1-segment mean of P and f be an arbitrary 1-segment. Since C is a $(1,\varepsilon)$-coreset for P,

$$\text{cost}(P,f) \leq (1+\varepsilon)\text{cost}(C,f) \leq (1+\varepsilon)\text{cost}(C,f^*) \leq (1+\varepsilon)^2\text{cost}(P,f^*) \leq (1+3\varepsilon)\text{cost}(P,f^*),$$

where in the last inequality we use the assumption $\varepsilon < 1$. Replacing ε with $\varepsilon/3$ in the above proof proves the corollary. □

In the previous section we showed that a 1-segment coreset (C,w) of size independent of n exists for every signal P. Unfortunately, the next example shows that, in general, for $k \geq 3$ such a coreset C must contain all the n points of P. This result justifies the more complicated definition of a (k,ε)-coreset in the next section; See Definition 3.

Claim 2. *For every integers $n, c, d \geq 1$ there is a set P of n points in $\mathbb{R}^{d+1}$ such that the following holds. If $C \subseteq \mathbb{R}^{d+1}$ and $|C| < n$ then there is a 3-segment f such that*

either

$$\text{cost}(C, f) \geq c \cdot \text{cost}(P, f) \;\; \textit{or} \;\; \text{cost}(P, f) \geq c \cdot \text{cost}(C, f).$$

Proof. Let $P = \{(i, 0, \cdots, 0)\}_{i=1}^{n}$, a constant-0 signal. Consider the 3-segment $f : \mathbb{R} \to \mathbb{R}^d$ such that $f(t) = (0, \cdots, 0)$ for every $t \in \mathbb{R}$. We have $\text{cost}(P, f) = 0$. If $\text{cost}(C, f) > 0$ then $\text{cost}(C, f) \geq c \cdot \text{cost}(P, f)$ as desired.

Otherwise, $\text{cost}(C, f) = 0$. Let $(t, p) \in P \setminus C$ and consider a 3-segment $g : \mathbb{R} \to \mathbb{R}^d$ such that $g(t) = f(t) = (0, \cdots, 0)$ for every $t \in \mathbb{R} \setminus \{t\}$ and $g(t) \neq p$. Hence,

$$\begin{aligned}\text{cost}(C, g) &= \sum_{(t', p') \in C} \|p' - g(t')\|^2 = \sum_{(t', p') \in C \setminus (t, p)} \|p' - g(t')\|^2 \\ &= \sum_{(t', p') \in C} \|p' - f(t')\|^2 = \text{cost}(C, f) = 0.\end{aligned}$$

Since $\text{cost}(P, g) = \|p - g(t)\|^2 > 0$ the last two inequalities imply $\text{cost}(P, g) \geq c \cdot \text{cost}(C, g)$. □

4.9 Conclusions

In this chapter we presented the k-segment coreset, enabling us to build systems for fast, content-based segmentation of data streams. In the next chapter we describe our system implementation and in Section 5.1 we present experimental results to validate our proposed approach on real data.

Chapter 5

Coresets for Segmentation – Applications to Video Streams and Financial Data

In this chapter[1] we present a system for efficient online segmentation of large data streams and detail experimental results. We demonstrate our algorithms on various data types: video streams, GPS, and financial ticker data, in several experimental modalities. We evaluate performance with respect to output size, running time and quality and compare our coresets to uniform and random sampling compression schemes. We demonstrate the effectiveness of our algorithm by running several analysis algorithms on the computed coreset instead of the full data. Our implementation allows real-time segmentation of video streams at 30 frames per second on a single machine. Lastly, we demonstrate the scalability of our algorithm by running our system on an Amazon cluster with 255 machines with near-perfect parallelism as demonstrated on 256,000 frames.

[1]Some of the content in this chapter was published in [114].

5.1 Experimental Results

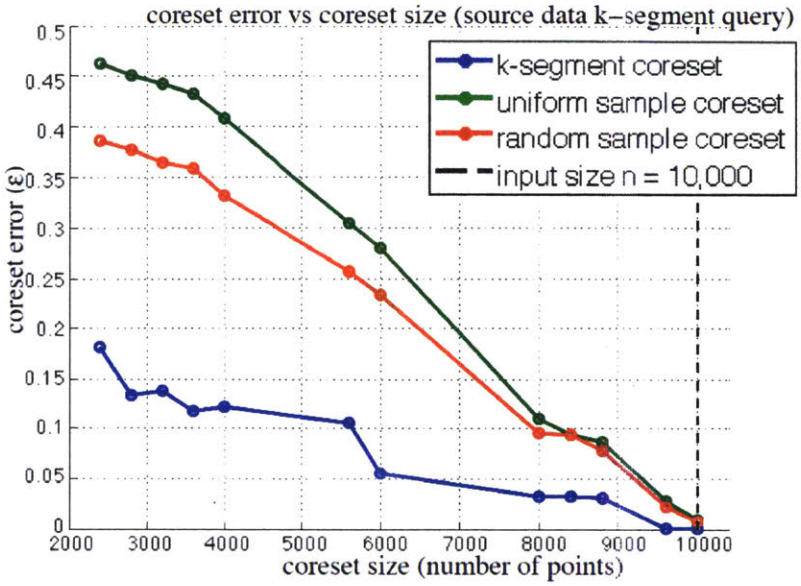
We now demonstrate the results of our algorithm on several data streams of varying length and dimensionality. We compare our algorithms against several other segmentation algorithms where applicable. We also show that the coreset effectively improves the performance of several segmentation algorithms by running the algorithms on our coreset instead of the full data.

5.1.1 Segmentation of Large Datasets

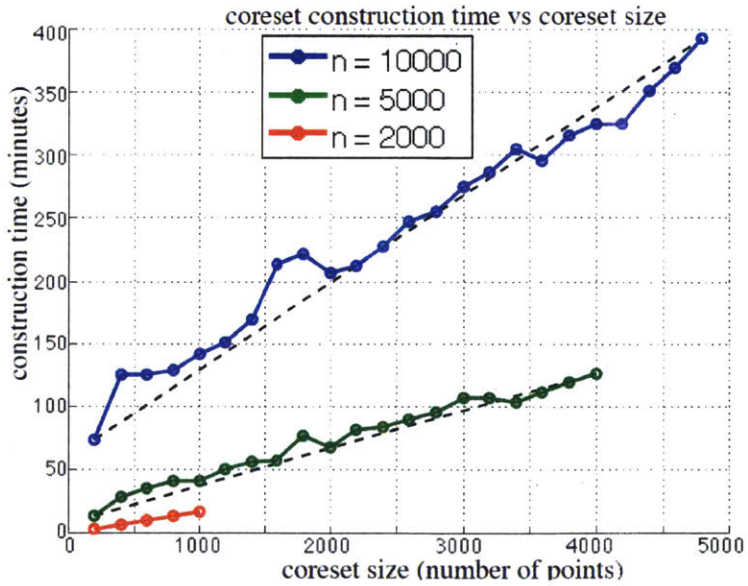
We first examine the behavior of the algorithm on synthetic data which provides us with easy ground-truth, to evaluate the quality of the approximation, as well as the efficiency, and the scalability of the coreset algorithms. We generate synthetic test data by drawing a discrete k-segment P with $k = 20$, and then add Gaussian and salt-and-pepper noise. We then benchmark the computed (k, ε)-coreset D by comparing it against piecewise linear approximations with (1) a uniformly sampled subset of control points U and (2) a randomly placed control points R. For a fair comparison between the (k, ε)-coreset D and the corresponding approximations U, R we allow the same number of coefficients for each approximation. Coresets are evaluated by computing the fitting cost to a query k-segment Q that is constructed based on the a-priori parameters used to generate P.

Approximation Power Figure 5-1a shows the aggregated fitting cost error for 1500 experiments on synthetic data. We varied the assumed k' segment complexity. In the plot we show how well a given k' performed as a guess for the true value of k. As Figure 5-1a shows, we significantly outperform the other schemes. As the coreset size approaches the size P the error decreases to zero as expected. This is in line with the intuition that for an arbitrarily large coreset size we can trivially construct a zero-error coreset by simply taking all of the input points.

Performance We evaluate performance by considering how the coreset size and coreset construction time depend on the input size and dimensionality. Figure 5-

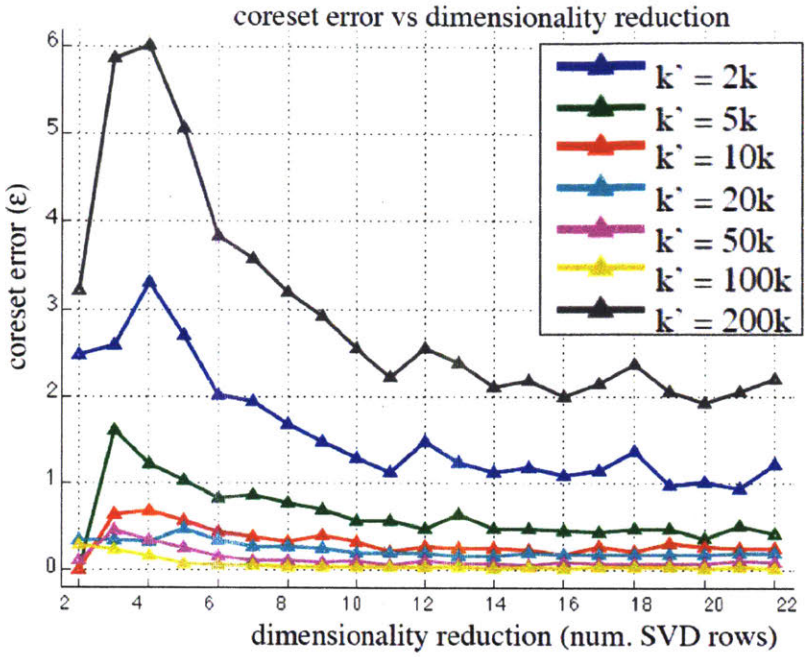


(a) Coreset error (ε) decreasing as a function of coreset size. The dotted black line indicates the point at which the coreset size is equal to the input size.



(b) coreset construction time in minutes as a function of coreset size. Trendlines show the linear increase in construction time with coreset size.

Figure 5-1: Coreset size vs error and construction time



(a) Coreset error as a function of the dimensionality of the 1-segment coreset, for a fixed input size (note that in practice dimensionality is often reduced down to $\mathbb{R}^2$.

Figure 5-2: Coreset error vs dimensionality reduction

1b shows the linear relationship between input size and construction time of D for different coreset size. Figure 5-2 shows how a high dimensionality benefits coreset construction. This is even more apparent in real data which tends to be sparse, so that in practice we are typically able to further reduce the coreset dimension in each segment.

Scalability The coresets presented in this work are parallelizable, as discussed in Section 4.8. We demonstrate scalability by conducting very large scale experiments on both real and synthetic data, running our algorithm on a network of 255 Amazon EC2 nodes. We compress a 256,000-frame *bags-of-words* (BOW) stream in approximately 20 minutes, representing an almost-perfect scalability. For a comparable single node running on the same data dataset, we estimate a total running time of approximately 42 hours.

5.1.2 Real Data Experiments

We compare our coreset against uniform sample and random sample coresets, as well as two other segmentation techniques: Ramer-Douglas-Peucker (RDP) algorithm [111, 38], which uses a maximum distance cost function, and the Dead Reckoning (DR) algorithm [130], which uses a sum of squared distances cost function. We also show that we can combine our coreset with segmentation algorithms, by running the algorithm on the coresets itself. We emphasize that segmentation techniques (RDP, DR) were purposely chosen as simple examples and are not intended to reflect the state of the art – the point is to demonstrate how the k-segment coreset can be used to improve on any given algorithm.

To demonstrate the general applicability of our techniques, we run our algorithm using financial (1D) time series data, as well as GPS data (2D).

For the 2D case we use GPS data from a taxi fleet of 343 taxis in San Francisco. This is of interest because a taxi-route segmentation has an intuitive spacial interpretation that we can easily evaluate, and on the other hand GPS data forms an increasingly large information source which we are interested of analyzing. Figures 5-3a, 5-3b show example results for a single taxi. Again, we observe that computing a DR segmentation produces segments with a meaningful spatial interpretation. Figure 5-5 shows a plot of coreset errors for the first 50 taxis (right), and the table gives a summary of experimental results for the Bitcoin and GPS experiments.

For the 1D case we use Bitcoin price data from the now defunct Mt.Gox Bitcoin exchange. Bitcoin is of general interest because its price has grown exponentially with its popularity in the past two years. Bitcoin has also sustained several well-documented market crashes [13, 25] that we can relate to our analysis. Figure 5-6a shows the results for the Bitcoin data. Notable market crash events are highlighted by local price highs (green) and lows (red). We observe that running the simple DR algorithm on our k-segment coreset to compute a segmentation captures these events quite well. Figure 5-6b shows Bitcoin price segmentation vs Google Trends interest over time for the search term "Bitcoin" [1]. It is interesting to note that the most

prominent search even occurred on April 10, 2013 [13, 25], which the segmentation identified despite the market capitalization being an order of magnitude less than it was a year later.

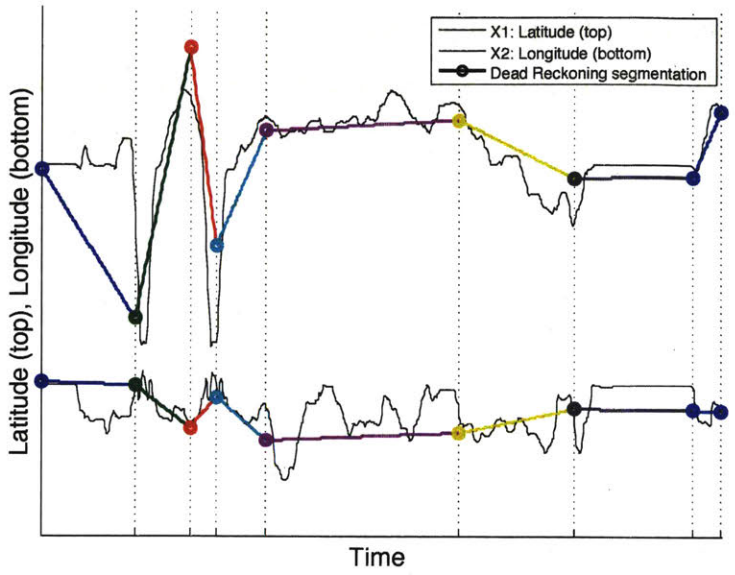
As a high-dimensional temporal stream from the financial domain we look at a stream of S&P 500 stock quotes. Specifically, we take the daily ask price for S&P 500 over 2000 days, between 8/29/2005 and 8/09/2013. Pruning stocks that do not appear in the index for the full duration, we have a 477-dimensional vector per day, normalized w.r.t. the initial stock value. In Figures 5-7–5-9 we demonstrate the coreset computation and segmentation over the period. As can be seen, both the coreset segments, and the resulting dynamic programming segmentation capture main events of the period, such as the October 2008 and August 2011 crashes.

5.1.3 Semantic Video Segmentation

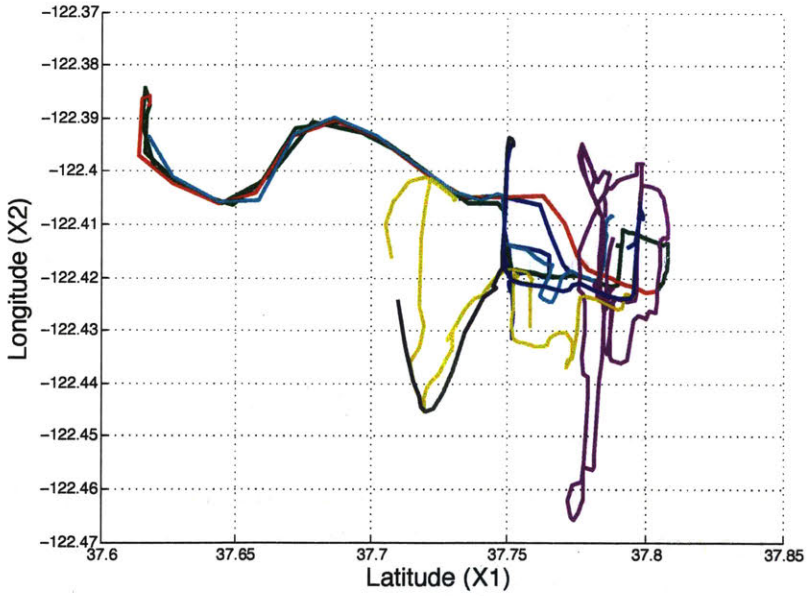
In addition, we demonstrate use of the proposed coreset for video streams summarization and compression. In order to get a meaningful segmentation and analysis of video streams, embedding video frames into a semantically meaningful representation is important. The basis for video segmentation and summarization is content comparison across video frames. The dissimilarity measure is subjective and depends on our own definition of activities and segments in the video, and it may involve user interaction or input. While different choices of frame representations for video summarization are available [124, 86, 91], we used BOWs based on color-augmented SURF features, quantized into 5000 visual words, trained on the ImageNet 2013 dataset [37]. We gather about 3.5×10^8 vectors from the data by an online streaming coreset [49] for k-means representation, allowing the k-means clustering to be computed in a few minutes on an Intel $i7$ CPU.

The resulting signals are compressed in a streaming coreset. Computation in on a single core runs at 6Hz; A parallel version achieves 30Hz on a single i7 machine, processing 6 hours of video in 4 hours on a single machine, i.e. faster than real-time. We then can perform segmentation using dynamic programming [14].

In Figure 5-10 we demonstrate segmentation of a video feed taken from Google

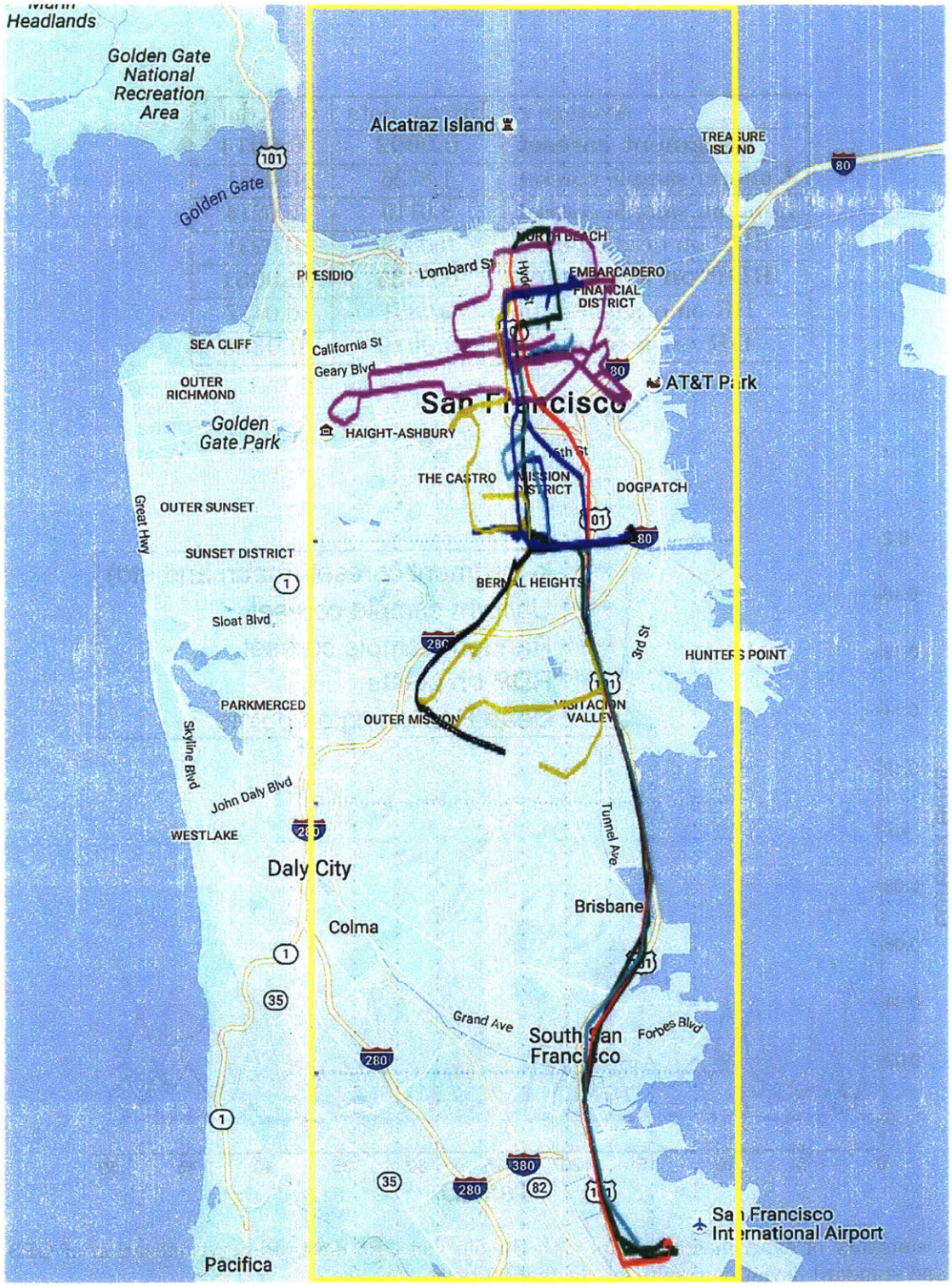


(a) Normalized GPS data from taxis in San Francisco overlayed with a Dead Reckoning segmentation computed on our coreset.



(b) Latitude/Longitude plot demonstrating that the segmentation yields a meaningful spacial interpretation (cf. Fig. 5-4)

Figure 5-3: Segmentation of GPS data from taxis in San Francisco



Latitude/Longitude plot overlayed on a map of San Francisco. Note that the segmentation yields a meaningful spacial interpretation, by separating trajectories across the various parts of the downtown area (yellow box).

Figure 5-4: Lat/Long plot overlayed on a map of San Francisco

Average ε	Bitcoin data	GPS data
k-segment coreset	**0.0092**	**0.0014**
Uniform sample coreset	1.8726	0.0121
Random sample coreset	8.0110	0.0214
RDP on original data	0.0366	0.0231
RDP on k-segment	**0.0335**	**0.0051**
DR on original data	0.0851	0.0417
DR on k-segment	**0.0619**	**0.0385**

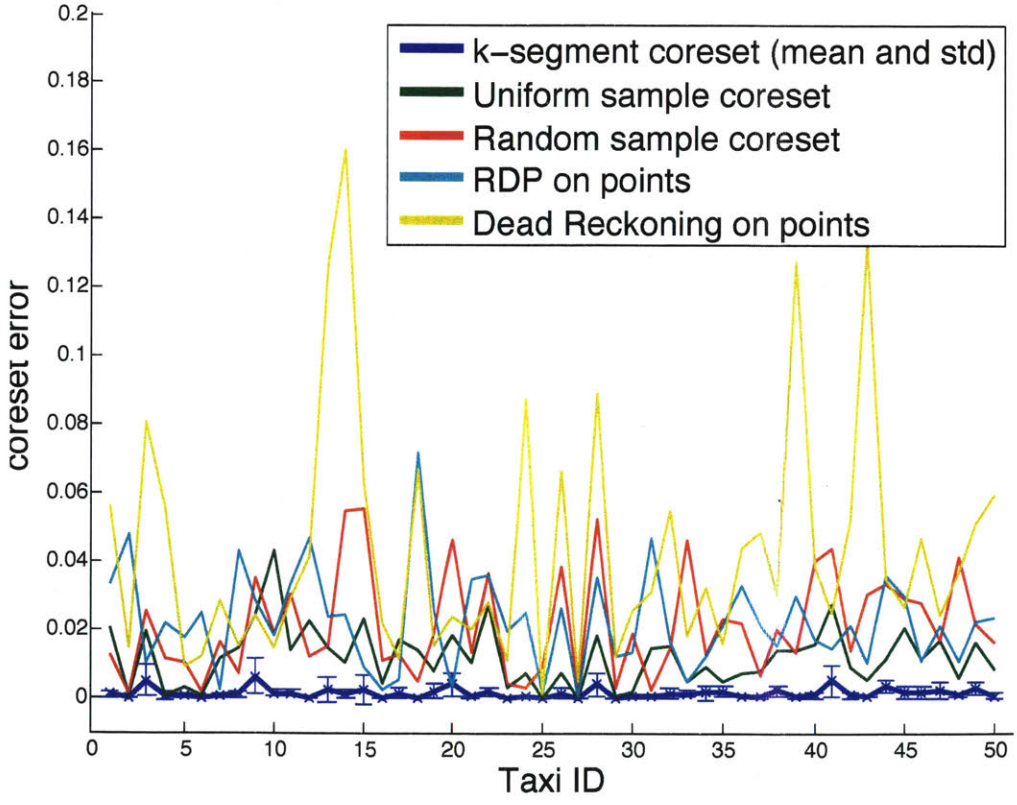
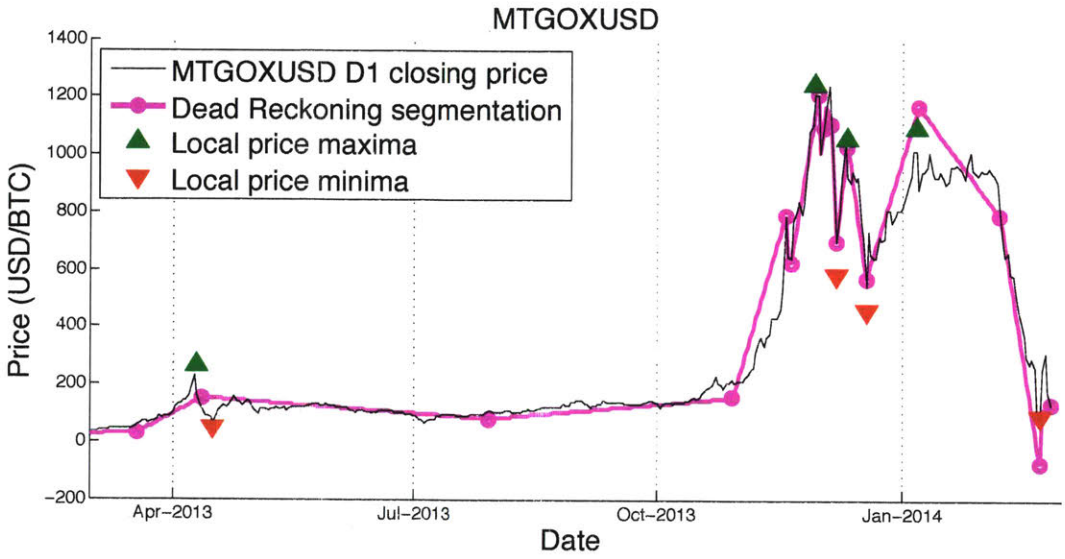
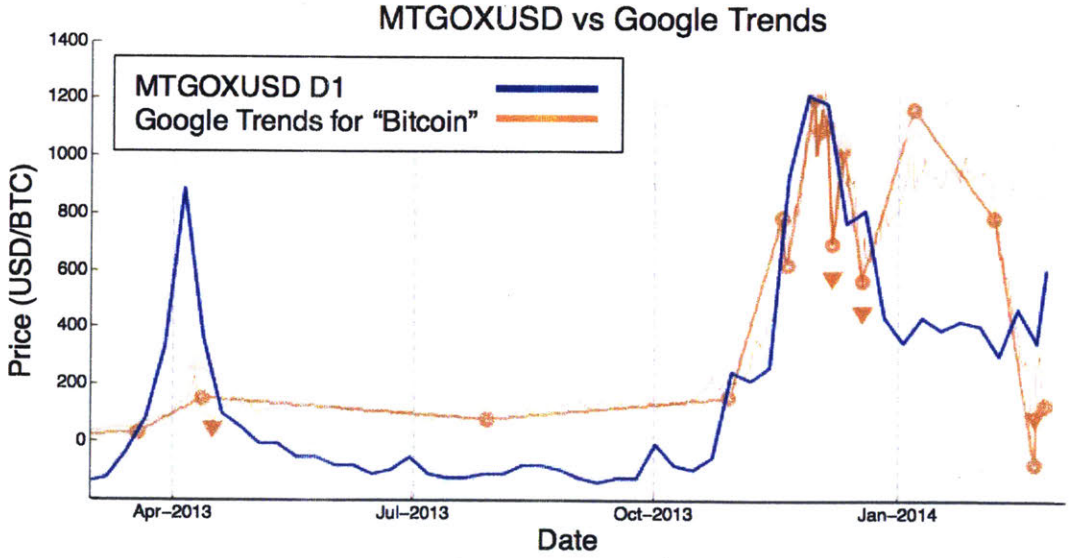


Table: summary of experimental results with Bitcoin and GPS data. Plot: visualization of GPS error and standard deviation results for the first 50 taxis.

Figure 5-5: Summary of experiments with Bitcoin and GPS data



(a) Daily Bitcoin price data (MTGOXUSD D1), from Jan 1, 2013 – Apr 1, 2014, overlayed with a Dead Reckoning segmentation computed on our coreset. The red/green triangles represent local min/max, indicating prominent market events [13, 25].

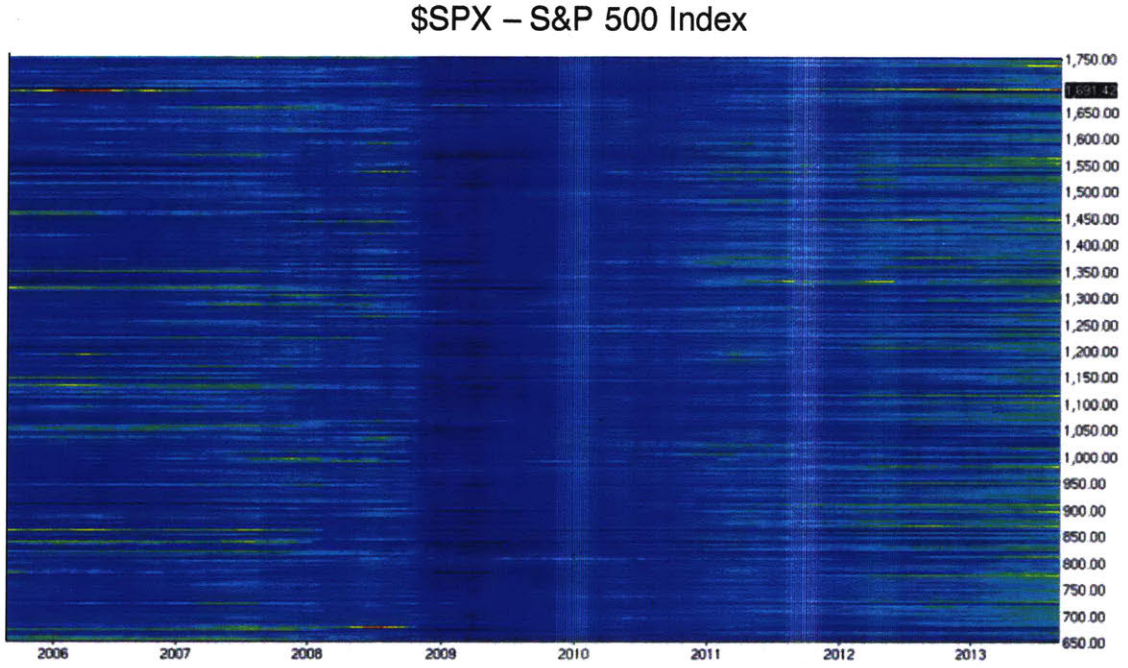


(b) MTGOXUSD daily Bitcoin prices (Jan 1, 2013 – Apr 1, 2014) compared against Google Trends search interest over time for "Bitcoin" [1].

Figure 5-6: MTGOXUSD daily Bitcoin price segmentation

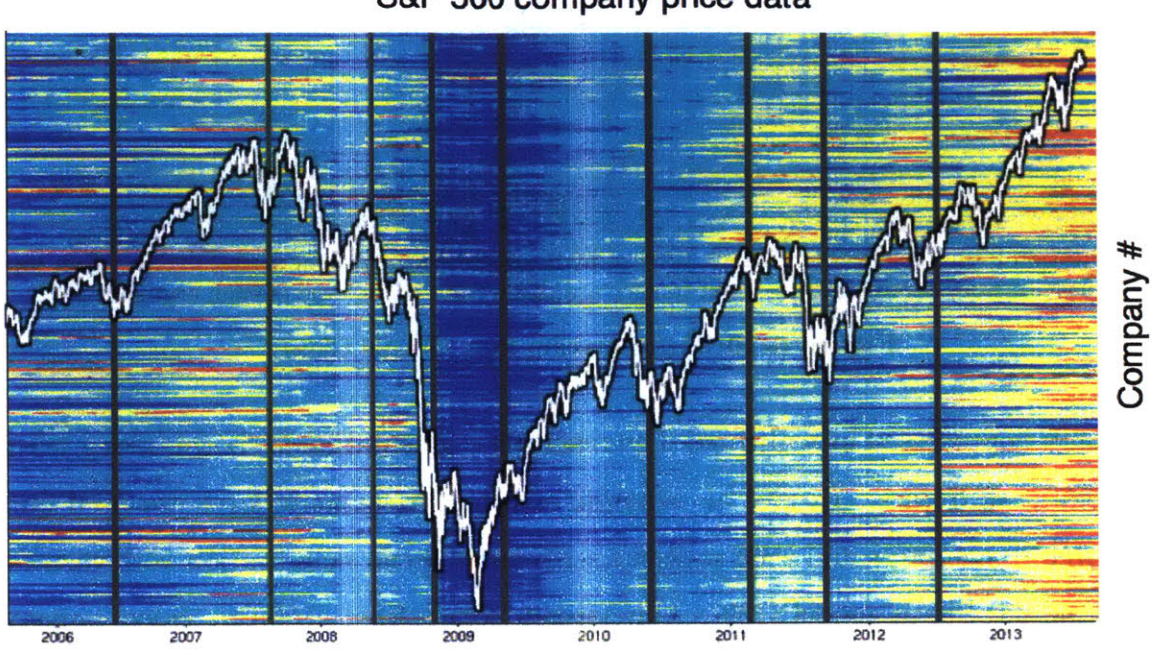
S&P 500 Index ($SPX) for the period from Aug 29, 2005 – Aug 9, 2013.

Figure 5-7: S&P 500 index plot



Normalized S&P 500 stock quotes for the period from Aug 29, 2005 – Aug 9, 2013.

Figure 5-8: S&P 500 stock quotes



S&P 500 Index vs segmentation of normalized S&P 500 stock quotes using Bellman's algorithm over the coreset, for $K = 10$.

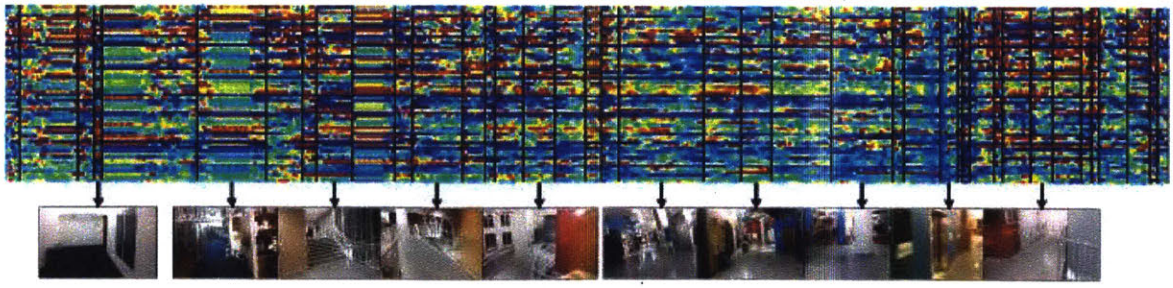
Figure 5-9: S&P 500 index vs S&P 500 stock quotes segmentation

Glass. We visualize the BOWs, as well as the segments suggested by the k-segment mean algorithm [14] run on the coreset. Inspecting the results, most segment transitions occur at scene and room changes. The resulting segmentation can be used so as to reason about the places traversed, activity of the wearer, and unusual events.

An additional property of our algorithm is the ability to handle segmentation based on several data streams, whose coresets are computed separately. Segmentation can then be computed on the combined data stream.

We note that semantic segmentation of video is still unsolved and in particular, it can not be done in real-time. Our method for segmentation runs in real-time and can further be used to automatically summarize the video by associating representative frames with segments. To evaluate the "semantic" quality of our segmentation, we compared the resulting segments to uniform segmentation by contrasting them with a human annotation of the video into scenes. Our method gave a 25% improvement in the Rand index [112] over a 3000 frames sequence.

A larger scale example is given in Figure 5-11c, where data from a 3-hours tour



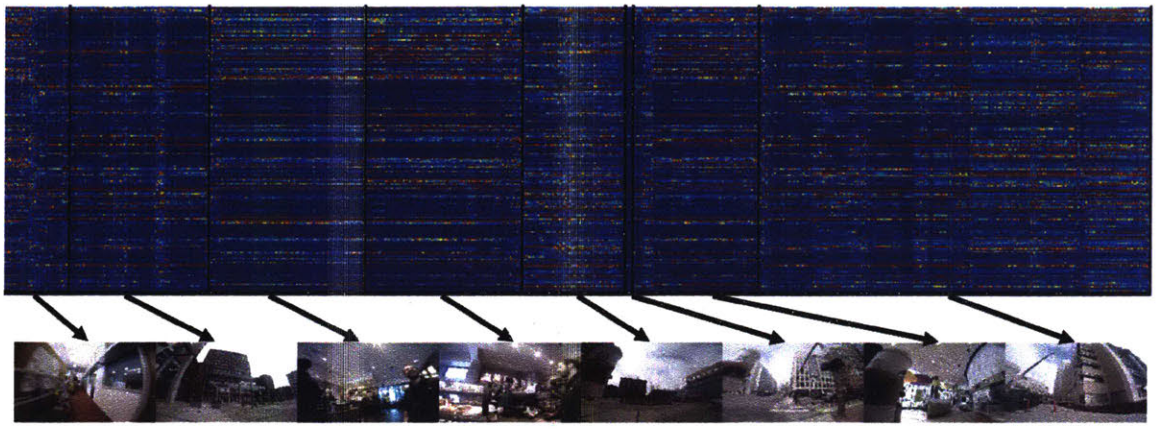
Segmentation from Google Glass. Black vertical lines present segment boundaries, overlayed on top of the bags of word representation. Icon images are taken from the middle of each segment.

Figure 5-10: Segmentation from Google Glass

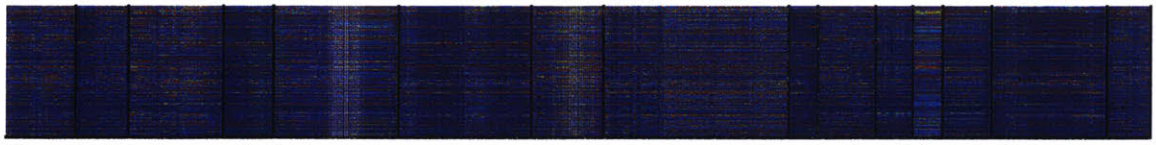
of Boston is processed in real-time. The resulting segmentation is shown, along with representative frames. To give an example at a higher semantic level, we used our algorithm to compress and then segment a stream of scene classification vote vectors based on the Places model [139]. Our stream comes from a GoPro video camera strapped to a person, going from a lab space, to a couple of meetings, and back to the lab, totaling 180,000 frames and vector dimensionality of 205. As can be seen in Figure 5-11, the resulting 8-segmentation clearly shows the transitions between scene types, and would match the intuitive summary for this video.

5.1.4 Technical Summary

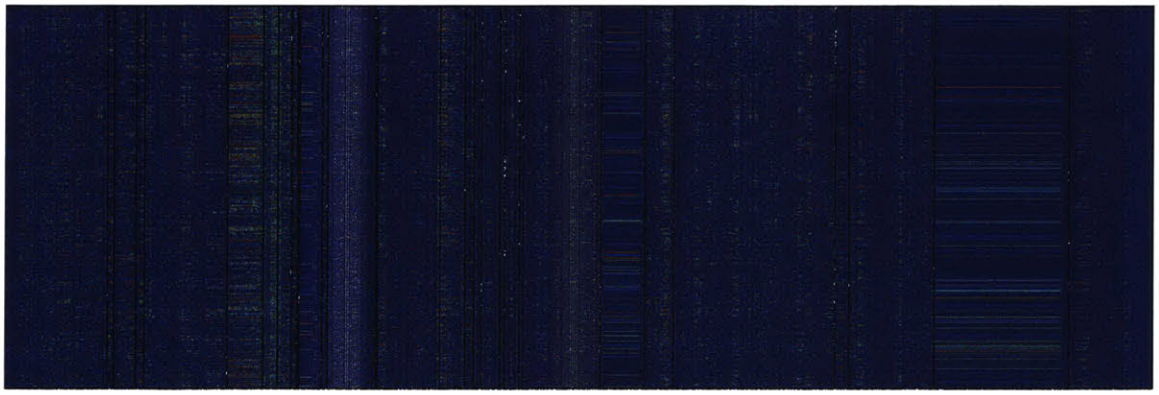
- SVD provides a $(1, 0)$-coreset for the 1-segment mean.
- BICRITERIA algorithm estimates the complexity of the data in $O(\log n)$ iterations.
- BALANCEDPARTITION algorithm uses the complexity BICRITERIA estimate to construct a (k, ε)-coreset for the k-segment mean.
- The k-segment mean coreset is of size $O(k/\varepsilon^2)$.
- The k-segment mean coreset is constructed in $O(dk)$ time.
- The k-segment mean coreset is constructed using $O(\log n)$ memory.



(a) Segmentation based on the Places CNN model of 30 minutes of egocentric video. Black vertical lines present segment boundaries, overlayed on top of the scene votes. Icon images are taken from the middle of each segment. The two meeting places (coffeeshop and restaurant) are easily captured as their own segment, where the other segments include time spent inside the laboratory and walking in the street.



(b) Segmentation based on the Places CNN model for a period of 1h:40m, or 180,000 frames. Black vertical lines present segment boundaries, overlayed on top of the scene votes. As can be seen, the resulting segments capture the changes in scene type.



(c) Segmentation based on bags of features of 3 hours taken from the Boston Trolley tour. Black vertical lines present segment boundaries, overlayed on top of the scene votes. Icon images are taken from the middle of each segment. The resulting segmentation matches the changes in scene content, as expected.

Figure 5-11: Segmentation based on the Places CNN model

5.2 Conclusions

In Chapters 4 and 5 we demonstrated a new framework for segmentation and event summarization of video data from robot cameras. We showed the effectiveness and scalability of our proposed algorithms, and their applicability for large distributed video analysis. In the context of video processing, we demonstrate how using the right framework for analysis and clustering, even relatively straightforward representations of image content lead to a meaningful and reliable segmentation of video streams at real-time speeds.

Chapter 6

Coresets for Summarization – Applications to Localization and Retrieval

In this chapter[1] we demonstrate how our algorithms for high-dimensional stream compression based on the k-segment mean coreset can be leveraged for efficient summarization, state estimation, and retrieval for large video streams in continuously operating robotic systems. We present an efficient feature-based coreset algorithm for summarizing video data with significantly lower memory requirements than existing methods. We demonstrate a system implementation of the described algorithm that generates a visual tree of keyframes that provide an image-based summary of the video. We present a variety of experimental results that characterizes the efficiency and utility of the resulting system for robotic localization and loop closure.

This chapter is organized as follows. We define and describe the relevant coresets, structures, and algorithms in Section 6.1. In Section 6.2 we define the require notations for localization. This is followed by the details of the proposed closure detection and retrieval algorithms in Section 6.3. Finally, in Section 6.4 we demonstrate empirical results of our algorithm, both on existing and new datasets.

[1]Some of the content in this chapter was published in [135].

6.1 Coresets And Stream Compression

We now turn to describe the coreset used in this work and the specific properties that make it useful for video retrieval and summarization. In the problem statement we query an observed image from a static set of observed locations. However, in practice we are given a possible unbounded video stream of multiple frames per second (50/60 is the HD standard). To this end, we select a representative over-segmentation of the video stream (along with a compact representation of each segment) called a *coreset*. The coreset approximates the original data in a provable way, that guarantees a good trade-off between the size of the coreset and the approximation of each scene in the video stream. More precisely, we embed images into $\mathbb{R}^d$ based on a naive Bayes approximation, representing n images as a set n points in $\mathbb{R}^d$. The algorithm outputs a set, called ε-coreset, of roughly k/ε weighted segments approximating the data, such that the sum of squared distances over the original points and approximating segments to every k piecewise linear function is the same, up to a factor of $1 \pm \varepsilon$. The existence and construction of coresets has been investigated for a number of problems in computational geometry and machine learning in many recent papers (cf. surveys in [46]). The assumption in this model is that similar images corresponds to points on approximately the same time segment.

In this work we leverage our most recent results for high-dimensional data segmentation [114] and present algorithms to perform efficient loop closure detection and retrieval for arbitrary large videos. The core of our system is the k-segment coreset, which provides flexibility with respect to varying dimensionalities, multiple sensors, and different cost functions. For a single segment ($k = 1$) no segmentation is necessary, and the data is represented using SVD. For multiple segments we want to know how many segments are in the partition, how to divide them, and how to approximate each segment. We present a two-part coreset construction: first we estimate the complexity of the data using a *bicriteria* algorithm; second we define a fine partition of the data into coreset segments using a *balanced partition* algorithm, and we approximate each segment by SVD. The algorithm guarantees a segmentation

that is close to k-segments with a cost that is close to the optimal cost.

6.1.1 Streaming and Parallelization

One major advantage of coresets is that they can be constructed in parallel, as well as in a streaming setting where data points arrive one by one. This is important in scenarios where it is impossible to keep the entire data in random access memory. The key insight is that coresets satisfy certain composition properties, which have first been used by [65] for streaming and parallel construction of coresets for k-median and k-means clustering. These properties are:

1. The union of two ε-coresets is an ε-coreset.

2. An ε-coreset of such a union is an $\varepsilon(1+\varepsilon)$-coreset.

We note that while the first property may seem trivial by concatenating the elements of the coresets, the second property relates to the desired compactness of the representation, and states that we can further compactness the unified coreset so that it scales nicely as more data is added.

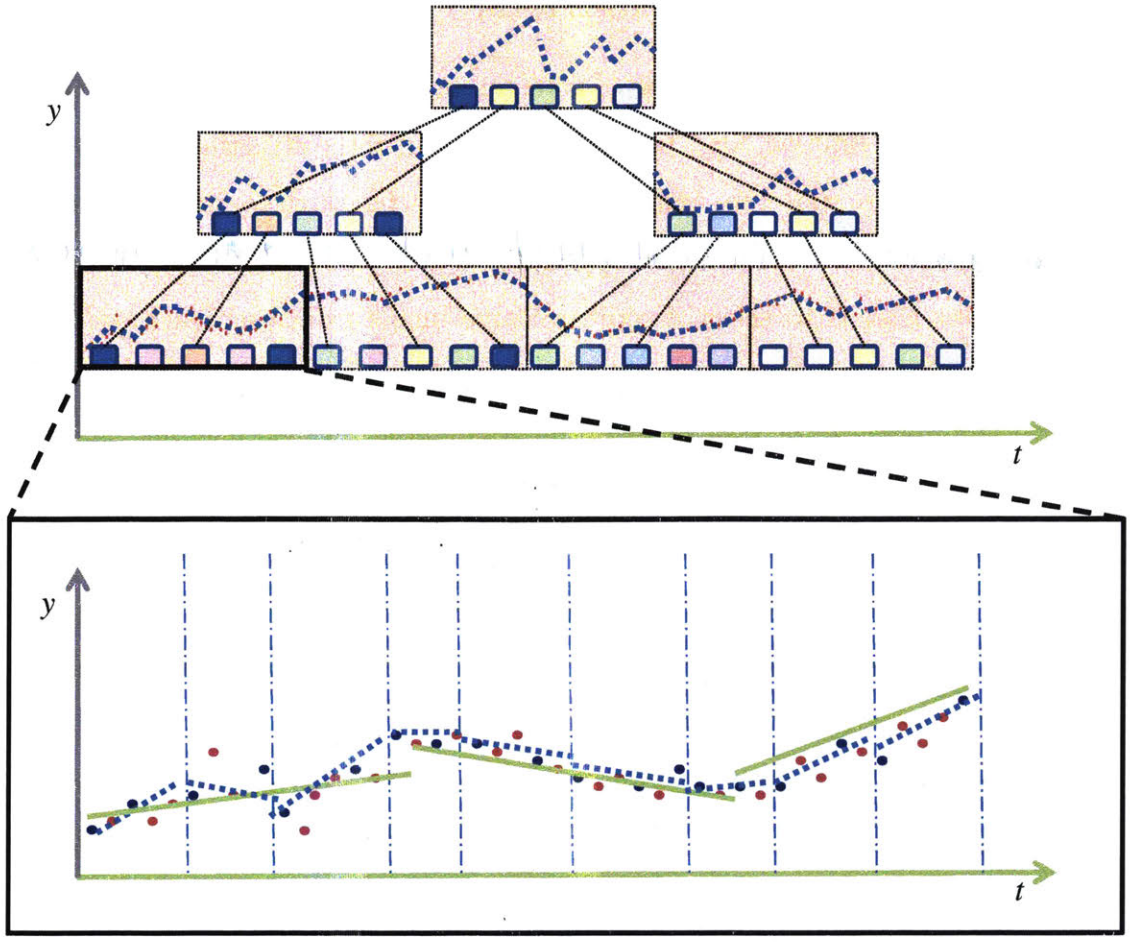
Streaming In the streaming setting, we assume that points arrive one-by-one, but we do not have enough memory to remember the entire data set. Thus, we wish to maintain a coreset over time, while keeping only a small subset of $O(\log n)$ coresets in memory. Since each coreset is small, the overall memory consumption is also small. Using the properties above, we can construct a coreset for every block of consecutive points arriving in a stream. When we have two coresets in memory, we can merge them (by property 1), and re-compress them (by property 2) to avoid increase in the coreset size. An important subtlety arises: while merging two coresets does not increase the approximation error, compressing a coreset does increase the error. However, using a binary tree as shown in Fig. 6-1, the final approximation in the root is roughly $O(\varepsilon \log n)$ for an original stream of n points, which can be reduced to ε by using a little smaller value for ε (cf. [114] for a discussion of this point in the context of k-segmentation).

We call the tree resulting from the coreset merges the *coreset streaming tree.* We denote the coresets created directly from data as *streaming leaves.* An additional, useful structure can be defined by looking at segment merges in the balanced partition algorithm in [114]. This structure is the *coreset segment tree.*

Parallelization Using the same ideas from the streaming model, a (nonparallel) coreset construction can be transformed into a parallel one. We partition the data into sets, and compute coresets for each set, independently, on different computers in a cluster. We then merge two coresets (by property 1), and compute a single coreset for every pair of such coresets (by property 2). Continuing in this manner yields a process that takes $O(\log n)$ iterations of parallel computation. This computation is also naturally suited for map-reduce [35] style computations, where the map tasks compute coresets for disjoint parts of the data, and the reduce tasks perform the merge-and-compress operations. We note that unlike coresets for clustering such as the one used in [108], parallel computation requires us to keep track of the time associated with the datapoints sent to each processor.

New Approaches We now discuss how our work differs from, and builds on, existing state of the art in this respect. Using the above techniques, existing coreset construction algorithms allow us to handle streaming and parallel data but not both. This is because the parallel approach assumes that we can partition the data in advance and split it between the machines. However, we cannot split an unbounded stream in such a way when not all the data is available in advance. In our problem we wish to process streaming video on the cloud, i.e., computing it for streaming data *and* in parallel. In other words, we want to compress the data in a distributive manner while it is uploaded to the cloud.

There are two contexts in which our system allows simultaneous streaming and parallelization. The first is streaming of buffered data. For example (a robotic scenario), in the case of an autonomous exploration vehicle or UAV collecting a high volume of video, it is still useful and sometimes necessary to stream the data to a collection point at a later time. Another example is a wearable device with an intermittent connection to a data server that will buffer data at times when it is unable



Streaming coreset construction of a data stream. The bottom figure illustrates the online construction and segmentation of a block from an incoming data stream. Coreset segments are shown with dashed blue lines. The top figure illustrates the continuous compression of the data stream through progressive merge/reduce computation of the coresets from lower level coresets.

Figure 6-1: Streaming coreset construction

to upload it in real time. In both cases, the context will dictate a sufficient leaf size beyond which temporal continuity is not expected, and continuous data blocks of this size can be streamed in parallel.

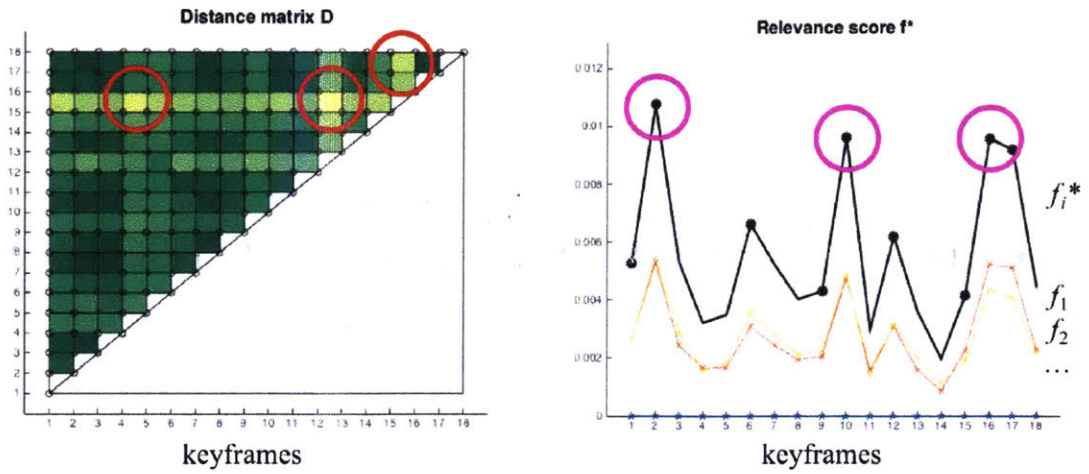
Another context that is prevalent in robotics is a multi-source video stream (such as the FAB-MAP dataset [31], which is considered an industry standard). In this case, we can parallelize the coreset construction by streaming each view separately, multiplexing video streams where necessary. Temporal continuity is naturally preserved.

6.1.2 Summarization and Retrieval

Roughly, in video summarization the goal is to get an image representative from each scene in a given time interval, for a partition that captures the structure of the video. Using the right feature space, we assume that images in the same scene are generated by a simple model. We use a linear approximation to allow short-scale temporal variations. Our goal is to select a representative image from each such segment. However, the coreset for this problem only guarantees that the k-segment of the coreset will approximate the k-segment of the original data, which may not be a fine enough segmentation. In addition, the assumption above holds usually but not always – thein practice images that are intuitively similar appear on different segments, due to appearance of new objects or viewing of different scene areas with a small FOV camera. In this work we demonstrate how the coresets streaming framework allows us to overcome these difficulties with little effort.

In this work we differentiate between several hierarchical structures that are created during the stream processing. The traditional two are the coreset streaming tree defined by the merging of batches in the data stream, and the coreset segment tree, which depicts the merging of coreset segments during the streaming. The former has a fixed topology regardless of the data, while the latter is adaptive to transitions in the data, yet does not prioritize different aspects of the data points themselves. We add on top of these a third structure, which allows us to get this unary adaptivity.

Specifically, we add a layer that stores a carefully chosen set of images along with each node of the coresets tree, called *key frames*. Besides accomodating possible partitions of the data, these key frames are expected to capture the variability of the observed scenes in the video, and provide for various applicative needs. We detail the selection of the key frames in Section 6.3.1. These key frames are merged both according to the images quality and their ability to represent other images in the video, and according their representation of video transitions, as captured by segment merges in the coreset algorithm. We call the tree formed by selection of prominent key frames the *keyframe merge tree*.



(a) The ℓ_2 distance matrix D shows the difference between the candidate frames in descriptor space. The 3 red circles indicate the 3 chosen keyframes using FPS only. (b) The relevance score f^* is the sum of all the keyframe selection metrics $f_1, f_2, \ldots$. The 3 magenta circles indicate the 3 chosen keyframes using the relevance score only. The 9 out of 18 keyframes eventually selected during the merge of two nodes are shown with black dots.

Figure 6-2: Distance matrix vs relevance score

In the context of life-long processing and retrieval, the coreset streaming tree is usually used to support the streaming and parallel model as explained in Section 6.1.1, where only a small subset of the coresets in the tree exists at any given moment.

6.2 Loop Closure Problem Formulation

We now describe the loop closure problem following the notation of [31]. Denoting the set of observations at time k, usually extracted from an observed image by Z_k, we wish to associate it with a unique location L_i. This is done by evaluation of the set under a location-dependent probabilistic model

$$\Pr(Z_k \mid L_i) = \Pr(z_1, \ldots, z_{|v|} \mid L_i), \tag{6.1}$$

where $|v|$ denotes the number of features, and z_i denotes the indicator for appearance of feature i in the image k. While some methods consider the variables z_i as binary variables, in many cases, counts of feature appearance may be multiple (especially if the visual vocabulary used is small).

We are looking for the location that maximizes the conditional probability

$$\Pr(L_i \mid Z_k) = \frac{\Pr(Z_k \mid L_i)\Pr(L_i)}{\Pr(Z_k)}, \tag{6.2}$$

and for the purpose of this work, we ignore the temporal dependency that is often sought [31]

$$\Pr(L_i \mid Z_k, Z^{k-1}) = \frac{\Pr(Z_k \mid L_i, Z^{k-1})\Pr(L_i, Z^{k-1})}{\Pr(Z_k, Z^{k-1})}, \tag{6.3}$$

where Z^{k-1} denotes the observation history up to time k.

Several approximation have been considered for computing $\Pr(Z_k \mid L_i)$. The simplest approximation is the naive Bayes approximation

$$\Pr(Z_k \mid L_i) = \prod_{i=1}^{|v|} \Pr(z_i \mid L_i), \tag{6.4}$$

which leads to ℓ_2 distance measure between observations and location distributions, while assuming

$$\Pr(z_j \mid L_i) \propto exp\left\{-(z_j - \mu_j(L_i))^2/\sigma^2\right\}, \tag{6.5}$$

where we do not assume a binary appearance vector. We note that the log-probability of the observation given a location naive Bayes model is the ℓ_2 distance in feature-space. This distance is the distortion measure approximated by k-segment mean coreset for k-segment models.

Another often used approximation is the Chow-Liu tree [21]. As proposed in [31], the naive Bayes PDF model can be improved upon by an optimal tree (in the KL-divergence sense) with respect to the empirical distribution of the location in a way that is still tractable (i.e by solving a max-weight spanning tree problem).

6.3 Retrieval Algorithms

We now detail the construction required for the summarization and retrieval tasks described above, in terms of the augmentation of the coreset structures, and the

algorithms running on top of the coresets.

6.3.1 Incorporating Keyframes into a Coresets Tree

We first describe the incorporation of keyframes into the coreset streaming tree construction. This allows us to demonstrate the use of keyframes stored in the coreset tree in localization, keeping only a fraction of the frames for localization, At each streaming leaf SL we keep a set of K keyframes, for some fixed K. With each node merge in the tree, we select the new set of keyframes from the ones existing in the children nodes by running a modified *farthest-point-sampling* (FPS) algorithm [68, 60] on the feature vectors. The FPS chooses the frame with feature vector x from the data, that is farthest from the existing set S_{j-1}

$$x_j = \underset{x}{\operatorname{argmax}}\, d(x, S_{j-1}), \tag{6.6}$$

with S_{j-1} marking the set of previously chosen frames, in terms of their feature vectors, and $d(x_i, x_j)$ is the ℓ_2 distance between keyframes x_i, x_j in the feature space. Here, we modify the FPS selection rule by adding an image relevance score term that can include image quality (sharpness and saliency), temporal information (time span and number of represented segments), and other quality and importance measures. The relevance score is defined as

$$f^*(x) = \alpha_T f_T(x) + \alpha_S f_S(x) + \alpha_B f_B(x) \tag{6.7}$$

where positive $f_i(x_j)$ indicates higher relevance. The relevance score $f_B(x_j)$ is the the blur measure for image j based on [29] (negated for consistency with our positive relevance convention). The relevance scores $f_T(x_j), f_S(x_j)$ denote video time and number of coreset segments associated with the keyframe x_j, respectively. More generally

$$f^*(x_j) = \textstyle\sum_{i=1}^{N} \alpha_i \; f_i(x_j) \tag{6.8}$$

for any set of metrics $1 \ldots N$, such as the example in Fig. 6-2(b). The weights α allow us to fine-tune the system, for example to give more weight to image quality vs temporal span. This allows us to get a rich set of representative keyframes that are meaningful in the video in terms of time span and complexity, Given a starting point x_0 we modify the FPS algorithm to include the relevance score. The new point is then given by

$$\begin{aligned} x_j &= \operatorname*{argmax}_x \left\{ \hat{d}(x, S_{j-1}) \right\} \\ &= \operatorname*{argmax}_x \left\{ d(x, S_{j-1}) + f^*(x) \right\}. \end{aligned} \tag{6.9}$$

It can be shown that the resulting selected set is close to the optimal set if the values of the score function are bounded and sufficiently close to 0, converging in the limit to the 2-optimality guarantee of FPS. Let S be the set chosen by the modified FPS, and let

$$\rho(S) = \max_x \hat{d}(x, S). \tag{6.10}$$

Lemma 5. *Let S^* be an optimal representative set given by $S^* = \operatorname{argmin}_{S'} \rho(S')$. Then*

$$\rho(S) \leq 2\rho(S^*) - \min_x f^*(x) \tag{6.11}$$

Proof. The proof for a specific k and S_k is done similar to the FPS proof [99], by looking at x_{max}, the maximizer of $\hat{d}(x, S_k)$, along with S_k. By comparing this set to S_k^* using

$$\hat{d}(x, y), \; x \in \{x_{max}\} \cup S_k, \; y \in S_k^* \tag{6.12}$$

we have two elements x_1, x_2 with $\hat{d}$ minimal to the same element $s^* \in S_k^*$, by the pigeonhole principle. Let us assume x_1 to be the latest of the two. It can be shown

Algorithm 8 SAMPLEOLDNODE(V)

Input: $V = v_1, \ldots, v_{end}$, the set of coreset tree nodes
Output: v, node sampled from the tree before v_{end}

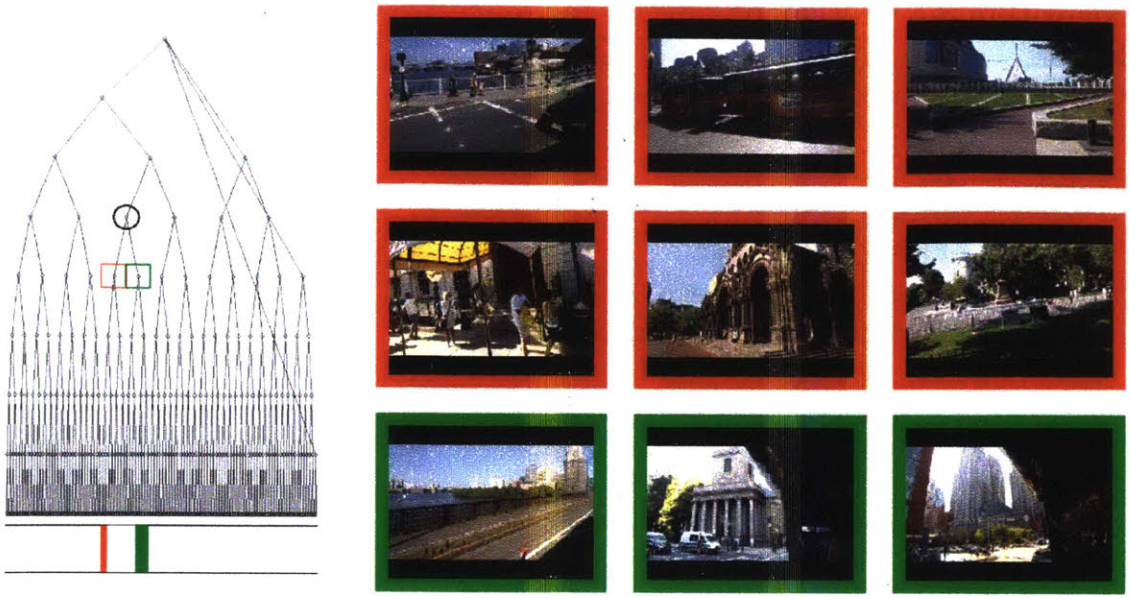
$v \leftarrow v_{end}$
$t_{max} \leftarrow \infty$
started_descent $\leftarrow 0$
$\alpha < 1$ is a fixed parameter
while *started_descent* $\neq 1$ **do**
  sample p uniformly at random from $[0, 1]$
  if v is not the root node **and** $p < \alpha$ **then**
    $t_{max} \leftarrow t_{start}(v)$
    $v \leftarrow$ PARENT(v)
  else
    started_descent $\leftarrow 1$
  end if
end while
while v is not a leaf node **do**
  $v \leftarrow$ OLDCHILD(v, t_{max}, KEYFRAMES(v))
  started_descent $\leftarrow 1$
end while
return v

that $\hat{d}(x_{max}, S_k) \leq \hat{d}(x_1, x_2)$, due to the order of selection of x_{max}, x_1, x_2,

$$\begin{aligned} \hat{d}(x_1, x_2) &\leq \hat{d}(x_1, s^*) + \hat{d}(x_2, s^*) - f(x_2) \\ &\leq 2\rho(S_k^*) - \min_x(f^*(x)), \end{aligned} \tag{6.13}$$

by triangle inequality over d and the definition of $\hat{d}$. □

The function f^* makes the coreset tree more adapted to the data. By storing this relevance score for each node in the coreset tree, the relative importance of individual keyframes is propagated along the binary tree structure, allowing us to query the tree for this information at each level. In general, our coreset tree system facilitates the addition of other metrics, such as object detection, optic flow, etc. that allow us to emphasize the semantic content that is most appropriate to the problem.



The interactive UI described in Section 6.3.2 showing the coreset tree for the Boston Trolley Tour (left). The image collage shows 9 keyframes captured from this data (right). Observe that the keyframes with the red/green margins propagated from the left/right child nodes; the corresponding represented time interval is shown at the bottom of the tree.

Figure 6-3: Interactive coreset tree retrieval UI

6.3.2 User-Interface for Retrieval

The proposed key-frames allow easy and useful search and retrieval for a human user. We now describe user interface that allows the user to browse through the video stream and see the summarization and representative images from each interval using a visual tree, as shown for example in Fig. 6-3. We note that our user interface, summarization and time search approach can be useful for any other type of coresets on images, such as k-means clustering, or for trajectories summarization. The interface demonstrate a non-standard retrieval task and could be relevant for robotic tasks, such as searching for people and objects in a subsequence of the video, change detections, and so forth.

In the proposed UI, the user can browse through the binary tree on the left. The node with the white circle corresponds to the selected time interval. Its left child is always marked by a red circle, and its right child by a green one. The red and green rectangulars on the bottom of Fig. 6-3 mark the relevant leaves of the red and green nodes respectively. In the right side of the figure we show the selected key frames in the white nodes. The key frames that were chosen from the red note are marked by

a red border, and similarly the other key frames are marked by a green border.

Additionally, the user can specify a time span that he or she is interested in summarizing. The user interface computes the minimum subtree that summarizes the activity in that time region. The relevant images are extracted from the hard drive according to the user clicks. Theoretically, the retrieval time of the relevant coreset is poly-logarithmic in the size of the related video stream using the properties and structures of the coreset tree.

6.3.3 Life-long loop closure

We now proceed to describe how life-long loop closure can be performed based on the coreset streaming segment tree. While the segments in the coreset streaming tree provide an adaptive tradeoff between temporal resolution and efficiency, different nodes in the coreset streaming tree are still of equal time span. However, we expect the data-adaptivity of the coreset to allow efficient retrieval for arbitrarily long videos. To this end, we define a method for random caching of frames for loop closure detection, based on the graph distance in the streaming segment tree. Similar to RTAB-MAP [78], we assume the system to include a *working memory* (*WM* in the notation of [78]) and an index based on the coreset streaming tree in memory. The coreset streaming tree nodes point to a database of frames (or their corresponding locations, in the case of a location-based retrieval), we denote as *long-term memory* (or LTM), where each leaf is a page of frames to be swapped in and out. We define retrieval and discard rules for pages containing previous frames, between a working memory and the database of the robot's visual history. We note that a similar approach could be incorporated with location-based mapping by replacing the frames descriptors with pointers to locations observation models.

The retrieval rule we employ randomly selects pages based according to the procedure SAMPLEOLDNODE presented in Algorithm 8. The procedure uses the function OLDCHILD($v, t, keyframes$), that returns a child of v recorded at t_{end} older than t at random. We adapt the sampling so that each child has a weight proportional to the number of keyframes the parent node drew from it, plus 1 (to ensure a non-zero

Algorithm 9 UPDATECLOSURECACHE(x_{ref})

Input: x_{ref}, the reference image descriptor
Output: X, the matching candidates for x_{ref}

$WM = \emptyset$
while 1 **do**
  $v \leftarrow$ SAMPLEOLDNODE(v_{end})
  if $v \notin WM$ **and** FULL(WM) **then**
    select node v_{rem} from WM at random
    $WM \leftarrow WM \setminus \{v_{rem}\}$
  end if
  $WM \leftarrow WM \cup \{v\}$.
  compute loop closure probabilities for x_{ref} using WM
  compute matching candidates X
end while
return X

An image retrieval example from the Boston dataset, using the tree-sampling method in Algorithms 8, 9. The green frame marks the query image corresponding to x_{ref} in Algorithm 9. The other images are the maximum-probability match found by sampling.

Figure 6-4: Boston tour loop closure results

sampling probability). This allows us to take into account quality metrics based on the coreset as well as image quality/saliency, as described in Subsection 6.3.1. The effect of the weighted traversal can be seen in Fig. 6-5b.

It can be seen that the probability of reaching leaf ℓ, by traversing from the last leaf in the tree v_{end}, is non-zero. Since reaching each leaf from v_{end} has exactly one path, and this corresponds to a single set of choices in the conditionals we can bound

this probability from below by looking at the direct route from v_{end} to the node ℓ,

$$\begin{aligned} & p\left(\text{SAMPLEOLDNODE}(v_{end}) = \ell\right) \\ & \geq \alpha^{d_U(v_{end},\ell)} \left(\frac{1}{d_{max}}\right)^{d_D(v_{end},\ell)} \\ & \geq \left(\min\left(\alpha, \frac{1}{d_{max}}\right)\right)^{d(v_{end},\ell)}, \end{aligned} \tag{6.14}$$

where $d_D(v_{end}, \ell)$ denotes the path length from the common root of v_{end} and ℓ to ℓ, and $d_U(v_{end}, \ell)$ denotes the path length from the common root to v_{end}. It is easy to see that

$$p(\text{SAMPLEOLDNODE}(v_{end})) = \ell) \leq \alpha^{d_U(v_{end},\ell)}. \tag{6.15}$$

Let us mark by t_{start} and t_{end} the beginning and end of the time span associated with each node or leaf of the tree. It is furthermore easy to see that going up to the parent node, t_{start} is non-increasing and t_{end} is non-decreasing. Going down to an earlier child, t_{start} is non-increasing. This can be summed up in the following lemma:

Lemma 6. SAMPLEOLDNODE(v) *samples a leaf whose span ends before* $t_{end}(v)$*, and all previous leaves have a non-zero probability of being sampled, as described in equation* (6.14)*.*

Based on this sampling procedure we present the loop closure detection algorithm described in Algorithm 9, which operates according to a maximum time allotted per turn, in order to fit a real-time regime. We define FULL(WM) to be a function indicating whether the working memory is full. The sampling probability of leaves according to Algorithm 8 can be seen in Fig. 6-5a, showing the adaptiveness of the method.

It can be shown that the pages in WM are distributed exponentially decreasing with respect to the tree distance from start node v_{end}, assuming pages are kept in a FIFO order in the cache. The probability of a leaf to be added to WM is bounded from zero, thus ensuring that every leaf (and the locations pointed by it) has a chance of being sampled.

6.4 Experimental Results

The primary dataset used in this study is a recording of a Boston Trolley Tour. The video was captured using Google Glass and spans 3 hours. Doing away with an obvious redundancy for these application [32], we conservatively subsample by a factor of 5, to around 75k frames. Fig. 6-3 (left) shows the coreset tree generated by processing the entire tour.

A preliminary set of experiments serve as a hard ground-truth demonstration of the correctness of the coreset tree for the purposes of video segmentation. For this demonstration we select 15 still frames capturing scenes of interest during the tour. The stills were duplicated for random periods of time into a synthetic video file. The purpose of these experiments is to demonstrate (a) the coreset tree's utility in successfully segmenting data, and (b) the capability of the coreset tree to adaptively propagate important keyframes to the higher nodes without repetition. Fig. 6-3 (right) shows the results of these experiments. We observe that the coreset tree has successfully segmented the data and captured all representative frames, and that each video still was captured by a keyframe, regardless of how long the original video segment was. This demonstrates the coreset's capacity to capture information. Secondly, we observe that as the keyframes propagate up to the node of the tree, the modified FPS algorithm described by equation (6.9) favors few repetitions of similar keyframes, by definition of the FPS algorithm. This highlights our coreset tree's capacity to summarize information in an adaptive manner that can be tailored to the problem domain.

6.4.1 Loop Closure Experiments

Loop closure experiments were first conducted on a short video of 6000 frames with 2 known ground-truth loops. A coreset tree was created using a set of 5000 VQ representatives trained on 100k SURF descriptors. In general, the choice of leaf size depends on the problem domain, and will reflect the typical temporal resolution of the sequence. We used leaf sizes of $100, 150, 200$. With 9 keyframes per leaf, a leaf

size of 200 represents a subsampling of the input data by a factor of more than 22. With larger leaf sizes, results were too sparse for this short video sequence, however for larger videos such as the Boston data, a leaf size that is an order of magnitude on par with the size of the test data is appropriate. An ℓ_2 distance map was calculated based on the descriptors of the leaf-node keyframes, and thresholded to produce a loop closure map.

The loop closure map produced by our coreset was compared against an equivalent loop closure using uniform sampling of frames from the test video. Keyframe descriptors were primed for a number of thresholds that give meaningful loop closure patterns. Results were evaluated objectively by computing the precision/recall trends for our coreset tree against uniform sampling (Fig. 6-6). We see a typical trend of precision decreasing with recall, as the true positives get outweighed by false negatives with increasing threshold. For all values of recall, we achieve a higher precision by using descriptors from the keyframes of the coreset tree compared against uniform sampling. These results demonstrate the ability the coreset tree to capture information that is useful for state of the art loop closure algorithms such as [32].

6.4.2 Large Scale Experiments

Large-scale experiments were carried out on the complete 3 hour video of the Boston tour. The complete video consists of 3 hours, totaling a 360,000 frames at 30 frames per second. This is a true demonstration of life-long loop closure, as attempting to store candidate frames for a video of this size takes a prohibitively large amount of space. The video consists of a single tour of the city traversed in a loop, such that there is approximately 30 minutes of overlapping trajectory that forms a loop closure at an offset of approximately 2 hours.

A coreset was computed offline for the entire video, and took 159 minutes to construct, which is faster than real-time with respect to the length of the video. We use SURF descriptors and VQ to compute a bag-of-words representation of the video, exactly as described in the previous chapter.

A Chow-Liu tree was constructed for the video using the coreset keyframes using

the FAB-MAP package [31]. We compute loop closure using both ℓ_2 distance and FAB-MAP confidence scores. We chose the best set of parameters after pre-computing confidence scores for a small set of ground truth pairs of frames displaying the same scene at different parts of the tour. We ran the existing closure detection system on these frames, and determined parameters by pivoting one parameter at a time against the fixed set of remaining parameters. The parameters were altered in the order above – first downsampling, then finding the number of strongest features, and finally the parameters for the feature detection and extraction steps.

Finally, we apply the experimental setup to compute loop closure for the entire video. Figure 6-7 shows the results. The thin lines indicate confidence scores for different time offsets. The thick black line shows the aggregate confidence score, and the dotted red line indicates a confidence threshold used to determine whether a loop was detected or not. We observe that a loop closure detection occurs at approximately 234,000 frames or 2.2 hours, which matches exactly with the ground truth.

6.4.3 Retrieval Experiments

We now demonstrate a retrieval application based on the keyframes defined in Subsection 6.3.1. For a larger scale retrieval experiment, we demonstrate retrieval for a given query image in Fig. 6-4. Given a query image, we show the results of 3 runs of the a search for a match (with some threshold on the ℓ_2 distance) that includes the query image and similar results. We resample tree leaves until a match is found, and in each leaf retrieved, we look for the minimum ℓ_2 distance match, starting with an empty WM.

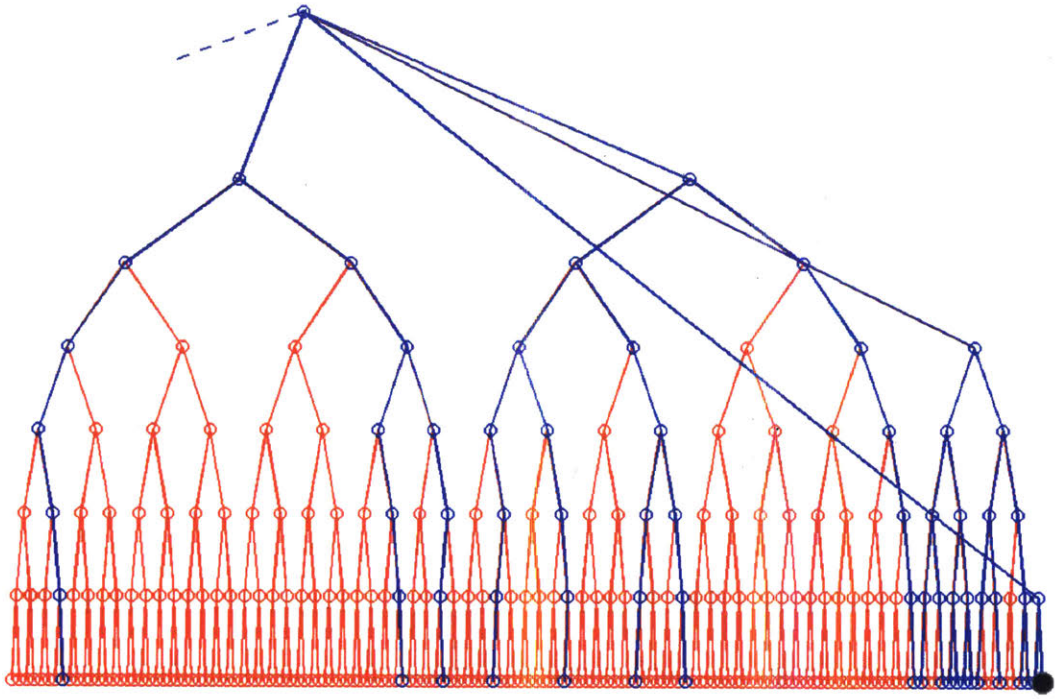
We note that processing the overall video of 75k frames can be done at 3 frames per second on an $i7$ CPU. The histogram of leaf retrievals until match is shown in Fig. 6-5c. The seek access attempts average of 70.5 is significantly lower that the uniform expected number of attempts of 108.5 because we expect caching according to recent hits and more adaptive keyframe-based sampling to further improve results.

6.4.4 Technical Summary

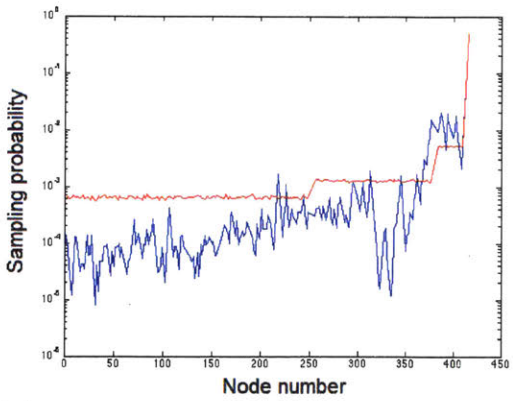
- Using union, compression, and merging properties we can compute a streaming coreset tree with $O(\log n)$ coresets.
- Each node in the coreset tree contains 9 representative keyframes that provide a semantic summary of the underlying coreset segments.
- Variability is represented by the ℓ_2 distance matrix for the candidate frames.
- Relevance is represented by encoding context-based relevance scores, which are weighted to form a single score for each frame.
- An adaptive keyframe selection algorithm propagates keyframes up the coreset tree by optimizing variability and relevance, and provably yields a result to within a constant factor approximation.
- SAMPLEOLDNODE and UPDATECLOSURECACHE probabilistically sample the coreset tree to select the best loop closure candidate frames.

6.5 Conclusions

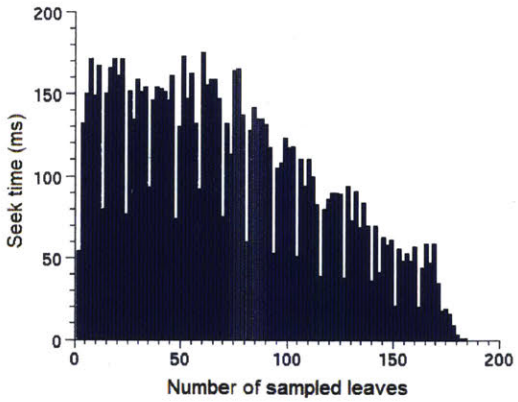
In this chapter we demonstrated how coresets for the k-segment means can form a basis for summarization of visual histories, for retrieval and localization tasks. We show how the coreset streaming tree can be used for visual loop closure and image retrieval from a video sequence over large videos, essentially summarizing the video at real-time speeds. In future work we expect to extend the use of coresets into additional retrieval and understanding tasks, explore the use of additional coresets, and consider the more challenging case of indexing dynamic environments.



(a) The sampling paths (blue) starting from the query node v_{end} (black) along the coreset streaming tree of the Boston data in Fig. 6-3. . Blue paths indicate individual samples according to Algorithm 8. The query node v_{end} is shown in black.

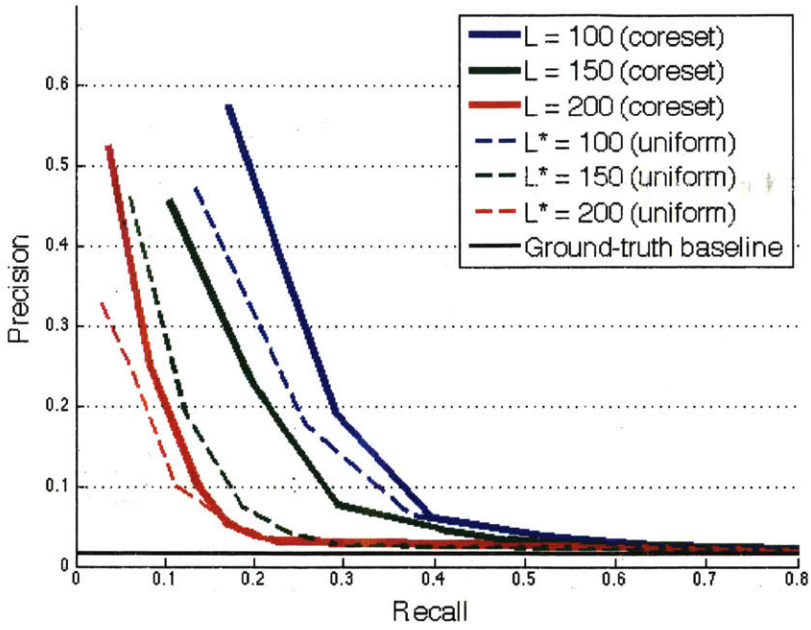


(b) Comparison of leaf sample frequency without usage of keyframes computed on the data in Fig. 6-3 for $500k$ trials (red) against sampled nodes using keyframes (blue). Intuitively, high-probability nodes correspond to segments in the video that had salient and clear sections in the video.



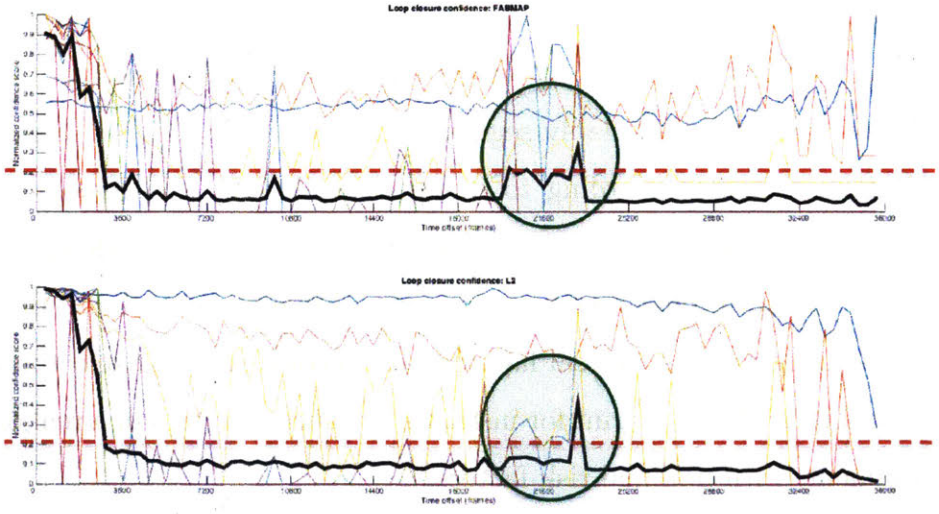
(c) Seek time histogram for the example shown in Fig. 6-4, using the tree-sampling method in Algorithms 8,9. As expected, the tree-sampling seek times drop as the number of sampled leaves increases.

Figure 6-5: Loop closure sampling algorithm results



Precision/recall plot comparing keyframes from the coreset tree (solid lines) against uniformly sampled keyframes from the video (dashed lines), as inputs for ℓ_2 loop closure. The ground-truth line represents the asymptotic precision score, i.e. $TP/(P+N)$ for a $P = 1, N = 0$ classifier.

Figure 6-6: Precision/recall plot for coreset tree vs uniform sampling



Experimental results of loop closure detection for a 3 hour video of a tour of Boston. Plots showing ℓ_2 (bottom) and FAB-MAP (top) confidence scores for increasing time offsets in the video.

Figure 6-7: Experimental results of loop closure detection

Chapter 7

Coresets for Classification – Applications to Laparoscopic and Robot-Assisted Surgery

In this chapter[1] we present a system that utilizes coresets to automatically segment and identify phases of laparoscopic and robot-assisted surgery video. We leverage the technical knowledge of expert surgeons to aggregate a feature space that best captures the axes of variability in the specific surgical video domain. Using and SVM and HMM combination we train our system using annotated ground truth surgical video, and show that we can reliably segment an unseen laparoscopic surgical video and classify the segments according to its surgical phases. We demonstrate that by using coresets we obtain results of almost identical quality, while using only a fraction of the computational memory resources. Finally, we evaluate our system experimentally using video from real laparoscopic vertical sleeve gastrectomy (LSG) procedures, and present a blueprint for instantiating such a system in a real operating room environment, with minimal risk factors.

[1]Some of the content in this chapter was published in [134].

(2) (3) (4) (5) (6)

Phases (2–6 shown) of the laparoscopic sleeve gastrectomy (LSG) procedure: (1) port, (2) biopsy, liver retraction, (3) omentum removal, dissection, hiatus inspection, (4) stapling, (5) bagging, (6) irrigation, (7) final inspection, withdrawal.

Figure 7-1: Phases of the laparoscopic sleeve gastrectomy

7.1 Problem Formulation

The objective of this study is to use coresets to create an efficient automatic real-time phase recognition and video segmentation system for laparoscopic and robot-assisted surgery; and to propose a cost-effective blueprint for piloting the system in a real operating room environment, with minimal risk factors.[2]

7.1.1 Solution Overview

We give a high-level roadmap for our technical approach and solution here:

1. Ground truth data. We build a corpus of recorded video footage and text annotations of laparoscopic procedures performed by expert surgeons. In this study we focus on the laparoscopic vertical sleeve gastrectomy (LSG) procedure, which can be performed both manually or as a robot-assisted operation.

2. Frame representation. We use the ground truth data to compose a feature space that captures the axes of variability and main discriminant factors that inform the surgical phases. We use the bag-of-words (BOW) model to represent each frame.

3. Phase prediction. We train a Support Vector Machine (SVM) for each phase of the procedure, introduce an observation function and Hidden Markov Model (HMM) to model the transitions and associated likelihoods, and use the Viterbi algorithm to

[2]This study is IRB approved for research use of surgical footage.

compute the final phase prediction.

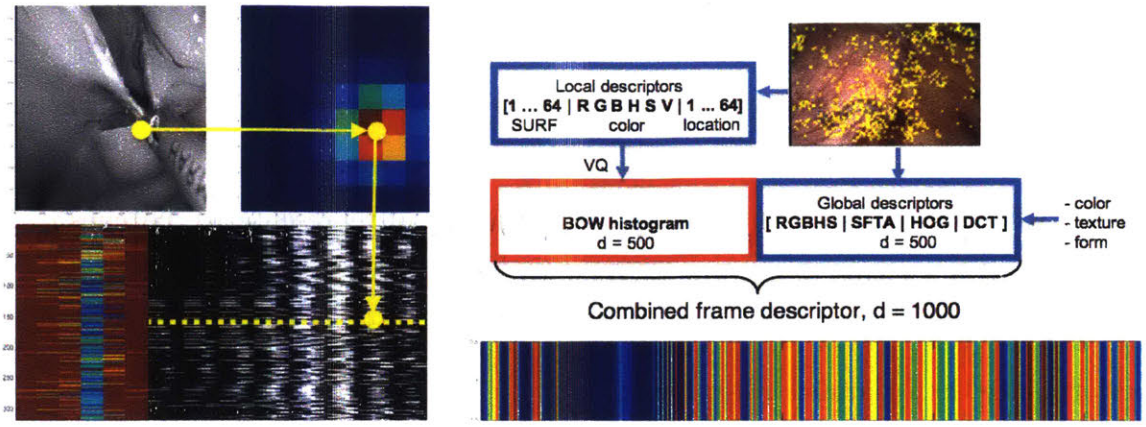
4. Coreset segmentation. As in our previous work [114, 135], we compute a coreset representation of the video instead of using the entire video. We show that using coresets gives the same level of accuracy, while providing a computationally tractable approach that enables our system to work in real-time, and using only a fraction of the computational resources.

5. Experiments. Finally, we assess our process with with cross-validation experiments, and evaluate the performance of our system against ground truth.

7.2 Technical Approach

1. Ground truth data. For this study we used 10 videos of the laparoscopic vertical sleeve gastrectomy (LSG) procedure performed by expert surgeons at the Massachusetts General Hospital. The surgeons identified 7 basic phases for this procedure to be: (1) port, (2) biopsy, liver retraction, (3) omentum removal, dissection, hiatus inspection, (4) stapling, (5) bagging, (6) irrigation, (7) final examination, withdrawal (Fig. 7-1). We note that some phases have multiple stages, and a much finer granularity is generally possible. In-fact, some very complicated procedures such as The Whipple Procedure (pancreaticoduodenectomy) can have more than a hundred identifiable phases, wherein a single misstep can result in morbidity and mortality [97]. For this study we assume that the phases always occur in the specified order. We also note that not all videos contain all the phases, which presents an additional challenge to segment videos with missing phases. We then interviewed the surgeons who performed the procedures, and collected two kinds of information: (1) qualitative annotations describing how they identified the phase from the video; (2) specific timestamps of phase transitions that serve as our ground truth segmentation.

2. Frame representation. From the video annotations we identified several visual cues, categorized broadly as local and global descriptors, that inform surgeons of the current phase. We use these cues to compose a feature space of local and global descriptors that captures the principal axes of variability and other discriminant fac-



(a) Augmented local descriptor **(b)** Augmented global descriptor

Diagram representing the augmented local and global descriptor framework, and the final frame representation using the BOW model.

Figure 7-2: Augmented descriptors for surgical video

tors that determine the phase. We now describe these visual cues, the augmented descriptor structure, and the final frame representation using the BOW model.

(i) *Color.* Histogram intersection has been used in similar work to extract color-oriented visual cues by creating a training image database of positive and negative images [79]. Other descriptor categories for individual RGBHSV channels can be utilized to increase dimensionality to discern features that depend on color in combination with some other property. Pixel values can also be used as features directly [16]. In this work, we use RGB/HSV components to augment both the local descriptor (color values) and global descriptor (color histogram).

(ii) *Position.* Relative position of organs and instruments is an important visual cue. We encode the position of SURF-detected keypoints with an 8×8 grid sampling of a Gaussian surface centered around the keypoint (Fig. 7-2a). The variance of the Gaussian defines the spatial "area of influence" of a keypoint.

(iii) *Shape.* Shape is important for detecting instruments, which are some of most obvious visual cues for identifying the phase. Shape can be encoded with various techniques, such as the Viola-Jones object detection framework [133], using image segmentation to isolate the instruments and match against artificial 3D models [127], and other methods. For local frame descriptors we use the standard SURF descriptor

as a base, and for global frame descriptor we add grid-sampled HOG descriptors [34] and DCT coefficients [33].

(iv) *Texture.* Texture is a crucial visual cue to distinguish vital organs, which tend to exhibit a narrow variety of color. Texture can be extracted using a co-occurrence matrix with Haralick descriptors [80], by a sampling of representative patches to be evaluated with a visual descriptor vector for each patch [79], and other methods. In this work, we use the newer SFTA texture descriptor [28], which has shown better performance than Haralick filter banks.

(v) *Temporal Regularity.* Time is a very valuable non-visual cue. Hidden Markov Models (HMM), with the states corresponding to the surgical phases, and Dynamic Time Warping (DTW) are commonly utilized to build a model of the "average" surgery that captures the underlying semantics [79, 16]. We use a simple low-pass filtering approach to construct an observation function that relies on the inherent regularity of surgical phases, and combine this with HMM to produce the final phase prediction.

Finally, we combine the augmented local and global descriptors into a single fixed-dimension frame descriptor. For this we use the bag-of-words (BOW) model, which is is a simplifying representation commonly used to standardize the dimensionality of features [126]. We compute a representative vector quantization (VQ) by sampling frames using local descriptors only. Any set of local descriptors (Fig. 7-2a, bottom) can then be represented as a histogram of projections in the fixed VQ dimension ($d_1 = 500$). The final frame descriptor is then composed of the BOW histogram of augmented local descriptors and the additional dimensions ($d_2 = 500$) of the global descriptor (Fig. 7-2b, top), for a combined dimension $d = 1000$ (Fig. 7-2b, bottom).

3. Phase prediction. As a first step, we train a series of support vector machines for each phase. Each SVM classifies a phase i by outputting a binary variable $p_i = 1$, $P \backslash \{p_i\} = 0$. This approach was shown to be more accurate than a single multi-class SVM in a similar visual domain [80]. This is an iterative step that involves interviewing surgeons, re-calibrating the feature space, re-training the classifiers, etc. Interviewing surgeons is expensive and time-consuming therefore it is important to

repeat this step first until we achieve the desired level of accuracy. The first step is to ensure that the augmented feature space presented in Section 7.2 yields an acceptable level of accuracy for this domain with respect to the ground truth phases. Fig. 7-3a shows the binary outputs produced by the SVM. Fig. 7-3b shows the rate of correct classification (accuracy) of the predictions compared against ground truth. We also note that there are two ambiguous cases: (i) multiple classifiers identify phase $y_i(t) = 1$; and (ii) every SVM outputs $y_i(t) = 0$.

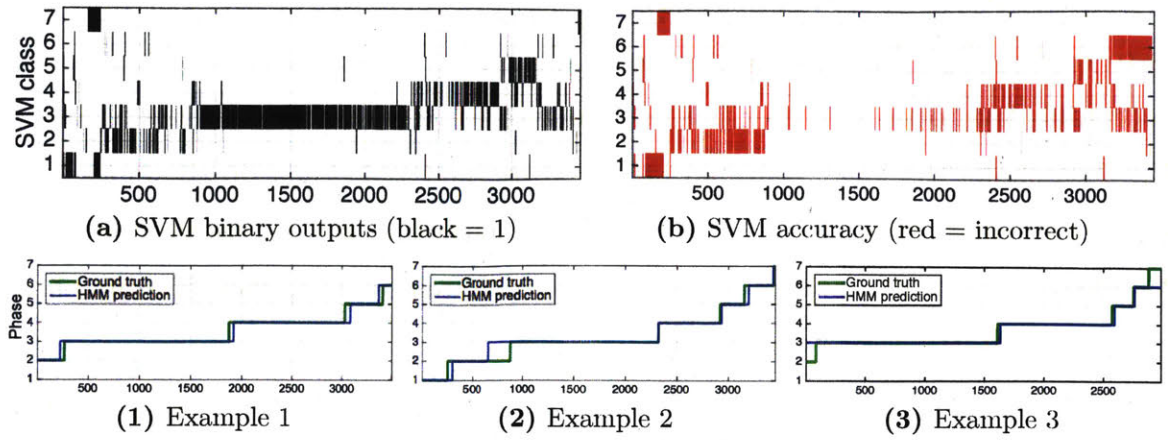
The second step is to make use of the temporal structure of surgical phases (monotonically increasing, MECE) to correct SVM predictions, resolve the ambiguous cases stated above, and compute a single time-series of phase predictions. We achieve this using an observation function $\phi(V, s; \alpha, \beta)$ that takes a sequence of SVM outputs V, the current state prediction s, a certainty parameter α, and lookback value β, and returns the next phase prediction p. Phase transitions are modeled using an HMM with the left-right restriction as in [79]. The final observation sequence $Q = p_1, \ldots, p_N$ is the emission sequence. Finally, we run the Viterbi algorithm [55] on the emission sequence to find the most likely sequence of hidden states (the phases).

4. Coreset segmentation. Coresets are compact data reduction constructs that can efficiently produce a problem dependent compression of the data. As in our previous works, we use an online k-segment coreset algorithm to compute an approximate segmentation of the video stream [114], and construct a keyframe compression of the video based on this segmentation [135]. Using coresets allows our system to run online, in real-time, using minimal computational resources.

5. Experiments. We assess our system with cross-validation experiments, using both the entire video and the coreset representation, and evaluate accuracy against ground truth segmentation. We present results in the next section.

7.3 Results

We perform cross-validation tests by training the system on each subset of $N-1=9$ videos in the dataset, using a standard 80/20 training/validation split. The system



(a) SVM binary outputs (black = 1)

(b) SVM accuracy (red = incorrect)

(1) Example 1 **(2)** Example 2 **(3)** Example 3

Typical results (x = frame number). (7-3a–7-3b) highlights the shortfalls of SVM-only system with no temporal component. (1–3) shows HMM-corrected predictions.

Figure 7-3: Surgical phase prediction results

is then tested on each remaining unseen video, and the results aggregated over the N subsets. The observation function parameters were determined empirically (typical values are $\alpha = 0.8$, $\beta = 15$). The HMM transition and emission matrices learned from the data, similar to [79]. The Viterbi algorithm was run on the emission sequence to compute the final phase prediction given the outputs of the observation function. Fig. 7-3.1–7-3.3 shows typical experimental results. We demonstrate a 90.4% SVM prediction accuracy, and improve to 92.8% when combined with HMM. These results are on par with similar work in the surgical video domain [81], while achieving a 90+% coreset compression over the original video stream.

7.3.1 Technical Summary

- Used surgeons' expertise to build a corpus of ground truth segmentation of LSG surgical video.
- Fine-tuned a feature space that captures the principal axes of variability and other visual discriminant factors that inform the surgical phases.
- Used SVM, HMM for classification and prediction.
- Achieved average prediction accuracy 92% , as good or better than related work.

- Achieved 90%+ compression using coreset summary, while maintaining the same performance accuracy.

7.4 Conclusions

In this chapter we presented a system for automatically identifying the phases of laparoscopic and robot-assisted surgical procedures and segmenting them in real-time. We received a set of laparoscopic sleeve gastrectomy (LSG) videos, and accompanying ground truth segmentation. We used the surgeons' expert annotations to tailor a visual feature space that captures the main visual factors that the surgeons rely on to determine the current phase of the procedure. We trained a set of SVMs to classify each of the surgical phases independently, and then used an HMM to compute the phase predictions. We demonstrated the effectiveness of our system on unseen surgical video, achieving 92%+ prediction accuracy and a 90%+ compression by using the coreset summary.

Chapter 8

Coresets for Dimensionality Reduction of Stationary Data

In this chapter[1] we present an efficient construction for a coreset for principal component analysis (PCA) that is both small in size and a subset of the original data.

We present the first algorithm for computing a (k, ε)-coreset C of size independent of both n and d, for any given $n \times d$ input matrix. The algorithm takes as input a finite set of d-dimensional vectors, a desired approximation error ε, and an integer $k \geq 0$, and computes a weighted subset (coreset) C of k^2/ε^2 such vectors. This coreset can be used to approximate the sum of squared distances from the matrix $A \in \mathbb{R}^{n \times d}$, whose rows are the n vectors seen so far, to any k-dimensional affine subspace in $\mathbb{R}^d$, up to a factor of $1 \pm \varepsilon$. For a (possibly unbounded) stream of such input vectors the coreset can be maintained at the cost of an additional factor of $\log^2 n$. If each row of an input matrix A has $O(\text{nnz})$ non-zero entries, then the update time per insertion, the overall memory that is used by our algorithm, and the low rank approximation of the coreset C is $O(\text{nnz} \cdot k^2/\varepsilon^2)$, i.e. independent of n and d.

Key contributions

The main contributions of this chapter are:

[1] Some of the content in this chapter was published in [54].

(i) A new algorithm for dimensionality reduction of sparse data that uses a weighted subset of the data, and is independent of both the size and dimensionality of the data. (ii) An efficient algorithm for computing such a reduction, with provable bounds on size and running time. (iii) A system that implements this dimensionality reduction algorithm and an application of the system to compute principal component analysis (PCA) of the entire English Wikipedia.

Chapters 8 and 9 are organized as follows. Section 8.1 sets up the problem formulation, establishes the definition of coreset as used in this work, and sets up the main existence theorem and two supporting lemmas that form the burden of proof for the rest of the work. .Section 8.1 concludes by using these results to prove the main result of the work. Sections 8.3 and 8.4 contain the definitions and theorems to prove the main results of this work. In the next chapter we describe our system implementation and in Section 9.1 we present experimental results.

8.1 Problem Formulation

Given a matrix A, a coreset C in this work is defined as a weighted subset of rows of A such that the sum of squared distances from any given k-dimensional subspace to the rows of A is approximately the same as the sum of squared weighted distances to the rows in C. Formally,

For a compact set $S \in \mathbb{R}^d$ and a vector x in $\mathbb{R}^d$, we denote the Euclidean distance between x and its closest points in S by

$$\text{dist}^2(x, S) := \min_{s \in S} \|x - s\|_2^2$$

For an $n \times d$ matrix A whose rows are $a_1, \ldots, a_n$, we define the sum of the squared distances from A to S by

$$\text{dist}^2(A, S) := \sum_{i=1}^{n} \text{dist}^2(a_i, S)$$

Definition 7 ((k,ε)-coreset). *Given a $n \times d$ matrix A whose rows $a_1, \cdots, a_n$ are n points (vectors) in $\mathbb{R}^d$, an error parameter $\varepsilon \in (0,1]$, and an integer $k \in [1, d-1] = \{1, \cdots, d-1\}$ that represents the desired dimensionality reduction, n (k,ε)-coreset for A is a weighted subset $C = \{w_i a_i \mid w_i > 0 \text{ and } i \in [n]\}$ of the rows of A, where $w = (w_1, \cdots, w_n) \in [0,\infty)^n$ is a non-negative weight vector, such that for every affine k-subspace S in $\mathbb{R}^d$ we have*

$$\left|\text{dist}^2(A,S)) - \text{dist}^2(C,S))\right| \leq \varepsilon \, \text{dist}^2(A,S)). \tag{8.1}$$

That is, the sum of squared distances from the n points to S approximates the sum of squared weighted distances $\sum_{i=1}^{n} w_i^2 (\text{dist}(a_i, S))^2$ to S. The approximation is up to a multiplicative factor of $1 \pm \varepsilon$. By choosing $w = (1, \cdots, 1)$ we obtain a trivial $(k, 0)$-coreset. However, in a more efficient coreset most of the weights will be zero and the corresponding rows in A can be discarded. The cardinality of the coreset is thus the sparsity of w, given by $|C| = \|w\|_0 := |\{w_i \neq 0 \mid i \in [n]\}|$.

If C is small, then the computation is efficient. Because C is a weighted subset of the rows of A, if A is sparse, then C is also sparse. A long-open research question has been whether we can have such a coreset that is both *of size* independent of the input dimension (n and d) and a *subset of the original input rows.*

8.2 Technical Solution

Given a $n \times d$ matrix A, we propose a construction mechanism for a matrix C of size $|C| = O(k^2/\varepsilon^2)$ and claim that it is a (k,ε)-coreset for A. We use the following corollary for Definition 7 of a coreset, based on simple linear algebra that follows from the geometrical definitions (e.g. see [48]).

Property 1 (Coreset for sparse matrix). *Let $A \in \mathbb{R}^{n \times d}$, $k \in [1, d-1]$ be an integer, and let $\varepsilon > 0$ be an error parameter. For a diagonal matrix $W \in \mathbb{R}^{n \times n}$, the matrix $C = WA$ is a (k,ε)-coreset for A if for every matrix $X \in \mathbb{R}^{d \times (d-k)}$ such that $X^T X =$*

I, we have

$$\text{(i)} \quad \left|1 - \frac{\|WAX\|}{\|AX\|}\right| \leq \varepsilon, \quad \text{and} \qquad \text{(ii)} \quad \|A - WA\| < \varepsilon \operatorname{var}(A) \tag{8.2}$$

where $\operatorname{var}(A)$ *is the sum of squared distances from the rows of* A *to their mean.*

The goal of this work is to prove that such a coreset (Definition 7) exists for any matrix A (Property 1) and can be computed efficiently. Formally,

Theorem 7. *For every input matrix* $A \in \mathbb{R}^{n \times d}$*, an error* $\varepsilon \in (0, 1]$ *and an integer* $k \in [1, d-1]$*:*

(a) there is a (k, ε)*-coreset* C *of size* $|C| = O(k^2/\varepsilon^2)$*;*

(b) such a coreset can be constructed in $O(k^2/\varepsilon^2)$ *time.*

Theorem 7 is the formal statement for the main technical contribution of this work. Sections 8.2–8.4 constitute a proof for Theorem 7.

To establish Theorem 7(a), we first state our two main results (Theorems 8 and 9) axiomatically, and show how they combine such that Property 1 holds. Thereafter we prove the these results in Sections 8.3 and 8.4, respectively. To prove Theorem 7(b) (efficient construction) we present an algorithm for computing a matrix C, and analyze the running time to show that the C can be constructed in $O(k^2/\varepsilon^2)$ iterations.

Let $A \in \mathbb{R}^{n \times d}$ be a matrix of rank d, and let $U\Sigma V^T = A$ denote its full SVD. Let $W \in \mathbb{R}^{n \times n}$ be a diagonal matrix. Let $k \in [1, d-1]$ be an integer. For every $i \in [n]$ let

$$v_i = \left(U_{i,1}, \cdots, U_{i,k}, \frac{U_{i,k+1:d}\Sigma_{k+1:d,k+1:d}}{\|\Sigma_{k+1:d,k+1:d}\|}, 1\right). \tag{8.3}$$

Then the following two results hold:

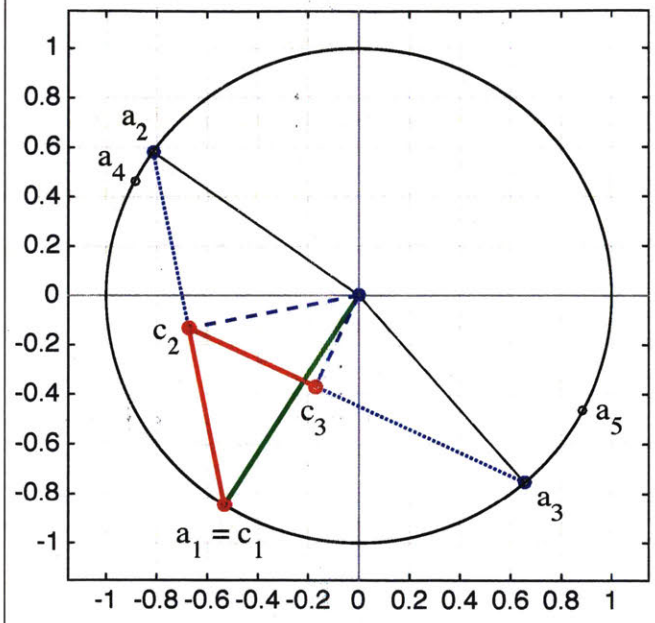
Theorem 8 (Coreset for sum of vectors)**.** *For every set of of* n *vectors* $v_1, \cdots, v_n$ *in* $\mathbb{R}^d$ *and every* $\varepsilon \in (0, 1)$*, a weight vector* $w \in (0, \infty)^n$ *of sparsity* $\|w\|_0 \leq 1/\varepsilon^2$ *can be computed deterministically in* $O(nd/\varepsilon)$ *time such that*

$$\left\| \sum_{i=1}^{n} v_i - \sum_{i=1}^{n} w_i v_i \right\| \leq \varepsilon \sum_{i=1}^{n} \|v_i\|^2. \tag{8.4}$$

Algorithm 10 SUMVECSCORESET(A, ε)

Input: A: n input points $a_1, \ldots, a_n$ in $\mathbb{R}^d$
Input: $\varepsilon \in (0,1)$: the approximation error
Output: $w \in [0,\infty)^n$: non-negative weights
$A \leftarrow A - \text{mean}(A)$
$A \leftarrow cA$ where c is a constant s.t. $\text{var}(A) =$
$w \leftarrow (1, 0, \ldots, 0)$
$j \leftarrow 1$, $p \leftarrow A_j$, $J \leftarrow \{j\}$
$M_j = \{y^2 \mid y = A \cdot A_j^T\}$
for $i = 1, \ldots, n$ **do**
  $j \leftarrow \text{argmin}\{w_J \cdot M_J\}$
  $G \leftarrow W' \cdot A_J$ where $W'_{i,i} = \sqrt{w_i}$
  $\|c\| = \|G^T G)\|_F^2$
  $c \cdot p = \sum_{i=1}^{|J|} G p^T$
  $\|c - p\| = \sqrt{1 + \|c\|^2 - c \cdot p}$
  $\text{comp}_p(v) = 1/\|c - p\| - (c \cdot p)/\|c - p\|$
  $\|c - c'\| = \|c - p\| - \text{comp}_p(v)$
  $\alpha = \|c - c'\|/\|c - p\|$
  $w \leftarrow w(1 - |\alpha|)$
  $w_j \leftarrow w_j + \alpha$
  $w \leftarrow w / \sum_{i=1}^n w_i$
  $M_j \leftarrow \{y^2 \mid y = A \cdot A_j^T\}$
  $J \leftarrow J \cup \{j\}$
  if $\|c\|^2 \le \varepsilon$ **then**
    break
  end if
end for
return w

(a) Coreset for sum of vectors



(b) Illustration showing first 3 steps of the algorithm

Fig. 8-1b shows the first 3 steps of Algorithm 10 . Given n input points $a_1,\ldots,a_n$ in $\mathbb{R}^d$, with weighted mean $\sum_i z_i a_i = \mathbf{0}$, pick an arbitrary starting point $a_1 = c_1$. At each step find the farthest point a_{j+1} from c_j, and compute c_{j+1} by projecting the origin onto the line segment $[c_j, a_{j+1}]$. Repeat this for $j = 1,\ldots,N$ iterations, where $N = 1/\varepsilon^2$. The output weight vector $w \in D^n$ satisfies $c_N = \sum_{i=1}^n w_i a_i$.

Section 8.3 establishes a proof for Theorem 8.

Theorem 9 (Coreset for Low rank approximation). *For every* $X \in \mathbb{R}^{d \times (d-k)}$ *such that* $X^T X = I$,

$$\left| 1 - \frac{\|WAX\|^2}{\|AX\|^2} \right| \le 5 \left\| \sum_{i=1}^n v_i v_i^T - W_{i,i} v_i v_i^T \right\|. \tag{8.5}$$

Section 8.4 establishes a proof for Theorem 9.

8.2.1 Proof of Theorem 7

Proof of Theorem 7(a). Replacing v_i with $v_i v_i^T$ and ε by $\varepsilon/(5d)$ in Theorem 8 yields

$$\left\| \sum_i v_i v_i^T - W_{i,i} v_i v_i^T \right\| \leq (\varepsilon/5d) \sum_{i=1}^{n} \left\| v_i v_i^T \right\|^2 .$$

Combining this inequality with (8.4) gives

$$\left| 1 - \frac{\|WAX\|^2}{\|AX\|^2} \right| \leq 5 \left\| \sum_{i=1}^{n} v_i v_i^T - W_{i,i} v_i v_i^T \right\| \leq (\varepsilon/5d) \sum_{i=1}^{n} \left\| v_i v_i^T \right\|^2 .$$

Thus the left-most term is bounded by the right-most term, which proves (8.2). This also means that $C = WA$ is a coreset for k-SVD, i.e., (non-affine) k-dimensional subspaces. To support PCA (affine subspaces) the coreset $C = WA$ needs to satisfy the expression in the last line of Property 1 regarding its mean. This holds using the last entry (one) in the definition of v_i (8.3), which implies that the sum of the rows is preserved as in equation (8.4). Therefore Property 1 holds for $C = WA$, which proves Theorem 7(a).

Claim Theorem 7(b) follows from simple analysis of Algorithm 11 that implements this construction. □

8.3 Coreset for Sum of Vectors ($k = 0$)

In order to prove the general result Theorem 7(a), that is the existence of a (k, ε)-coreset for any $k \in [1, d-1]$, we first establish the special case for $k = 0$. In this section, we prove Theorem 8 by providing an algorithm for constructing a small weighted subset of points that constitutes a general approximation for the sum of vectors.

To this end, we first introduce an intermediate result that shows that given n points on the unit ball with weight distribution z, there exists a small subset of points whose weighted mean is approximately the same as the weighted mean of the original points.

Let D^n denote the union over every vector $z \in [0,1]^n$ that represent a distribution, i.e., $\sum_i z_i = 1$. Our first technical result is that for any finite set of unit vectors $a_1, \ldots, a_n$ in $\mathbb{R}^d$, any distribution $z \in D^n$, and every $\varepsilon \in (0,1]$, we can compute a sparse weight vector $w \in D^n$ of sparsity (non-zeroes entries) $\|w\|_0 \leq 1/\varepsilon^2$.

Lemma 7. *Let $z \in D^n$ be a distribution over n unit vectors $a_1, \cdots, a_n$ in $\mathbb{R}^d$. For $\varepsilon \in (0,1)$, a sparse weight vector $w \in D^n$ of sparsity $s \leq 1/\varepsilon^2$ can be computed in $O(nd/\varepsilon^2)$ time such that*

$$\left\| \sum_{i=1}^{n} z_i \cdot a_i - \sum_{i=2}^{n} w_i \, a_i \right\|_2 \leq \varepsilon. \tag{8.6}$$

We note that the Caratheodory Theorem [20] proves Lemma 7 for the special case $\varepsilon = 0$ using only $d+1$ points. Our approach and algorithm can thus be considered as an ε-approximation for the Caratheodory Theorem, to get coresets of size independent of d. Note that our Frank-Wolfe-style algorithm might run more than $d+1$ or n iterations without getting zero error, since the same point may be selected in several iterations. Computing in each iteration the closest point to the origin that is *spanned* by all the points selected in the previous iterations, would guarantee coresets of size at most $d+1$, and fewer iterations. Of course, the computation time of each iteration will also be much slower. '

Proof. We assume that $\sum_i z_i a_i = \mathbf{0}$, otherwise we subtract $\sum_j z_j a_j$ from each input vector a_i. We also assume $\varepsilon < 1$, otherwise the claim is trivial for $w = \mathbf{0}$. Let $w \in D^n$ such that $\|w\|_0 = 1$, and denote the current mean approximation by $c = \sum_i w_i a_i$. Hence, $\|c\|_2 = \|a_i\| = 1$.

The following iterative algorithm updates c in the end of each iteration until $\|c\|_2 < \varepsilon$. In the beginning of the Nth iteration the squared distance from c to the mean (origin) is

$$\|c\|_2^2 \in [\varepsilon, \frac{1}{N}]. \tag{8.7}$$

The average distance to c is thus

$$\sum_i z_i \|a_i - c\|_2^2 = \sum_i z_i \|a_i\|_2^2 + 2c^T \sum_i z_i a_i + \sum_i z_i \|c\|_2^2 = 1 + \|c\|_2^2 \geq 1 + \varepsilon,$$

where the sum here and in the rest of the proof are over $[n]$. Hence there must be a $j \in [n]$ such that

$$\|q_j - c\|_2^2 \geq 1 + \varepsilon. \tag{8.8}$$

Let r be the point on the segment between a_j and c at a distance $\rho := 1/\|a_j - c\|_2$ from a_j. Since $\|a_j - r\|_2 = \rho = \rho\|a_j - \mathbf{0}\|_2$, and $\|a_j - \mathbf{0}\|_2 = 1 = \rho\|a_j - c\|_2$, and $\angle(\mathbf{0}, a_j, c) = \angle(c, a_j, \mathbf{0})$, the triangle whose vertices are a_j, r and $\mathbf{0}$ is similar to the triangle whose vertices are a_j, $\mathbf{0}$, and c with a scaling factor of ρ. Therefore,

$$\|r - \mathbf{0}\|_2 = \rho \cdot \|\mathbf{0} - c\|_2 = \frac{\|c\|_2}{\|q_j - c\|_2}. \tag{8.9}$$

From (8.8) and (8.9), by letting c' be the closest point to $\mathbf{0}$ on the segment between a_j and c, we obtain

$$\|c'\|_2^2 \leq \|r\|_2^2 = \frac{\|c\|_2^2}{\|a_j - c\|_2^2} \leq \frac{\|c\|_2^2}{1+\varepsilon}.$$

Combining this with (8.7) yields

$$\|c'\|_2^2 \leq \frac{\frac{1}{N}}{1+\varepsilon} \leq \frac{\frac{1}{N}}{1+\frac{1}{N}} = \frac{1}{N+1}.$$

Since c' is a convex combination of a_j and c, there is $\alpha \in [0,1]$, such that $c' = \alpha a_j + (1-\alpha)c$. Therefore,

$$c' = \alpha a_j + (1-\alpha)\sum_i w_i a_i$$

and thus we have $c' = \sum_i w'_i a_i$, where $w' = (1-\alpha)w + \alpha e_j$, and $e_j \in D^n$ is the jth standard vector. Hence, $\|w'\|_0 = N+1$. If $\|c'\|_2^2 < \varepsilon$ the algorithm returns c'. Otherwise

$$\|c'\|_2^2 \in [\varepsilon, \frac{1}{N+1}] \tag{8.10}$$

We can repeat the procedure in (8.7) with c' instead of c and $N+1$ instead of N. By (8.10) $N+1 \leq 1/\varepsilon$ so the algorithm ends after $N \leq 1/\varepsilon$ iterations. After the last iteration we return the center $c' = \sum_{i=1}^{n} w'_i a_i$ so

$$\left\| \sum_i (z_i - w'_i) a_i \right\|_2^2 = \|c'\|_2^2 \leq \frac{1}{N+1} \leq \varepsilon.$$

□

We prove Theorem 8 by providing a computation of such a sparse weight vector w. The intuition for this computation is as follows. Given n input points $a_1,\ldots,a_n$ in $\mathbb{R}^d$, with weighted mean $\sum_i z_i\, a_i = \mathbf{0}$, we project all the points on the unit sphere. Pick an arbitrary starting point $a_1 = c_1$. At each step find the farthest point a_{j+1} from c_j, and compute c_{j+1} by projecting the origin onto the line segment $[c_j, a_{j+1}]$. Repeat this for $j = 1,\ldots,N$ iterations, where $N = 1/\varepsilon^2$. We prove that $\|c_i\|^2 = 1/i$, thus if we iterate $1/\varepsilon^2$ times, this norm will be $\left\|c_{1/\varepsilon^2}\right\| = \varepsilon^2$. The resulting points c_i are a weighted linear combination of a small subset of the input points. The output weight vector $w \in D^n$ satisfies $c_N = \sum_{i=1}^{n} w_i\, a_i$, and this weighted subset forms the coreset.

Proof of Theorem 8. The proof of Theorem 8 follows by applying Lemma 7 after normalization of the input points and then post-processing the output. □

Algorithm 10 contains the pseudocode for SUMVECSCORESET. Fig. 8-1b illustrates the first steps of the main computation (lines 9–26).

8.3.1 Analysis of Algorithm 10

Algorithm 10 contains the full listing of the construction algorithm for the coreset for sum of vectors.

Input: A: n input points $a_1, \ldots, a_n$ in $\mathbb{R}^d$; $\varepsilon > 0$: the nominal approximation error.

Output: a non-negative vector $w \in [0, \infty)^n$ of only $O(1/\varepsilon^2)$ non-zeros entries which are the non-negative weights of the corresponding points selected for the coreset.

Analysis: The first step is to translate and scale the input points such that the mean is zero and the variance is 1 (lines 4–5). After initialization (lines 6–8), we begin the main iterative steps of the algorithm. First we find the index j of the farthest point from the initial point a_1. The next point added to the coreset is denoted by $p = a_j$. Next we compute $\|c - p\|$, the distance from the current point p to the previous center c. In order to do this we compute $G = W' \cdot A_J$ where J is the set of all previously added indices j, starting with the first point, and W' is defined in line 11. Note that G also gives us the error of the current iteration, $\varepsilon = \text{trace}(GG^T)$ (line 23). Next we find the point c' on the line from c to p that is closest to the origin, and find the distance between the current center c and the new center c' (lines 12–16). Finally, the ratio of distances between the current center, farthest point, and new center give us a value for α, the amount by which we update the coreset weights (lines 17–20).

The algorithm then updates the recorded indices J, update the lookup table M of previously computed row inner products for subsequent iterations, and repeat lines 10–26 until the loop terminates. The terminating conditions depend on the system specification – we may wish to bound the error, or the number of iterations. Moreover, if the update value α is below a specified threshold, we may also terminate the loop if such threshold is lower than a desired level of accuracy.

8.4 Coreset for Low Rank Approximation ($k > 0$)

In Section 8.3 we presented a new coreset construction for approximating the sum of vectors, showing that given n points on the unit ball there exists a small weighted subset of points that is a coreset for those points. In this section we describe the reduction of Algorithm 10 for $k = 0$ to an efficient algorithm for any low rank approximation with $k \in [1, d-1]$.

Conceptually, we achieve this reduction in two steps. The first step is to show that Algorithm 10 can be reduced to an inefficient computation for low rank approximation for matrices. To this end, we first prove Theorem 9, thus completing the existence clause Theorem 7(a).

Algorithm 11 LOWRANKCORESET(A, k, ε)

Input: A: A sparse $n \times d$ matrix
Input: $k \in \mathbb{Z}_{>0}$: the approximation rank
Input: $\varepsilon \in \left(0, \frac{1}{2}\right)$: the approximation error
Output: $w \in [0, \infty)^n$: non-negative weights
Compute $U\Sigma V^T = A$, the SVD of A
$R \leftarrow \Sigma_{k+1:d,k+1:d}$
$P \leftarrow$ matrix whose i-th row $\forall i \in [n]$ is
  $P_i = (U_{i,1:k}, U_{i,k+1:d} \cdot \frac{R}{\|R\|_F})$
$X \leftarrow$ matrix whose i-th row $\forall i \in [n]$ is
  $X_i = P_i / \|P_i\|_F$
$w \leftarrow (1, 0, \ldots, 0)$

(a) 1/2: Initialization

for $i = 1, \ldots, \lceil k^2/\varepsilon^2 \rceil$ **do**
  $j \leftarrow \text{argmin}_{i=1,\ldots,n}\{wXX_i\}$
  $a = \sum_{i=1}^n w_i (X_i^T X_j)^2$
  $b = \dfrac{1 - \|PX_j\|_F^2 + \sum_{i=1}^n w_i \|PX_i\|_F^2}{\|P\|_F^2}$
  $c = \|wX\|_F^2$
  $\alpha = (1 - a + b) \,/\, (1 + c - 2a)$
  $w \leftarrow (1 - \alpha) I_j + \alpha w$
end for
return w

(b) 2/2: Computation

Coreset for low rank approximation. Using MATLAB notation, we denote by $j{:}k$ the set of indices $\{j, j+1, \ldots, k\}$ for an integer $k \geq j$. The i-th row and j-th column of X are denoted by $X_{i,:} \in \mathbb{R}^{1 \times d}$ and $X_{:,j} \in \mathbb{R}^{n \times 1}$, respectively. The i-th entry of a vector $x = (x_1, \cdots, x_d)$ is denoted by x_i. The Frobenius norm (root of squared entries) of a matrix or a vector X is denoted by $\|X\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^d X_{i,j}^2}$. The identity matrix is denoted by I and the matrix of all zeros whose entries are all zeroes is denoted by $\mathbf{0}$.

Proof of Theorem 9. Let $\varepsilon = \left\| \sum_{i=1}^n (1 - W_{i,i}^2) v_i v_i^T \right\|$. For every $i \in [n]$ let $t_i = 1 - W_{i,i}^2$. Set $X \in \mathbb{R}^{d \times (d-k)}$ such that $X^T X = I$. Without loss of generality we assume $V^T = I$, i.e. $A = U\Sigma$, otherwise we replace X by $V^T X$. It thus suffices to prove that

$$\left| \sum_i t_i \left\| A_{i,:} X \right\|^2 \right| \leq 5\varepsilon \left\| AX \right\|^2 . \tag{8.11}$$

Using the triangle inequality, we get

$$\left| \sum_i t_i \left\| A_{i,:} X \right\|^2 \right| \leq \left| \sum_i t_i \left\| A_{i,:} X \right\|^2 - \sum_i t_i \left\| (A_{i,1:k}, \mathbf{0}) X \right\|^2 \right| \tag{8.12}$$

$$+ \left| \sum_i t_i \left\| (A_{i,1:k}, 0) X \right\|^2 \right| . \tag{8.13}$$

We complete the proof by deriving bounds on (8.12) and (8.13).

Bound on (8.12): It was proven in [1] that for every pair of k-subspaces S_1, S_2 in $\mathbb{R}^d$ there is $u \geq 0$ and a $(k-1)$-subspace $T \subseteq S_1$ such that the distance from every point $p \in S_1$ to S_2 equals to its distance to T multiplied by u. By letting S_1 denote the k-subspace that is spanned by the first k standard vectors of $\mathbb{R}^d$, letting S_2 denote

the k-subspace that is orthogonal to each column of X, and $y \in \mathbb{R}^k$ be a unit vector that is orthogonal to T, we obtain that for every row vector $p \in \mathbb{R}^k$,

$$\|(p, \mathbf{0})X\|^2 = u^2(py)^2. \tag{8.14}$$

After defining $x = \Sigma_{1:k,1:k}y / \|\Sigma_{1:k,1:k}y\|$, (8.12) is bounded by

$$\begin{aligned}
&\sum_i t_i \|(A_{i,1:k}, \mathbf{0})X\|^2 = \sum_i t_i \cdot u^2 \|A_{i,1:k}y\|^2 \\
&= u^2 \sum_i t_i \|A_{i,1:k}y\|^2 \\
&= u^2 \sum_i t_i \|U_{i,1:k}\Sigma_{1:k,1:k}y\|^2 \\
&= u^2 \|\Sigma_{1:k,1:k}y\|^2 \sum_i t_i \|(U_{i,1:k})x\|^2 .
\end{aligned} \tag{8.15}$$

The left side of (8.15) is bounded by substituting $p = \Sigma_{j,1:k}$ in (8.14) for $j \in [k]$, as

$$\begin{aligned}
&u^2 \|\Sigma_{1:k,1:k}y\|^2 = \sum_{j=1}^{k} u^2(\Sigma_{j,1:k}y)^2 = \sum_{j=1}^{k} \|(\Sigma_{j,1:k}, \mathbf{0})X\|^2 \\
&= \sum_{j=1}^{k} \sigma_j^2 \|X_{j,:}\|^2 \leq \sum_{j=1}^{d} \sigma_d^2 \|X_{j,:}\|^2 \\
&= \|\Sigma X\|^2 = \|U\Sigma X\|^2 = \|AX\|^2 .
\end{aligned} \tag{8.16}$$

The right hand side of (8.15) is bounded by

$$\left| \sum_i t_i \|(U_{i,1:k})x\|^2 \right| = \left| \sum_i t_i (U_{i,1:k})^T U_{i,1:k} \cdot xx^T \right| = \left| xx^T \cdot \sum_i t_i (U_{i,1:k})^T U_{i,1:k} \right|$$

$$\leq \left\|xx^T\right\| \cdot \left\|\sum_i t_i (U_{i,1:k})^T U_{i,1:k}\right\| \tag{8.17}$$

$$\leq \left\|\sum_i t_i (v_{i,1:k})^T v_{i,1:k}\right\| \leq \left\|\sum_i t_i v_i^T v_i\right\| = \varepsilon \tag{8.18}$$

where (8.17) is by the Cauchy-Schwartz inequality and the fact that $\left\|xx^T\right\| = \|x\|^2 = 1$, and in (8.18) we used the assumption $A_{i,j} = U_{i,j}\sigma_j = v_{i,j}$ for every $j \in [k]$.

Plugging (8.16) and (8.18) in (8.15) bounds (8.12) as

$$|\sum_i t_i \left\|(A_{i,1:k}, \mathbf{0})X\right\|^2| \leq \varepsilon \left\|AX\right\|^2 . \tag{8.19}$$

Bound on (8.13)**:** For every $i \in [n]$ we have

$$\begin{aligned}
&\left\|A_{i,:}X\right\|^2 - \left\|(A_{i,1:k}, \mathbf{0})X\right\|^2 \\
&= 2(A_{i,1:k}, \mathbf{0})XX^T(\mathbf{0}, A_{i,k+1:d})^T + \left\|(\mathbf{0}, A_{i,k+1:d})X\right\|^2 \\
&= 2A_{i,1:k}X_{1:k,:}(X_{k+1:d,:})^T(A_{i,k+1:d})^T + \left\|(\mathbf{0}, A_{i,k+1:d})X\right\|^2 \\
&= 2\sum_{j=1}^{k} A_{i,j}X_{j,:}(X_{k+1:d,:})^T(A_{i,k+1:d})^T + \left\|(\mathbf{0}, A_{i,k+1:d})X\right\|^2 \\
&= \sum_{j=1}^{k} 2\sigma_j X_{j,:}(X_{k+1:d,:})^T \cdot \left\|\sigma_{k+1:d}\right\| v_{i,j}(v_{i,k+1:d})^T + \\
&\quad \left\|\sigma_{k+1:d}\right\|^2 \left\|(\mathbf{0}, v_{i,k+1:d})X\right\|^2 .
\end{aligned} \tag{8.20}$$

Summing this over $i \in [n]$ with multiplicative weight t_i and using the triangle

inequality, will bound (8.13) by

$$\left|\sum_i t_i \|A_{i,:}X\|^2 - \sum_i t_i \|(A_{i,1:k},\mathbf{0})X\|^2\right|$$

$$\leq \Big|\sum_i t_i \sum_{j=1}^{k} 2\sigma_j X_{j,:}(X_{k+1:d,:})^T \tag{8.21}$$

$$\cdot \|\sigma_{k+1:d}\| v_{i,j}(v_{i,k+1:d})^T\Big|$$

$$+\left|\sum_i t_i \|\sigma_{k+1:d}\|^2 \|(\mathbf{0}, v_{i,k+1:d})X\|^2\right|. \tag{8.22}$$

The right hand side of (8.21) is bounded by

$$\left|\sum_{j=1}^{k} 2\sigma_j X_{j,:}(X_{k+1:d})^T \cdot \|\sigma_{k+1:d}\| \sum_i t_i v_{i,j}(v_{i,k+1:d})^T\right|$$

$$\leq \sum_{j=1}^{k} 2\sigma_j \|X_{j,:}X_{k+1:d}\| \cdot \|\sigma_{k+1:d}\| \left\|\sum_i t_i v_{i,j} v_{i,k+1:d}\right\| \tag{8.23}$$

$$\leq \sum_{j=1}^{k} (\varepsilon\sigma_j^2 \|X_{j,:}\|^2 + \frac{\|\sigma_{k+1:d}\|^2}{\varepsilon} \left\|\sum_i t_i v_{i,j} v_{i,k+1:d}\right\|^2) \tag{8.24}$$

$$\leq 2\varepsilon \|AX\|^2, \tag{8.25}$$

where (8.23) is by the Cauchy-Schwartz inequality, (8.24) is by the inequality $2ab \leq a^2 + b^2$. In (8.10) we used the fact that $\sum_i t_i (v_{i,1:k})^T v_{i,k+1:d}$ is a block in the matrix $\sum_i t_i v_i v_i^T$, and

$$\begin{aligned} &\|\sigma_{k+1:d}\|^2 \leq \|AX\|^2 \quad \text{and} \quad \sum_{j=1}^{k} \sigma_j^2 \|X_{j,:}\|^2 \\ &= \|\Sigma_{1:k,1:k} X_{1:k,:}\|^2 \leq \|\Sigma X\|^2 \leq \|AX\|^2. \end{aligned} \tag{8.26}$$

Next, we bound (8.22). Let $Y \in \mathbb{R}^{d\times k}$ such that $Y^TY = I$ and $Y^TX = \mathbf{0}$. Hence, the columns of Y span the k-subspace that is orthogonal to each of the $(d-k)$ columns

of X. By using the Pythagorean Theorem and then the triangle inequality,

$$\|\sigma_{k+1:d}\|^2 \,|\sum_i t_i \,\|(\mathbf{0}, v_{i,k+1:d})X\|^2\,| \tag{8.27}$$

$$= \|\sigma_{k+1:d}\|^2 \,|\sum_i t_i \,\|(\mathbf{0}, v_{i,k+1:d})\|^2$$

$$- \sum_i t_i \,\|(\mathbf{0}, v_{i,k+1:d})Y\|^2\,|$$

$$\leq \|\sigma_{k+1:d}\|^2 \,|\sum_i t_i \,\|v_{i,k+1:d}\|^2\,| \tag{8.28}$$

$$+ \|\sigma_{k+1:d}\|^2 \,|\sum_i t_i \,\|(\mathbf{0}, v_{i,k+1:d})Y\|^2\,|. \tag{8.29}$$

For bounding (8.29), observe that Y corresponds to a $(d-k)$ subspace, and $(\mathbf{0}, v_{i,k+1:d})$ is contained in the $(d-k)$ subspace that is spanned by the last $(d-k)$ standard vectors. Using same observations as above (8.14), there is a unit vector $y \in \mathbb{R}^{d-k}$ such that for every $i \in [n]$ $\|(\mathbf{0}, v_{i,k+1:d})Y\|^2 = \|(v_{i,k+1:d})y\|^2$. Summing this over t_i yields,

$$|\sum_i t_i \,\|(\mathbf{0}, v_{i,k+1:d})Y\|^2\,| = |\sum_i t_i \,\|v_{i,k+1:d}y\|^2\,|$$

$$= |\sum_i t_i \sum_{j=k+1}^{d} v_{i,j}^2 y_{j-k}^2| = |\sum_{j=k+1}^{d} y_{j-k}^2 \sum_i t_i v_{i,j}^2|.$$

Replacing (8.29) in (8.27) by the last inequality yields

$$\|\sigma_{k+1:d}\|^2 \,|\sum_i t_i \,\|(\mathbf{0}, v_{i,k+1:d})X\|^2\,|$$

$$\leq \|\sigma_{k+1:d}\|^2 \,(|\sum_i t_i v_{i,d+1}^2| + \sum_{j=k+1}^{d} y_{j-k}^2 \left\|\sum_i t_i v_i v_i^T\right\|) \tag{8.30}$$

$$\leq \|\sigma_{k+1:d}\|^2 \,(\varepsilon + \varepsilon \sum_{j=k+1}^{d} y_{j-k}^2) \leq 2\varepsilon \,\|AX\|^2\,, \tag{8.31}$$

where (8.30) follows since $\sum_i t_i v_{i,j}^2$ is an entry in the matrix $\sum_i t_i v_i v_i^T$, in (8.31) we used (8.26) and the fact that $\|y\|^2 = 1$. Plugging (8.10) in (8.21) and (8.31) in(8.16)

gives the desired bound on (8.13) as

$$|\sum_i t_i \|A_{i,:}X\|^2 - \sum_i t_i \|(A_{i,1:k}, 0)X\|^2| \leq 4\varepsilon \|AX\|^2 .$$

Finally, using (8.19) in (8.12) and the last inequality in (8.13), proves the desired bound of (8.11).

□

Together, Theorems 8 and 9 show that the error of the coreset is a $1 \pm \varepsilon$ approximation to the true weighted mean. By Theorem 9, we can now simply apply Algorithm 10 to the right hand side of (8.5) to compute the reduction. The intuition for this inefficient reduction is as follows. We first compute the outer product of each row vector x in the input matrix $A \in \mathbb{R}^{[n \times d]}$. Each such outer products $x^T x$ is a matrix in $\mathbb{R}^{d \times d}$. Next, we expand every such matrix into a vector, in $\mathbb{R}^{d^2}$ by concatenating its entries. Finally, we combine each such vector back to be a vector in the matrix $P \in \mathbb{R}^{n \times d^2}$. At this point the reduction is complete, however it is clear that this matrix expansion is inefficient.

The second step of the reduction is to transform the slow computation of running Algorithm 10 on the expanded matrix $P \in \mathbb{R}^{n \times d^2}$ into an equivalent and provably fast computation on the original set of points $A \in \mathbb{R}^d$. To this end we make use of the fact that each row of P is a sparse vector in $\mathbb{R}^d$ to implicitly run the computation in the original row space $\mathbb{R}^d$. We present Algorithm 11 and prove that it returns the weight vector $w = (w_1, \cdots, w_n)$ of a (k, ε)-coreset for low-rank approximation of the input point set P, and that this coreset is small, namely, only $O(k^2/\varepsilon^2)$ of the weights (entries) in w are non-zeros. Algorithm 11 contains the pseudocode for LowRankCoreset.

8.4.1 Analysis of Algorithm 11

Algorithm 11 contains the full listing of the construction algorithm for the coreset for low rank approximation.

Input: A: n input points $a_1, \ldots, a_n$ in $\mathbb{R}^d$; $k \geq 1$: the approximation rank; $\varepsilon > 0$: the nominal approximation error.

Output: a non-negative vector $w \in [0, \infty)^n$ of only $O(1/\varepsilon^2)$ non-zeros entries which are the non-negative weights of the corresponding points selected for the coreset.

Analysis: Algorithm 11 starts by computing the k-SVD of input matrix A (line 5). This is possible because we use the streaming model, so that the input arrives in small blocks. For each block we perform the computation to create its coreset. By merging the resulting coresets we preserve sparsity and can aggregate the coreset for A. Lines 7–8 use the k-SVD of this small input block to restructure the input matrix A into a combination of the columns of A corresponding to its k largest eigenvalues and the remaining columns of D, the singular values of A.

After initialization, we begin the main iterative steps of the algorithm. Note that lines 12–19 of Algorithm 11 are heavily optimized but functionally equivalent to lines 9–27 of Algorithm 10 – the end result in both cases is a computation of α at each iteration of the for loop, and an update to the vector of weights w. First we find the index j of the farthest point from the initial point a_1 (Line 13). The next point is implicitly added to the coreset is by updating w, and in turn affects the next farthest point as the computation wXX_i is performed iteratively. The variables a, b, c implicitly compute the distance from the current point p to the previous center q, the error of the current iteration ε, the point on the line from the p to q that is closest to the origin, and the distance between the current center q and the new center q'. Finally, line 17 updates α and line 18 updates w using the new value of α.

The algorithm terminates after k^2/ε^2 iterations, and we omit the explicit computation of ε since it is implied in the guarantees proven in the following section. As in Algorithm 10, the terminating conditions depend on the system specifications. We may wish to bound the error, or the number of iterations, or the update value α.

8.5 Conclusions

In this chapter we presented an efficient construction for a coreset for principal component analysis (PCA) that is both small in size and a subset of the original data. In the next chapter we describe our system implementation and in Section 9.1 we present experimental results.

Algorithm 10 SUMVECSCORESET(A, ε)

Input: A: n input points $a_1, \ldots, a_n$ in $\mathbb{R}^d$
Input: $\varepsilon \in (0, 1)$: the approximation error
Output: $w \in [0, \infty)^n$: non-negative weights
$A \leftarrow A - \text{mean}(A)$ *// Translate points of A to the origin $\mathbf{0}$ of $\mathbb{R}^d$ s.t. mean of $\overline{A}$ = 0*
$A \leftarrow c\,A$ where c is a constant s.t. $\text{var}(A) = 1$ *// Scale each point by constant C s.t. variance of A = 1*
$w \leftarrow (1, 0, \ldots, 0)$
$j \leftarrow 1,\ p \leftarrow A_j,\ J \leftarrow \{j\}$ *// Initialize starting point*
$M_j = \left\{y^2 \mid y = A \cdot A_j^T\right\}$ *// Maintain lookup table of computed inner products*
for $i = 1, \ldots, n$ **do**
  $j \leftarrow \text{argmin}\,\{w_J \cdot M_J\}$ *// Compute farthest point A_j*
  $G \leftarrow W' \cdot A_J$ where $W'_{i,i} = \sqrt{w_i}$
  $\|c\| = \left\|G^T G)\right\|_F^2$
  $c \cdot p = \sum_{i=1}^{|J|} G\,p^T$
  $\|c - p\| = \sqrt{1 + \|c\|^2 - c \cdot p}$
  $\text{comp}_p(v) = 1/\|c - p\| - (c \cdot p)\,/\|c - p\|$
  $\|c - c'\| = \|c - p\| - \text{comp}_p(v)$
  $\alpha = \|c - c'\|\,/\,\|c - p\|$
  $w \leftarrow w(1 - |\alpha|)$ *// Update weights*
  $w_j \leftarrow w_j + \alpha$
  $w \leftarrow w / \sum_{i=1}^{n} w_i$
  $M_j \leftarrow \left\{y^2 \mid y = A \cdot A_j^T\right\}$ *// Update lookup table*
  $J \leftarrow J \cup \{j\}$
  if $\|c\|^2 \leq \varepsilon$ **then**
    break
  end if
end for
return w

Algorithm 11 LOWRANKCORESET(A, k, ε)

Input: A: A sparse $n \times d$ matrix
Input: $k \in \mathbb{Z}_{>0}$: the approximation rank
Input: $\varepsilon \in \left(0, \frac{1}{2}\right)$: the approximation error
Output: $w \in [0, \infty)^n$: non-negative weights
Compute $U\Sigma V^T = A$, the SVD of A
$R \leftarrow \Sigma_{k+1:d,k+1:d}$
$P \leftarrow$ matrix whose i-th row $\forall i \in [n]$ is
    $P_i = (U_{i,1:k}, U_{i,k+1:d} \cdot \frac{R}{\|R\|_F})$
$X \leftarrow$ matrix whose i-th row $\forall i \in [n]$ is
    $X_i = P_i / \|P_i\|_F$
$w \leftarrow (1, 0, \ldots, 0)$
for $i = 1, \ldots, \lceil k^2/\varepsilon^2 \rceil$ **do**
    $j \leftarrow \operatorname{argmin}_{i=1,\ldots,n}\{wXX_i\}$
    $a = \sum_{i=1}^n w_i (X_i^T X_j)^2$
    $b = \dfrac{1 - \|PX_j\|_F^2 + \sum_{i=1}^n w_i \|PX_i\|_F^2}{\|P\|_F^2}$
    $c = \|wX\|_F^2$
    $\alpha = (1 - a + b) / (1 + c - 2a)$
    $w \leftarrow (1 - \alpha) I_j + \alpha w$
end for
return w

Chapter 9

Coresets for Dimensionality Reduction – Applications to Topic Modeling

In this chapter[1] we provide an efficient implementation our coreset for dimensionality reduction presented in the previous chapter. We present a system that can compute a latent semantic analysis for the term-document matrix of Wikipedia with provable error bounds. We demonstrate our system on a standard laptop as well as on the Amazon Elastic Cloud Compute (EC2), and show a significant improvement in running time and accuracy compared to existing heuristics. We evaluate our system on synthetic data to provide a ground-truth for the quality, efficiency, and scalability of our system. Finally, we apply our algorithm to compute the principal component analysis (PCA) of the Wikipedia document-term matrix, and use this to compute a topic model of the English Wikipedia.

[1]Some of the content in this chapter was published in [54].

9.1 Evaluation and Experimental Results

The coreset construction algorithm described in Section 8.4 was implemented in MATLAB. We make use of the redsvd package [61] to improve performance, but it is not required to run the system. We evaluate our system on two types of data: synthetic data generated with carefully controlled parameters, and real data from the English Wikipedia under the "bag of words" (BOW) model. Synthetic data provides ground-truth to evaluate the quality, efficiency, and scalability of our system, while the Wikipedia data provides us with a grand challenge for latent semantic analysis computation.

For our synthetic data experiments, we used a moderate size sparse input of (5000×1000) to evaluate the relationship between the error ε and the number of iterations of the algorithm N. We then compare our coreset against uniform sampling and weighted random sampling using the squared norms of U ($A = U\Sigma V^T$) as the weights. Finally, we evaluate the efficiency of our algorithm by comparing the running time against the MATLAB `svds` function and against the most recent state of the art dimensionality reduction algorithm [26]. Figure 9-2a–9-2d show the exerimental results.

Approximation error. We carried out experiments on a moderate size sparse input of (5000×1000) to evaluate the relationship between the error ε and the number of iterations of the algorithm N. for a hyperplane coreset (i.e. $k = d - 1$). Fig. 9-2d shows how the characteristic function of the approximation error $f(N)$ behaves with respect to increasing number of iterations N (normalized to $N = n$). Note that three of the plotted functions $f(N)$ converge as N increases, while the last one ramps up and then increases linearly. From this we conclude that ε decreases at a true rate somewhere between the rates of increase of $f(N) = N \log N$ and $f(N) = N^2$. The true characteristic $f^*(N) + C$ indicates the theoretical breakpoint between increasing and decreasing error.

We then compare our coreset against uniform sampling and weighted random sampling, using the squared norms of U ($A = U\Sigma V^T$) as the weights. Tests were

carried out on a small subset of Wikipedia (n=1000, d=257K) to ensure representative data structure. Figure 9-2a–9-2c shows the results. As expected, approximation error decreases with coreset size, as well as the subspace rank. (Note that since our algorithm is deterministic, there is zero variance in the approximation error.)

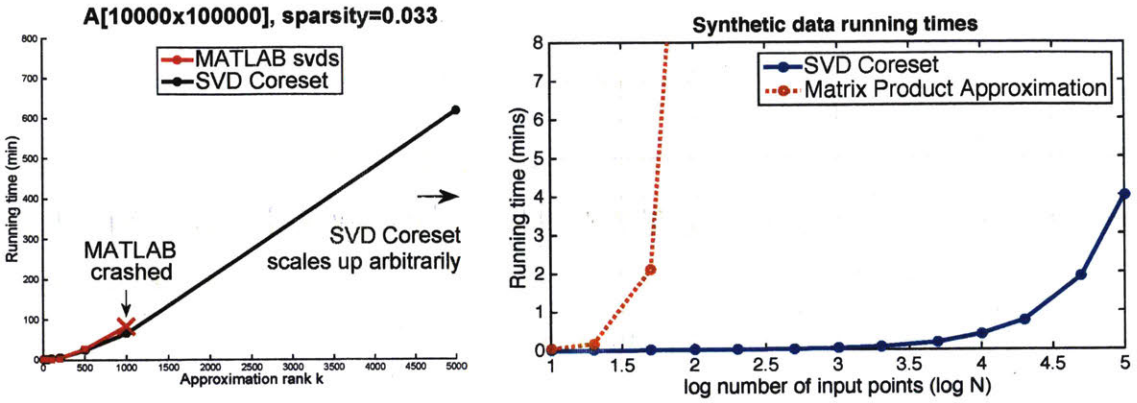
Running time. We evaluate the efficiency of our algorithm by comparing the running time (coreset construction) against the built-in MATLAB **svds** function and against the most recent state of the art dimensionality reduction algorithm [26].

We implemented the matrix spectral approximation algorithm presented in [26]. Fig. 9-1a shows the running time of our coreset compared against MATLAB svds. Fig. 9-1b shows the running time of our algorithm compared against Algorithm 12 run on synthetic data for the same set of input parameters. We used a fixed dimensionality $d = 1000$, approximation rank $k = 100$, sparsity 10^{-6} and evaluated construction time for increasing input size N. The results are plotted as a function of the log of the input size to show the order of magnitude difference in performance.

Besides the fact that our algorithm minimizes the Frobenius norm and support PCA, an important advantage of our technique compared to existing coreset constructions is that it is much numerically stable and faster in practice. For example, the result of [27] is based on the technique of [11]. This technique needs to compute many inverse of matrices during the computation, which makes it not only less stable but also very inefficient. Indeed, we implemented the coreset construction of [27] and the running time comparison to our algorithm for the same coreset size can be found in Fig. 9-1b. In conclusion, our algorithm is faster, numerically stable, and can be computed on practically unbounded size input data.

9.1.1 Latent Semantic Analysis of Wikipedia

For our large-scale grand challenge experiment, we apply our algorithm for computing principal component analysis (PCA) on the entire English Wikipedia. The size of the data is $n = 3.69$M (documents) with a dimensionality $d = 7.96$M (words). Unfortunately, there is no ground truth for the sum of squared distances to the real SVD, as that has never been computed on such a large dataset as Wikipedia. We specify a



(a) Relative error ($k = 10$) **(b)** Relative error ($k = 20$)

Fig. 9-1a shows the runtimes of our coreset compared against MATLAB svds. Fig. 9-1b shows the runtimes of our coreset compared against the algorithm in [26].

Figure 9-1: Coreset runtime experiments

nominal error of $\varepsilon = 0.5$, which is a theoretical upper bound for $N = 2k/\varepsilon$ iterations, and show that the coreset error remains bounded. In practice we can do much better than this. Figure 9-2f shows the log approximation error, i.e. sum of squared distances of the coreset to the subspace for increasing approximation rank $k = 1,\ 10,\ 100$. We see that the log error is proportional to k, and as the number of streamed points increases into the millions, coreset error remains bounded by k. Figure 9-2e shows the running time of our algorithm compared against `svds` for increasing dimensionality d and a fixed input size $n = 3.69$M (number of documents).

Finally, we show that our coreset can be used to create a topic model of 100 topics for the entire English Wikipedia. We construct the coreset of size $N = 1000$ words. Then to generate the topics, we compute a projection of the coreset onto a subspace of rank $k = 100$.

For these experiments we used three types of machines:

1. Regular desktop computer with quad-core Intel Xeon E5640 CPU @2.67GHz, 6GB RAM (low spec).

2. Modern laptop with quad-core Intel i7-4500U CPU @1.8GHz, 16GB RAM (medium spec).

3. High-performance computing clusters on Amazon Web Services (AWS) as well as local clusters, e.g. an EC2 c3.8xlarge machine with 32-core Intel Xeon E5-2680v2 vCPU @2.8Ghz, 60GB RAM (high spec).

We compute the coreset using a buffer stream of size $N/2$, parallelized across 64 nodes on Amazon Web Services (AWS) clusters. The 64 individual coresets are then unified into a single coreset. Figure 9-2e shows the running time of our algorithm compared against `svds` for increasing dimensionality d and a fixed input size $n = 3.69$M (number of documents). Note that this is a log-scale plot of dimensionality against running time, so the differences in performance represent orders of magnitude. The desktop computer with 6GB RAM crashed for $d = 2000$ and was omitted from the plot. The same algorithm running on the cluster (blue plot) outperformed the laptop (red plot), which also quickly ran out of memory. Comparing `svds` computation on AWS against our coreset (green plot) highlights the difference in performance for identical computer architectures. As the dimensionality d increases, any algorithm dependent on d will eventually crash, given a large enough input.

We show that our coreset can be used to create a topic model of $k = 100$ topics for the entire English Wikipedia, with a fixed memory requirement and coreset size of just $N = 1000$ words. We compute the projection of the coresets on a subspace of rank k to generate the topics. Table 9.2 shows a selection of 10 of the most highly weighted words from 4 of the computed topics. The total running time, including coreset construction, merging and topic extraction was 140.66 min.

A cursory glance at the words suggests that the "themes" of these topics are (1) urban planning, (2) economy and finance, (3) road safety, (4) entertainment. This serves as a qualitative proof of concept that our system can produce meaningful results topics on very large datasets. We view this result optimistically, as proof of concept that our system can be used to compute a topic model of the English language. A more objective analysis would involve using a corpus of tagged documents as a ground truth, projecting the corresponding vectors onto our topics, and comparing the classification error against topics computed by other systems. This is the subject of our ongoing work.

k	Size	N	Coreset error	Running time
1	4	4	0.269	34 min
10	37	40	0.0312	42 min
100	167	400	0.0004	140 min

Summary of Wikipedia experimental results (corpus size = 12GB).

Table 9.1: Wikipedia experimental results

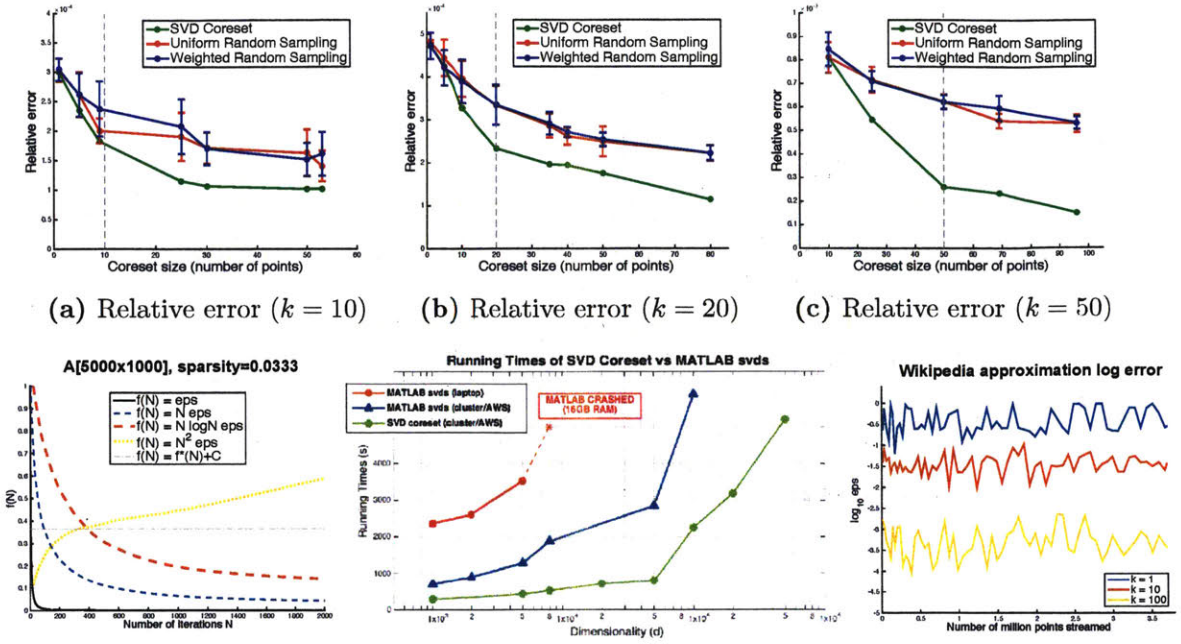
Topic 1	Topic 2	Topic 3	Topic 4
US	credit	drivers	comedy
highway	risk	distracted	nominated
bridge	plan	phone	actress
road	union	driver	awards
river	interest	text	television
traffic	rating	car	episode
downtown	earnings	brain	musical
bus	capital	accidents	writing
harbor	liquidity	visual	tv
street	asset	crash	directing
...	...	...	...

Example of the highest-weighted words from 4 topics of the $k = 100$ topic model of Wikipedia computed by our algorithm.

Table 9.2: Wikipedia topic model examples

9.1.2 Technical Summary

- First coreset for PCA that is both small and a subset of the original points, and preserves the sparsity of the data.



(a) Relative error ($k = 10$) **(b)** Relative error ($k = 20$) **(c)** Relative error ($k = 50$)

(d) Synthetic data errors **(e)** Wikipedia running time (x-axis log scale) **(f)** Wikipedia log errors

Fig. 9-2a–9-2c show the relative error of our coreset against uniform random sampling and weighted random sampling. Fig. 9-2d shows the coreset approximation error ε for increasing number of iterations N. Fig. 9-2e shows the algorithm running times for increasing d on large datasets ($n = 3.8$M). Fig. 9-2f shows the approximation error (in log scale) for the entire Wikipedia.

Figure 9-2: Experimental results for synthetic data and Wikipedia

---

- Coreset construction is independent of both n and d.
- We present SUMVECSCORESET, a coreset for the sum of vectors based on a generalization of the Frank-Wolfe algorithm.
- We present an efficient reduction to LOWRANKCORESET, a coreset for low-rank approximation (k-SVD) and PCA.
- Coreset is of size $O(k^2/\varepsilon^2)$.
- Implemented our algorithms on Amazon EC2.
- Computed the $k = 100$ low-rank approximation of the English Wikipedia matrix in just over 2 hours.

9.2 Conclusions

In Chapters 8 and 9 we presented a new approach for dimensionality reduction using coresets. Our solution is general and can be used to project spaces of dimension d to subspaces of dimension $k < d$. The key feature of our algorithm is that it computes coresets that are small in size and subsets of the original data. We benchmark our algorithm for quality, efficiency, and scalability using synthetic data. We then apply our algorithm for computing PCA on the entire Wikipedia – a computation task hitherto not possible with state of the art algorithms. We see this work as a theoretical foundation and practical toolbox for a range of dimensionality reduction problems, and we believe that our algorithms will be used to construct many other coresets in the future.

Chapter 10

Conclusions and Future Work

In this thesis, we presented a family of real-time data reduction algorithms for large streams. We discussed some of the challenges that arise when dealing with real Big Data systems, and motivated the need for new data reduction techniques, in the form of theoretical and practical open problems. We showed how we can use coresets to compute a compact and meaningful representation of the data that can enable efficient analysis such as segmentation, summarization, state estimation, and prediction.

10.1 Summary of Contributions

In Chapters 4 and 5, we proposed a coreset for the k-segment mean problem, of size $O(k/\varepsilon^2)$ that provides a $(1+\varepsilon)$-approximation for the sum of squared distances to any given k-piecewise linear function. The coreset is constructed in $O(dk)$ time, using $O(\log n)$ memory. We present the BICRITERIA algorithm estimates the complexity of the data in $O(\log n)$ iterations, and the BALANCEDPARTITION algorithm which uses the complexity estimate to construct a (k,ε)-coreset for the k-segment mean. We presented a system for approximating the k-segmentation of streaming data, and provide experimental results with video streams, GPS data, and financial price data.

In Chapter 6 we proposed algorithms for semantic summarization and retrieval of video frames from unbounded life-long video streams. Using union, compression, and merging we can compute a streaming coreset tree with $O(\log n)$ coresets. Using this

coreset tree, we presented a mechanism for computing an adaptive, semantic summary of the video. An adaptive keyframe selection algorithm propagates keyframes up the coreset tree by computing a tradeoff between variability (encoded as ℓ_2 distance between frames) and relevance (encoded as weighted per-frame scored that are based on the context of the video). We prove that the algorithm yields a result to within a constant factor approximation. Finally, we presented a system for efficient loop closure detection by novel sampling approach that is adaptive to the structure of the video, and validated our approach with experimental results on real video data of a tour of Boston.

In Chapter 7 we presented a system for automatically identifying the phases of laparoscopic and robot-assisted surgical procedures and segmenting them in real-time. By consulting with expert surgeons we compiled a corpus of ground truth segmentation of their recorded surgical video, specifically for the laparoscopic vertical sleeve gastrectomy (LSG) procedure. We then used their detailed verbal explanations to augment and fine-tune a visual feature space that captures the main axes of variability and other discriminant factors within this very narrow visual feature space. We trained a set of SVMs to classify each of the surgical phases, and used an HMM to compute the phase predictions. We demonstrated the effectiveness of our system on unseen surgical video, achieving 92%+ prediction accuracy. We also achieved a 90%+ compression using coreset summary, while maintaining our performance benchmark.

Finally, in Chapters 8 and 9, we proposed two novel dimensionality reduction algorithms for computing a (k, ε)-coreset of size independent of both n and d, for any given $n \times d$ input matrix, and is a weighted subset of the original data. The first algorithm, SumVecsCoreset, computes a coreset for the sum of vectors based on a generalization of the Frank-Wolfe algorithm. We then present a reduction to the second algorithm, LowRankCoreset, which computes a coreset of size $O(k^2/\varepsilon^2)$ for low-rank approximation (k-SVD) and PCA. We presented a system for computing an efficient low-rank approximation of a large sparse matrix, implemented on Amazon EC2, and showed how the system can be used to efficiently compute the PCA for the entire English Wikipedia.

10.2 Future Work

I believe that the following directions of future work are particularly important and fruitful to consider for anyone planning to build on any of these respective systems.

10.2.1 Video Segmentation

For further work with video segmentation, we should develop a better set of metrics and user interface tools for evaluating video segmentations. Currently, with our video segmentations, there is often no right or wrong answer, and claiming that a segmentation is good because some of the segments matched, seems like rationalizing. It would be prudent to develop a system for easy annotation of captured videos, so that we could collect ground truth against which to objectively evaluate our segmentations. Such a system needs to be designed with a degree of seriousness because poor usability can often be the difference between a willing and an annoyed volunteer. Another viable option is to consider using Amazon Mechanical Turk, to automate this process.

10.2.2 Financial Data Segmentation

Similarly for financial data segmentation, the kind of retrospective segmentation studies that we had performed hitherto are basically a form of *technical analysis*, which is disputed by the efficient market hypothesis [89], and is generally considered to be pseudoscience.

This calls into question some deeper issues concerning the predictive power of coresets. We are reminded that the (α, β)-approximation can only give us a useful complexity measure for offline data, or for representative data with a similar distribution. But even if we had a perfect complexity oracle allowing us to create perfect balanced partitions, unfortunately this would not entail any predictive power that would make an online k-segment coreset any more useful than say, a moving average, or any other *technical indicator* based on historical data.

If we are to delve further into analyzing financial markets, then future work should aim to embrace the capital asset pricing model (CAPM) and introduce a rigorous

system of model construction and backtesting. It is well possible that coresets have a predictive power, particularly the (k, m)-segment mean problem, which has an inherent capacity to express recurring patterns in temporal data, but for such a study we should implement the correct development and testing frameworks that are already well established for developing rigorous algorithmic trading models.

10.2.3 Semantic Video Summarization

I am very optimistic about future work on the coreset tree and semantic video summarization. I believe that it is still in its infancy in terms of its true potential. The keyframe selection algorithm is close to optimal, as was proved in [135], but I believe that we can research and implement and much more expressive and robust mechanism.

Currently, the retrieval interface was not showcased very often. Indeed, it is not very impressive as it lacks a web interface and/or a mobile app. But the idea of summarizing content with 9 images is still be very relevant at the time of writing this thesis [103].

10.2.4 Medical Data Analysis

The main focus of ongoing work is to improve the phase prediction accuracy. We are extending our system to consider continuous likelihood models, allowing us use temporal regularity to handle ambivalent phase predictions more effectively. By using the learned SVMs to model log-likelihoods of the individual phases, we can obtain improved results compared to the per-frame votes while enforcing a meaningful set of transitions. To this end, we are also looking into other temporal models for non-monotonic phase sequences.

Insofar as coresets will remain a part of this project, my goal going forward would be to extend the apply the semantic video summarization framework in [135] to create interactive visual summaries of laparoscopic and robot-assisted surgeries.

One outcome that the surgeons were very keen on if we could get enough video

data, is to evaluate our predictive model across different surgical procedures. I believe that this will require more than just additional videos. However, I think that this is a very useful extension, and a step in the right direction towards making a general surgical assistance system.

Just recently, the IBM Watson supercomputer was successfully able to diagnose a rare type of leukemia and identify the correct course of treatment faster than would have been possible through manual analysis [129]. With the increased reliance on computational resources, surgeons face a tough challenge in determining when they can confidently offload their judgment to a computer that can do the job faster, if not better. Going beyond a video-based phase prediction system, we believe that a more complete model of the operating room would be required for developing predictive medical systems in the future.

10.2.5 Coresets for Dimensionality Reduction

I believe that future work on this project should be dedicated towards improving the system implementation. The source code has been open-sourced upon acceptance of [54], and time should be spent making this remarkable theoretical result a more accessible practical reality. A more refined analysis of the Wikipedia topics that includes error bounds would be a very convincing result, and would help to engage information retrieval, natural language processing, and other communities.

10.3 Final Thoughts

"Don't Panic."

— Douglas Adams, *The Hitchhiker's Guide to the Galaxy* (1979)

This work has forced me reach wide and dig deep, and called into question many of my views about academia and about life. In the end, I have put my heart and soul into this thesis, and if my work goes on to save even a single life in the future then it will have been worth it.

Bibliography

[1] Google trends, "bitcoin" search term, August 2016.

[2] Dimitris Achlioptas and Frank Mcsherry. Fast computation of low-rank matrix approximations. *Journal of the ACM (JACM)*, 54(2):9, 2007.

[3] Douglas Adams. *The Hitchhiker's Guide to the Galaxy.* Pan Books, 1979.

[4] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximations via coresets. *Combinatorial and Computational Geometry - MSRI Publications*, 52:1–30, 2005.

[5] Jason Altschuler, Aditya Bhaskara, Vahab Mirrokni, Afshin Rostamizadeh, Morteza Zadimoghaddam, et al. Greedy column subset selection: New bounds and distributed algorithms. *arXiv preprint arXiv:1605.08795*, 2016.

[6] Roy Anati and Kostas Daniilidis. Constructing topological maps using markov random fields and loop closure detection. In *Advances in Neural Inf. Proc. Sys.*, pages 37–45, 2009.

[7] Sanjeev Arora, Elad Hazan, and Satyen Kale. A fast random sampling algorithm for sparsifying matrices. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 272–279. Springer, 2006.

[8] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. Sequential deep learning for human action recognition. In Albert Ali Salah and Bruno Lepri, editors, *HBU*, volume 7065 of *Lecture Notes in Computer Science*, pages 29–39. Springer, 2011.

[9] Sunil Bandla and Kristen Grauman. Active learning of an action detector from untrimmed videos. In *ICCV*, 2013.

[10] Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

[11] Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

[12] BBC. Nasa launches earth hunter probe, March 2009.

[13] BBC. Bitcoin panic selling halves its value, April 2013.

[14] Richard Bellman. On the approximation of curves by line segments using dynamic programming. *Commun. ACM*, 4(6):284, 1961.

[15] Irad Ben-Gal. On the use of data compression measures to analyze robust designs. *IEEE Transactions on Reliability*, 54(3):381–388, 2005.

[16] Tobias Blum, Hubertus Feuner, and Nassir Navab. Modeling and segmentation of surgical workflow from laparoscopic video. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6363(PART 3):400–407, 2010.

[17] Marc Bolanos, Ricard Mestre, Estefanía Talavera, Xavier Giró i Nieto, and Petia Radeva. Visual summary of egocentric photostreams by representative keyframes. *CoRR*, abs/1505.01130, 2015.

[18] Esther M Bonrath, Nicolas J Dedy, Lauren E Gordon, and Teodor P Grantcharov. Comprehensive surgical coaching enhances surgical skill in the operating room. *Annals of surgery*, 262(2):205–212, August 2015.

[19] L Bouarfa, P P Jonker, and J Dankelman. Surgical context discovery by monitoring low level activities in the OR. *MICCAI workshop on modeling and monitoring of computer assisted interventions (M2CAI)*, (October 2015), 2009.

[20] C Carathéodory. Uber den variabilitatsbereich der fourierschen konstanten von posi-tiven harmonischen funktionen, rend. circ. mat. palermo32 (1911), 193-217. *Carathéodory19332Rend. Circ. Mat. Palermo1911*, 1911.

[21] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. Inf. Theor.*, 14(3):462–467, May 1968.

[22] Winston Churchill and Paul Newman. Continually improving large scale long term visual navigation of a vehicle in dynamic urban environments. In *Proc. IEEE Intelligent Transportation Systems Conference (ITSC)*, Anchorage, USA, September 2012.

[23] Kenneth L Clarkson. Coresets, sparse greedy approximation, and the frank-wolfe algorithm. *ACM Transactions on Algorithms (TALG)*, 6(4):63, 2010.

[24] Kenneth L Clarkson and David P Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2013.

[25] CNBC. Bitcoin crash spurs race to create new exchanges, April 2013.

[26] Michael B. Cohen, Cameron Musco, and Jakub W. Pachocki. Online row sampling. *CoRR*, abs/1604.05448, 2016.

[27] Michael B Cohen, Jelani Nelson, and David P Woodruff. Optimal approximate matrix product in terms of stable rank. *arXiv preprint arXiv:1507.02268*, 2015.

[28] Alceu Ferraz Costa, Gabriel Humpire-Mamani, and Agma Juci Machado Traina. An efficient algorithm for fractal analysis of textures. In *Graphics, Patterns and Images (SIBGRAPI), 2012 25th SIBGRAPI Conference on*, pages 39–46. IEEE, 2012.

[29] Frederique Crete, Thierry Dolmiere, Patricia Ladret, and Marina Nicolas. The blur effect: perception and estimation with a new no-reference perceptual blur metric. In *Proc. SPIE*, volume 6492, 2007.

[30] Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, volume 1, pages 1–2. Prague, 2004.

[31] Mark Cummins and Paul Newman. FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance. *IJRR*, 27(6):647–665, 2008.

[32] Mark Cummins and Paul Newman. Appearance-only slam at large scale with fab-map 2.0. *IJRR*, 30(9):1100–1123, August 2011.

[33] Saeed Dabbaghchian, Masoumeh P Ghaemmaghami, and Ali Aghagolzadeh. Feature extraction using discrete cosine transform and discrimination power analysis with a face recognition technology. *Pattern Recognition*, 43(4):1431–1440, 2010.

[34] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[35] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[36] Frank Dellaert and Michael Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *IJRR*, 25(12):1181–1203, December 2006.

[37] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Computer Vision and Pattern Recognition*, 2009.

[38] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[39] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Sampling algorithms for l_2 regression and applications. In *Proc. 17th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1127–1136. ACM Press, 2006.

[40] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Subspace sampling and relative-error matrix approximation: Column-row-based methods. In *ESA*, pages 304–314, 2006.

[41] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Relative-error CUR matrix decompositions. *SIAM Journal on Matrix Analysis and Applications*, 30(2):844–881, 2008.

[42] Petros Drineas and Anastasios Zouzias. A note on element-wise matrix sparsification via a matrix-valued bernstein inequality. *Information Processing Letters*, 111(8):385–389, 2011.

[43] Brett L Ecker, Richard Maduka, Andre Ramdon, Daniel T Dempsey, Kristoffel R Dumon, and Noel N Williams. Resident education in robotic-assisted vertical sleeve gastrectomy: outcomes and cost-analysis of 411 consecutive cases. *Surgery for Obesity and Related Diseases*, 2015.

[44] E. Elhamifar, G. Sapiro, and R. Vidal. See all by looking at a few: Sparse modeling for finding representative objects. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1600–1607, June 2012.

[45] D. Feldman, A. Fiat, and M. Sharir. Coresets for weighted facilities and their applications. In *Proc. 47th IEEE Ann. Symp. on Foundations of Computer Science (FOCS)*, pages 315–324, 2006.

[46] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *STOC*, 2010. Manuscript available at arXiv.org.

[47] D. Feldman, M. Monemizadeh, C. Sohler, and D. P. Woodruff. Coresets and sketches for high dimensional subspace approximation problems. In *Proc. 21th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2010.

[48] D. Feldman, M. Schmidt, and C Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2013.

[49] D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering. *SODA*, 2013.

[50] D. Feldman, A. Sugaya, and D. Rus. An effective coreset compression algorithm for large scale sensor networks. In *IPSN*, pages 257–268, 2012.

[51] D. Feldman, C. Sung, and D. Rus. The single pixel gps: learning big data signals from tiny coresets. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 23–32. ACM, 2012.

[52] Dan Feldman and Leonard J. Schulman. Data reduction for weighted and outlier-resistant clustering. In *SODA*, pages 1343–1354, 2012.

[53] Dan Feldman, Mikhail Volkov, and Daniela Rus. Dimensionality Reduction of Massive Sparse Datasets Using Coresets. arXiv:1503.01663, February 2015.

[54] Dan Feldman, Mikhail Volkov, and Daniela Rus. Dimensionality reduction of massive sparse datasets using coresets. In *NIPS*. Curran Associates, Inc., 2016.

[55] G David Forney Jr. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[56] Mina Ghashami, Edo Liberty, Jeff M Phillips, and David P Woodruff. Frequent directions: Simple and deterministic matrix sketching. *arXiv preprint arXiv:1501.01711*, 2015.

[57] Mina Ghashami and Jeff M Phillips. Relative errors for deterministic low-rank matrix approximations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 707–717. SIAM, 2014.

[58] Anna C Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S Muthukrishnan, and Martin J Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *STOC*, pages 389–398. ACM, 2002.

[59] Yogesh Girdhar and Gregory Dudek. Efficient on-line data summarization using extremum summaries. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3490–3496. IEEE, 2012.

[60] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[61] Google. redsvd. https://code.google.com/archive/p/redsvd/, 2011.

[62] Sudipto Guha, Nick Koudas, and Kyuseok Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Transactions on Database Systems (TODS)*, 31(1):396–438, 2006.

[63] Nathan P Halko. *Randomized methods for computing low-rank approximations of matrices.* PhD thesis, University of Colorado, 2012.

[64] S. Har-Peled. Low rank matrix approximation in linear time. *Manuscript*, 2006.

[65] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *Proc. 36th Ann. ACM Symp. on Theory of Computing (STOC)*, pages 291–300, 2004.

[66] Robert M. Haralick, K. Shanmugam, and Its'Hak Dinstein. Textural Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(6):610–621, 1973.

[67] K. Ho and P. Newman. SLAM-Loop Closing with Visually Salient Features. In *ICRA*, April 2005.

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

[69] Dorit S Hochbaum. *Approximation algorithms for NP-hard problems.* PWS Publishing Co., 1996.

[70] Marcus Hutter. Towards a universal theory of artificial intelligence based on algorithmic probability and sequential decisions. In *European Conference on Machine Learning*, pages 226–238. Springer, 2001.

[71] Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability.* Springer, Berlin, 2005. 300 pages, http://www.hutter1.net/ai/uaibook.htm.

[72] Marcus Hutter. Human knowledge compression contest, July 2009.

[73] Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 332–339. ACM, 1994.

[74] H. Johannsson, M. Kaess, M.F. Fallon, and J.J. Leonard. Temporally scalable visual SLAM using a reduced pose graph. In *ICRA*, Karlsruhe, Germany, May 2013.

[75] Michel Journée, Yurii Nesterov, Peter Richtárik, and Rodolphe Sepulchre. Generalized power method for sparse principal component analysis. *The Journal of Machine Learning Research*, 11:517–553, 2010.

[76] Umashankkar Kannan, Brett L Ecker, Rashikh Choudhury, Daniel T Dempsey, Noel N Williams, and Kristoffel R Dumon. Laparoscopic hand-assisted versus robotic-assisted laparoscopic sleeve gastrectomy: experience of 103 consecutive cases. *Surgery for Obesity and Related Diseases*, 2015.

[77] Hema Swetha Koppula, Rudhir Gupta, and Ashutosh Saxena. Learning human activities and object affordances from rgb-d videos. *I. J. Robotic Res.*, 32(8):951–970, 2013.

[78] M. Labbe and F. Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE-TRA*, 29(3):734–745, June 2013.

[79] Florent Lalys, Laurent Riffaud, David Bouget, and Pierre Jannin. A framework for the recognition of high-level surgical tasks from video images for cataract surgeries. *IEEE Trans. on Biomedical Engineering*, 59(4):966–976, 2012.

[80] Florent Lalys, Laurent Riffaud, Xavier Morandi, and P Jannin. Automatic phases recognition in pituitary surgeries by microscope images classification. *Information Processing in Computer-Assisted Interventions LNCS Volume 6135*, pages 34–44, 2010.

[81] Florent Lalys, Laurent Riffaud, Xavier Morandi, and Pierre Jannin. Surgical phases detection from microscope videos by combining SVM and HMM. In *Medical Computer Vision. Recognition Techniques and Applications in Medical Imaging*, pages 54–62. Springer, 2010.

[82] Cornelius Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators.* United States Governm. Press Office Los Angeles, CA, 1950.

[83] M. Langberg and L. J. Schulman. Universal ε approximators for integrals. *SODA*, 2010.

[84] Anat Levin and Richard Szeliski. Visual odometry and map correlation. In *Computer Vision and Pattern Recognition*, volume I, pages 611–618, Washington, DC, June 2004. IEEE Computer Society.

[85] Shigang Li and S. Tsuji. Selecting distinctive scene features for landmarks. In *ICRA*, pages 53–59 vol.1, May 1992.

[86] Yunpeng Li, David J. Crandall, and Daniel P. Huttenlocher. Landmark classification in large-scale image collections. In *ICCV*, pages 1957–1964, 2009.

[87] Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–588. ACM, 2013.

[88] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007.

[89] Andrew W Lo and Jasmina Hasanhodzic. *The evolution of technical analysis: financial prediction from Babylonian tablets to Bloombery terminals*, volume 139. John Wiley & Sons, 2011.

[90] Benny PL Lo, Ara Darzi, and Guang-Zhong Yang. Episode classification for the analysis of tissue/instrument interaction with multiple visual cues. In *MICCAI*, pages 230–237. Springer, 2003.

[91] Zheng Lu and Kristen Grauman. Story-driven summarization for egocentric video. In *Computer Vision and Pattern Recognition*, pages 2714–2721, 2013.

[92] Peter Lyman and Hal Varian. How much information 2003? 2004.

[93] William P. Maddern, Michael Milford, and Gordon Wyeth. Towards persistent indoor appearance-based localization, mapping and navigation using cat-graph. In *IROS*, pages 4224–4230, 2012.

[94] Takuya Maekawa, Yutaka Yanagisawa, Yasue Kishino, Katsuhiko Ishiguro, Koji Kamei, Yasushi Sakurai, and Takeshi Okadome. Object-based activity recognition with heterogeneous sensors on wrist. In Patrik Floréen, Antonio Krüger, and Mirjana Spasojevic, editors, *Pervasive Computing*, volume 6030 of *Lecture Notes in Computer Science*, pages 246–264. Springer Berlin Heidelberg, 2010.

[95] Phil Maguire, Philippe Moser, and Rebecca Maguire. Understanding consciousness as data compression. *Journal of Cognitive Science*, 17(1):63–94, 2016.

[96] Matt Mahoney. Rationale for a large text compression benchmark, August 2006.

[97] Gabriele Marangoni, Gareth Morris-Stiff, Sunita Deshmukh, Abdul Hakeem, and Andrew M Smith. A modern approach to teaching pancreatic surgery. *Journal of gastrointestinal surgery*, 16(8):1597–1604, 2012.

[98] William McMahan, Ernest D Gomez, Liting Chen, Karlin Bark, John C Nappo, Eza I Koch, David I Lee, Kristoffel R Dumon, Noel N Williams, and Katherine J Kuchenbecker. A practical system for recording instrument interactions during live robotic surgery. *Journal of Robotic Surgery*, 7(4):351–358, 2013.

[99] David M. Mount. Design and analysis of computer algorithms. Lecture notes, University of Maryland, 2008.

[100] NASA. Kepler: Nasa's first mission capable of finding earth-size planets, February 2009.

[101] NASA. Pyke primer - 2. data resources, March 2014.

[102] NASA. Kepler and K2, spacecraft and instrument, July 2015.

[103] nine. nine, August 2016.

[104] Nicolas Padoy, Tobias Blum, Irfan Essa, Hubertus Feussner, Marie-odile Berger, Nassir Navab, and Loria-inria Lorraine. A Boosted Segmentation Method for Surgical. *MICCAI*, pages 102–109, 2007.

[105] Christopher C Paige. Computational variants of the lanczos method for the eigenproblem. *IMA Journal of Applied Mathematics*, 10(3):373–381, 1972.

[106] Christos H Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 159–168. ACM, 1998.

[107] R. Paul, D. Feldman, D. Rus, and P. Newman. Visual precis generation using coresets. In *ICRA*, pages 1304–1311, May 2014.

[108] Rohan Paul and Paul Newman. FAB-MAP 3D: Topological mapping with spatial and visual appearance. In *ICRA*, pages 2649–2656, Anchorage, Alaska, May 2010.

[109] Karl Pearson. On lines and planes of closest fit to systems of points in space. *London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. Sixth Series.

[110] Karl Raimund Popper. *The open universe: An argument for indeterminism*, volume 2. Psychology Press, 1988.

[111] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244 – 256, 1972.

[112] W.M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[113] Radim Rehurek, Petr Sojka, et al. Gensim–statistical semantics in python. 2011.

[114] Guy Rosman, Mikhail Volkov, Dan Feldman, John W Fisher III, and Daniela Rus. Coresets for k-segmentation of streaming data. In *NIPS*, pages 559–567. Curran Associates, Inc., 2014.

[115] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 143–152. IEEE, 2006.

[116] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. Improving image-based localization by active correspondence search. In *European Conf. Computer Vision*, pages 752–765, Berlin, Heidelberg, 2012. Springer-Verlag.

[117] Grant Schindler, Matthew Brown, and Richard Szeliski. City-scale location recognition. In *Computer Vision and Pattern Recognition*, 2007.

[118] Bruce Schneier. *Data and Goliath: The hidden battles to collect your data and control your world.* WW Norton & Company, 2015.

[119] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[120] Armin Shmilovici, Yoav Kahiri, Irad Ben-Gal, and Shmuel Hauser. Measuring the efficiency of the intraday forex market with a universal data compression algorithm. *Computational Economics*, 33(2):131–154, 2009.

[121] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[122] Nate Silver. *The signal and the noise: Why so many predictions fail-but some don't.* Penguin, 2012.

[123] Pritam Singh, Rajesh Aggarwal, Muaaz Tahir, Philip H Pucher, and Ara Darzi. A randomized controlled study to evaluate the role of video-based coaching in training laparoscopic skills. *Annals of surgery*, 261(5):862–869, 2015.

[124] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *ICCV*, volume 2, pages 1470–1477, October 2003.

[125] J. Sivic and A. Zisserman. Video Google: Efficient visual search of videos. In J. Ponce, M. Hebert, C. Schmid, and A. Zisserman, editors, *Toward Category-Level Object Recognition*, volume 4170 of *LNCS*, pages 127–144. Springer, 2006.

[126] Josef Sivic and Andrew Zisserman. Efficient visual search of videos cast as text retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 31(4):591–606, 2009.

[127] Stefanie Speidel, Julia Benzko, Sebastian Krappe, Gunther Sudra, Pedram Azad, Beat Peter Müller-Stich, Carsten Gutt, and Rüdiger Dillmann. Automatic classification of minimally invasive instruments based on endoscopic image sequences. In *SPIE Medical Imaging*, pages 72610A–72610A. International Society for Optics and Photonics, 2009.

[128] International Business Times. Active mobile phones outnumber humans for the first time, October 2014.

[129] The Japan Times. Ibm big data used for rapid diagnosis of rare leukemia case in japan, August 2016.

[130] Goce Trajcevski, Hu Cao, Peter Scheuermann, Ouri Wolfson, and Dennis Vaccaro. On-line data reduction and the quality of history in moving objects databases. In *MobiDE*, pages 19–26, 2006.

[131] Kasturi Varadarajan and Xin Xiao. On the sensitivity of shape fitting problems. *arXiv preprint arXiv:1209.4893*, 2012.

[132] Santosh S Vempala. *The random projection method*, volume 65. American Mathematical Soc., 2005.

[133] Paul Viola and Michael Jones. Robust real-time object detection. *International Journal of Computer Vision*, 4, 2001.

[134] Mikhail Volkov, Dan A. Hashimoto, Guy Rosman, Ozanan R. Meireles, and Daniela Rus. Machine learning and coresets for automated, real-time video segmentation of laparoscopic surgery. In *SAGES Emerging Technology Session*, Boston, Massachusetts, Mar. 16-19 2016.

[135] Mikhail Volkov, Guy Rosman, Dan Feldman, John W Fisher III, and Daniela Rus. Coresets for visual summarization with applications to loop closure. In *ICRA*, Seattle, Washington, USA, May 2015. IEEE.

[136] Heng Wang, Muhammad Muneeb Ullah, Alexander Kläser, Ivan Laptev, and Cordelia Schmid. Evaluation of local spatio-temporal features for action recognition. In *British Machine Vision Conference*, page 127, sep 2009.

[137] Wikipedia. Dikw pyramid, August 2016.

[138] Jiang Zheng, M. Barth, and S. Tsuji. Qualitative route scene description using autonomous landmark detection. In *ICCV*, pages 558–562, Dec 1990.

[139] Bolei Zhou, Àgata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *Advances in Neural Inf. Proc. Sys.*, pages 487–495, 2014.