Practical Considerations for Dimensionality Reduction

User Guidance, Costly Distances, and Document Data

by

Stephen Ingram

B.Sc. Computer Science, Georgia Institute of Technology, 2004

M.Sc. Computer Science, University of British Columbia, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Computer Science)

The University Of British Columbia

(Vancouver)

September 2013

Abstract

In this thesis, we explore ways to make practical extensions to Dimensionality Reduction, or DR algorithms with the goal of addressing challenging, real-world cases.

The first case we consider is that of how to provide guidance to those users employing DR methods in their data analysis. We specifically target users who are not experts in the mathematical concepts behind DR algorithms. We first identify two levels of guidance: global and local. Global user guidance helps non-experts select and arrange a sequence of analysis algorithms. Local user guidance helps users select appropriate algorithm parameter choices and interpret algorithm output. We then present a software system, DimStiller, that incorporates both types of guidance, validating it on several use-cases.

The second case we consider is that of using DR to analyze datasets consisting of documents. In order to modify DR algorithms to handle document datasets effectively, we first analyze the geometric structure of document datasets. Our analysis describes the ways document datasets differ from other kinds of datasets. We then leverage these geometric properties for speed and quality by incorporating ideas from text querying into DR and other algorithms for data analysis.

We then present the Overview prototype, a proof-of-concept document analysis system. Overview synthesizes both the goals of designing systems for data analysts who are DR novices, and performing DR on document data.

The third case we consider is that of costly distance functions, or when the method used to derive the true proximity between two data points is computationally expensive. Using standard approaches to DR in this important use-case can result in either unnecessarily protracted runtimes or long periods of user monitor-

ing. To address the case of costly distances, we develop an algorithm framework, Glint, which efficiently manages the number of distance function calculations for the Multidimensional Scaling class of DR algorithms. We then show that Glint implementations of Multidimensional Scaling algorithms achieve substantial speed improvements or remove the need for human monitoring.

Preface

Parts of this thesis have appeared in publications and journal submissions. Most of Chapter 3 is based on the following published conference paper:

- Stephen Ingram, Tamara Munzner, Veronika Irvine, Melanie Tory, Steven Bergner, and Torsten Möller. Dimstiller: Workflows for dimensional analysis and reduction. In *Visual Analytics Science and Technology (VAST), 2010 IEEE Symposium on*, pages 3–10, 2010.

A version of Chapter 4 has been submitted for publication to a journal as:

- Stephen Ingram, Tamara Munzner. The geometry of fast document queries: implications for visual analytics algorithms.

Sections of Chapter 5 are under preparation for a future conference submission and have also appeared in the following technical report:

- Stephen Ingram, Tamara Munzner, and Jonathan Stray. Hierarchical clustering and tagging of mostly disconnected data. *Technical Report TR-2012-01*, University of British Columbia Department of Computer Science, May 2012.

Chapter 6 is based on the following published conference paper:

- Stephen Ingram and Tamara Munzner. Glint: an MDS framework for costly distance functions. In *Proceedings of SIGRAD*, number 81, pages 29–38, 2012.

The author of this thesis is the first author and main contributor on each of these published papers and submissions. As first author, I performed the majority of writing for each of the above-referenced papers. As main contributor, I was responsible for designing the algorithms and coding the software presented in the different thesis chapters with the exception of the Overview prototype, the development of which received minor coding contributions from co-author Jonathan Stray.

Table of Contents

List of Tables

List of Figures

Acknowledgments

This dissertation has been made possible with help from many good folks. I would like to thank my PhD supervisor, Tamara Munzner, for being a delightful and patient academic mentor. Tamara was instrumental in helping me to select research topics, in imparting good writing and research habits, and in having reasonable expectations. I want to thank my other committee members, Torsten Möller and Nando de Freitas, for providing many helpful suggestions for greatly improving the original text of this thesis. Thanks to my university examiners, Giuseppe Carenini and Edie Rasmussen, for supplying many useful comments and corrections to the exam draft of the thesis. I'd like to thank my external examiner, Leland Wilkinson, for making the trip to Vancouver to attend my oral examination, as well as for being a wellspring of helpful advice and useful pointers to the literature. Thanks to Jonathan Stray for offering to collaborate with my colleagues and me on the Overview project, as well as for being an abundant source of research ideas. Thanks to the InfoVis group, namely Matthew Brehmer, Jessica Dawson, Joel Ferstay, and Michael Sedlmair, for patiently reading many paper drafts, always providing their impressions and feedback, and for being good company. Thanks to my friend and mentor, Art Warburton, for encouraging me to finish my PhD and giving me the opportunity to work in a comfortable place while doing so.

Special thanks to my patient partner in life, Kelsey, for keeping me sane in a crazy world, for being a wonderful mother to our children, Eleanor and Henry, and for staying my best friend after all these years. And, of course, I want to thank my Mother and Father, for encouraging my interests and for always accepting me, no matter what.

Chapter 1

Introduction

Data analysis is the act of making interpretive statements from the careful scrutiny of recorded quantities. A data analyst is the name of one who is engaged in data analysis regardless of the context, whether professional, educational, or recreational.

A common technique for performing data analysis is to consider the objects under scrutiny to be positioned in space, even when such objects are entities without any concrete reality. The space in which such objects are placed may be metaphorical, and not the three-dimensional physical space in which we reside. The abstract space has dimensions that are determined by properties of the data deemed important by the data analyst.

For example, when data is organized in a tabular format, its transformation into a set of points is straightforward. In a table of data, the individual data samples are assigned to rows, and the different numerical measurements per sample are assigned to corresponding columns in each row. A single row of such a table then represents a *point* of the dataset, while an entire column represents a *dimension*. Fisher's famous iris dataset [33] illustrates this concept nicely: the different irises being analyzed can be conceived of as points in a "measurement" space.

Another spatial data format is the *distance matrix*, where rows similarly represent points. In contrast to tabular data, though, columns no longer represent dimensions, but distances to other points. The distance matrix therefore purely catalogues the proximities of points relative to each other, without invoking any di-

mensional measurements. If the number of data points is large, then these matrices can be very inefficient to store and compute.

The spatial metaphor aids in the visual analysis of the data. There is sometimes a direct correspondence between the spatial arrangement of points to meaningful facts about the data. For example, the data might belong to two classes which correspond to two separate densities of points in space. Furthermore, the human visual system can quickly detect complex spatial properties of the data like linear relationships and clustering. Thus, by positioning objects in a space that we can visually scrutinize, one can rapidly observe spatial patterns that may provide underlying truths about the data.

When the number of dimensions are small, as is the case of the aforementioned iris dataset, then visual scrutiny of the data is straightforward. Even if there are more than two dimensions, but fewer than a dozen, techniques like scatterplot matrices and parallel coordinates plots allow for effective visualization of spatial patterns [7, 60]. But when the number of dimensions exceed a dozen or so, the number of scatterplots in a matrix or the complexity of the parallel coordinates plot makes visual analysis of all the linked views cumbersome and confusing.

Dimensionality Reduction, or DR, is a suite of data processing techniques designed to reduce the number of dimensions of a dataset while best preserving the spatial properties useful for understanding the data. Though DR has broad applicability to a variety of domains and problems, our focus in this thesis is on the use of DR for visual analysis tasks like data exploration. By reducing the dimensionality of the data to two, many techniques appropriate for low-dimensional data, like scatterplots, can be applied to high-dimensional cases.

1.1 Dimensionality Reduction

A major proportion of this thesis covers systems and algorithms for DR. As its name suggests, DR expresses a dataset in a smaller number of dimensions than originally found in the data. The input for such systems takes the form of a table representing dataset *coordinates* or a *distance matrix*.

Input tables of coordinates are represented as a matrix $\mathbf{X}$ with n rows and m columns. The matrix entry x_{ij} stores the ith point's jth coordinate value. Each row

represents an instance of a sampled data point with the m measurements stored in the corresponding columns. The magnitudes of the values in the different columns may be distributed at widely varying scales and need to be normalized to a meaningful scale if they are to be combined together.

Distance matrices D hold all the pairs of distances between points, where the matrix entry d_{ij} stores the distance between. Distance matrices are symmetric, with a zero diagonal representing a zero self-distance, and therefore only a lower triangle of the matrix need be stored. The distances stored in the matrix are the resulting output of a non-negative *distance function* $f(i, j)$, also known as a distance metric, or similarity or kernel function [79], and are interpreted as a measure of similarity between the two data points. While the most commonly encountered distance function is Euclidean, there exist many other possibilities that compute a more appropriate measure of similarity for a given application [42].

Output from dimensionality reduction is always in the form of a matrix of coordinates $\mathbf{Y}$ with a user-controlled number of dimensions. For some methods, like principal component analysis [64], metadata is available about how the coordinates are constructed from the high-dimensional input. In all but a few special cases, $\mathbf{Y}$ is invariant to both rotation and reflection. This important fact must be kept in mind to avoid ascribing meaning to the global locations of clusters and manifolds in $\mathbf{Y}$; only relative positioning is meaningful.

1.2 Methods and Uses for Dimensionality Reduction

At a high level, the methods of reducing dimensions can be grouped into three simple classes: culling, collecting, and synthesizing. Culling and collecting techniques assume the input is a table of coordinates, while synthesizing algorithms generalize to both coordinate and distance data. Here we describe each of these classes and some of their important practical uses.

Culling dimensions, also called feature selection [46], means removing a subset of dimensions outright from the dataset. Perhaps the most important reason for removing a dimension is because it does not contribute any useful information to the structure of the dataset other than noise. A trivial example is a hypothetical dataset of two dimensions, one with a bimodal distribution of values, and one with

a uniform distribution of values. The modes of distribution of the first dimension's values permit classification into one of two groups, while the distribution of the values of the second dimension only work to obscure this grouping. By culling the second dimension, the useful structure of the first dimension is clearer, both for visualization and for automated classification. A variety of techniques exist for manually detecting and culling unimportant dimensions from a dataset [62] as well as a growing literature for sophisticated automatic-selection methods [77].

Like culling, *collecting* dimensions is another form of dimension filtering. Instead of removing dimensions, though, we combine them together linearly, as a weighted average. Collecting is appropriate when two dimensions are highly correlated, or expressing largely identical relationships to the other dimensions in the data. For example, in a database of different car models, the overall weight of the car is highly correlated with the fuel economy of the car. One simple reason for collecting together highly correlated dimensions is to reduce the number of redundant visual comparisons to other dimensions in scatterplots. Collecting dimensions can be performed manually, though it is customary to use automatic techniques like Principal Component Analysis that group together correlated dimensions [64].

The third and final group of dimensionality reduction methods *synthesizes* new dimensions from input dimensions. Synthesizing methods create new datasets whose dimensions are complex, often nonlinear combinations of the original input dimensions. Synthesizing techniques feature prominently in two broadly defined use cases of visual high-dimensional analysis [98].

The first use case of synthesizing dimensions from old is that of uncovering "hidden dimensions," also called latent or hidden variables. In this case, one hypothesizes that there are a small set of dimensions that account for the observed behavior of a larger set of dimensions. A classic example of such a use case is in inferring orientation axes, such as left/right and up/down, from a database of images of an object. Here the measured dimensions are light intensities at pixel locations in an $P \times P$ image, and the synthesized "hidden dimensions" are the axes of orientation [21]. As shown in Figure 1.1, image orientation axes are confined to run along smooth, nonlinear manifolds contained within the larger space of all possible $P \times P$ images. Manifolds such as these cannot be captured by simple culling or collecting of input dimensions. The meaning and orientation of the hid-

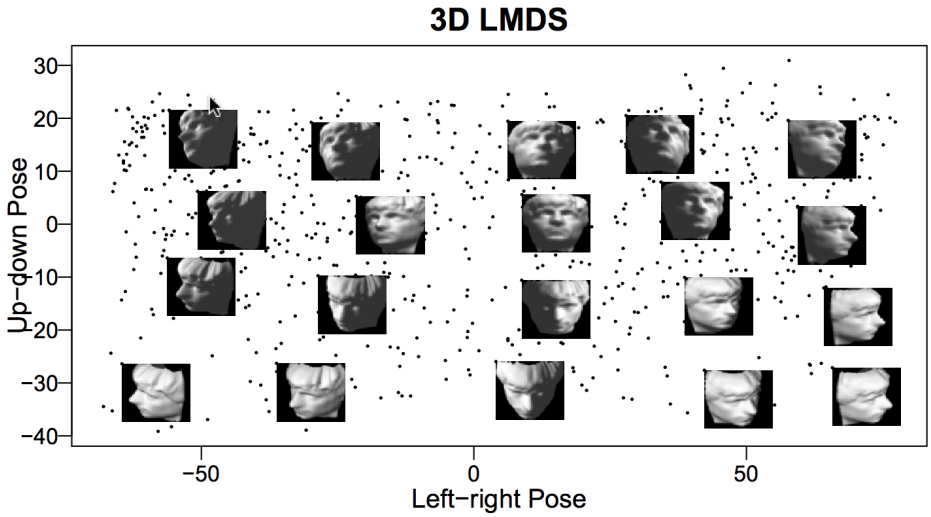


Figure 1.1: Uncovering "hidden dimensions" in a database of images. Here, the underlying data points have dimensions describing pixel intensities. By using a dimensionality reduction technique [21], orientation dimensions can be constructed from the proximity relationships in the data. ©2009 by Taylor & Francis. Reprinted by permission.

den dimensions is not given by these dimensionality reduction algorithms. Only by carefully analyzing the changes of the data across the synthesized space can the orientation and meaning of these dimensions be described [76].

The second use case of dimensionality reduction by synthesizing dimensions is that of visualizing high-dimensional clusters. A data analyst may not be interested in the way the input dimensions express the data and instead merely want a way to visually verify if there is cluster separation of their data. By using dimensionality reduction algorithms that preserve distance relationships an analyst can adequately visualize separate clusters whose numerous separating boundaries may span across a very large set of input dimensions, for example by using multidimensional scaling on a document database [57]. The dimensions spanned by these separating boundaries will be ignored by cull and collect algorithms because they will contain useful variation and often be uncorrelated. Worse, when the dimensions number in the hundreds or thousands, the separating boundaries become impossible to visualize across a set of scatterplot matrices. Figure 1.2 shows an

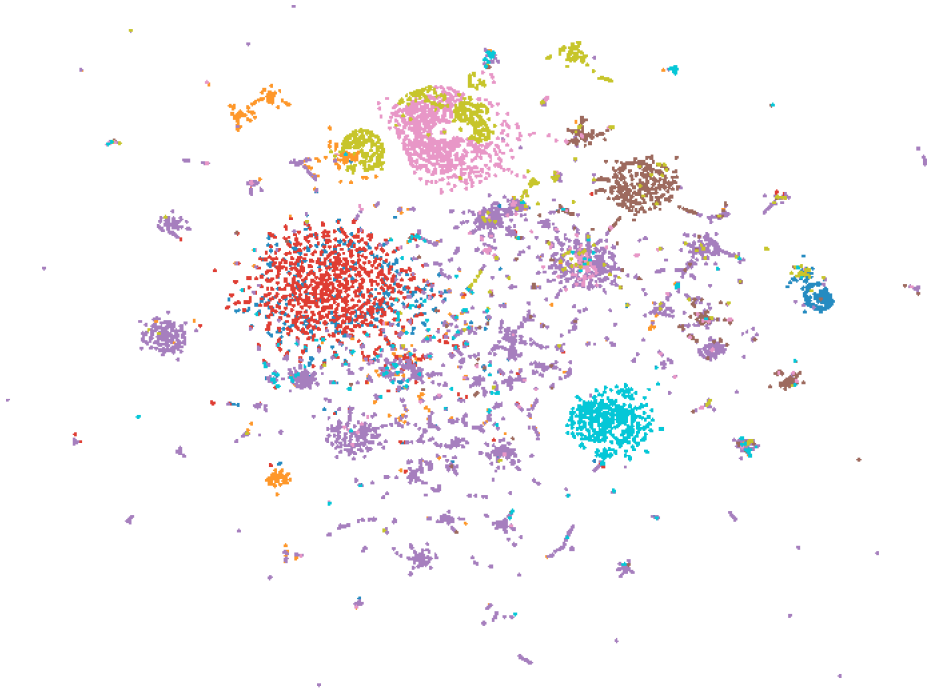


Figure 1.2: Visualizing high-dimensional clusters in a text database described in Chapter 4. The underlying data points are term-vectors whose dimensions describe the presence or absence of a term in a text. By reducing to two dimensions, clusters of similar documents become readily apparent. Cluster colors correspond to cluster assignment by non-negative matrix factorization [101].

example of dimensionality reduction producing a visualization of distinct clusters of a high-dimensional dataset.

1.3 Thesis Contributions

This doctoral thesis is greatly informed by research done in a previous Master's thesis [56]. In that work, I presented a parallel, dimensionality-reduction algorithm, *Glimmer*, designed to optimize both speed and generality across data sets. In spite of its intended generality, Glimmer does not address many important dimensionality reduction problems that do not fit neatly into its assumptions. The

components of this thesis were inspired by deep complications that emerged when Glimmer was applied to real-world cases of high-dimensional data analysis. These complications are both at a system and algorithm level and our proposed solutions make up the content of our thesis contributions.

In the remainder of this section, we present our thesis contributions as a set of research question-answer pairs. In each answer we describe the complication and how our thesis contribution helps answer the research question and give pointers in the text for more details.

How do we design systems for high-dimensional analysis aimed at DR novices? Supplying users with a powerful set of dimensionality reduction algorithms to answer their research questions only creates new questions like, "which algorithm for what task?" and "which parameter settings?" and "what am I looking at?". We contribute the design and implementation of a general analysis system called DimStiller, specifically targeted at non-expert, middle-ground users. DimStiller is built around the notion of *guidance*, where sensible parameter selections are built into the techniques themselves and commonly used compositions of techniques are stored as user-defined workflows. Chapter 3 presents the DimStiller system.

How do we design efficient DR and clustering algorithms for the important special case of large document datasets, which have very high dimensionality? Due to their complex structure, translating text into spatial points results in abstract spaces of very high-dimensionality. Existing dimensionality reduction techniques encounter many algorithm-level and qualitative problems when processing such data. We contribute a set of design implications for such data sets that takes careful advantage of the spatial structure of such spaces. The design implications create a bridge between the field of Information Retrieval and high-dimensional analysis. We furthermore contribute a set of algorithms implementing these implications and illustrate and measure their improvement over competing approaches. Chapter 4 discusses the algorithm design implications induced by the geometry of large-document datasets.

We also contribute a special-case, high-dimensional analysis system called the Overview prototype, as part of a collaboration with computational journalists. Overview targets journalists interested in rapidly exploring large, unlabelled docu-

ment dumps. These journalists are often unfamiliar with concepts such as clustering and DR. The prototype presents a set of linked visualizations of text collections that permit users to annotate different groupings of points and make sense of the larger text collection. Chapter 5 describes the Overview prototype system.

How do we practically handle costly distance functions? The design of dimensionality reductions algorithms often ignores the computational cost computing distances, focusing instead on solely reducing the cost of computing new dimensions from those distances. But, in some important cases, the functions that produce the distances between points are themselves costly to compute. This special case is handled inefficiently by current dimensionality reduction algorithms. We contribute an algorithm framework, Glint, for performing dimensionality reduction efficiently in these cases. Glint is designed to be algorithm agnostic, and focuses on minimizing the number of distance calculations to achieve a stable layout. Chapter 6 presents the Glint framework.

Chapter 2

Related Work

In this section, we describe previous work related to the contributions of this thesis. Dimensionality reduction algorithms appear in every subsequent chapter, so we begin by providing a survey of dimensionality reduction algorithm research, with particular focus on multidimensional scaling algorithms. Because of our focus on analyzing text data in Chapters 4 and 5, we then describe relevant algorithms and techniques from information retrieval, nearest-neighbor search, and hierarchical clustering, with priority given to algorithms designed for processing text data. Finally, as Chapter 3 describes a software system for facilitating the analysis of multidimensional data, we survey relevant software systems targeting this use case.

2.1 Dimensionality Reduction Algorithms

We group our survey of dimensionality reduction algorithms into four different families based on their qualitative objectives: orthogonal projections, global distances, manifold distances, and probability distributions.

2.1.1 Orthogonal Projections

The family of orthogonal projections includes methods such as principal component analysis (PCA) [64] which computes the linear projection of coordinate data onto an orthonormal basis that best preserves the variance of the data. The transformation has numerous useful properties, both by being the linear projection of the

data into the low-dimensional space with least-squared error, and also by providing a method for back-projection to the original embedding space of the data.

PCA provides useful metadata for analyzing the orthogonal projection. First, the algorithm outputs a weighted ranking of the orthogonal basis vectors, permitting determination of the most important directions of data dispersion. Second, the algorithm also supplies vectors, also called *loadings*, that contain weights describing the contribution of each input dimension given to each output dimension.

Variants of PCA algorithms exist to address speed and robustness concerns. For example, PCA can also be computed very efficiently using approximation algorithms of the singular value decomposition of a matrix [47]. Robust PCA [54] is designed to handle outliers in the data that would distort the proper alignment of the orthogonal axes.

PCA is implemented as a software component in Chapter 3 and is applied to processing text for visualization in Chapter 4.

2.1.2 Global Distances

Global distance methods, such as Multidimensional Scaling (MDS) [14], compute a low-dimensional layout of points with inter-point distances that best match the input high-dimensional distance matrix.

MDS refers to an entire family of algorithms with different objective functions, computational complexities, and qualitative results [14, 36]. The common thread is that they all minimize objectives that are some function of the difference between the Euclidean distances of the lower-dimensional layout coordinates and the magnitude of the original high-dimensional dissimilarities. The most recurrent function in the literature is the Stress function, a sum of squared residuals between high and low dimensional distances and can be written compactly as follows

$$Stress(D,\Delta)^2 = \frac{||D-\Delta||_F^2}{||D||_F^2}$$

where D is the input distance matrix and Δ is the distance matrix computed from the low dimensional coordinates, and $||X||_F^2$ represents the square of the Frobenius norm of the matrix X, or $\sum_{ij} x_{ij}^2$. Clearly, Stress goes to 0 when the distances of the

low dimensional coordinates match the input. Stress may be a multimodal function with many stationary points, making global optimization problematic [44].

Here, we discuss four major classes of MDS algorithms in terms of their shortcomings in handling costly distances. We do not describe the family of non-metric MDS methods based on distance rankings [69], as these techniques do not factor into our research.

MDS is used as a software component in both Chapters 3 and 5. In Chapter 4, we describe some shortcomings of using MDS as a dimensionality reduction technique for visualizing text datasets. The distinction between the different classes of MDS algorithms described below are important for understanding the Glint MDS algorithm framework in Chapter 6.

Analytic Algorithms

The original MDS algorithm, now called Classic MDS [113] or Principal Coordinates Analysis [41], computes a one-step global minimum of an objective function called Strain, which is expressed as

$$Strain(X) = ||XX^T - B||^2$$

where X is the $n \times l$ matrix of low dimensional coordinates and B is the so-called *double-centred* distance matrix. Double-centring subtracts the row mean from each matrix row, the column mean from each matrix column, and the mean of all the entries in the matrix from each matrix entry. Strain minimizes the discrepancy between low and high-dimensional inner products, not distances as in the Stress function. A major benefit of using Strain over Stress is that it is a convex function whose minimum can be computed without iterative techniques. The algorithm relies on computing the full SVD of a dense $N \times N$ matrix. The SVD of a dense, square matrix requires $O(N^3)$ steps [40] and is therefore too computationally complex to be suitable for large datasets or problems with costly distance functions.

Several scalable Classic MDS approximation algorithms based on the Nyström approximation of the SVD have been presented [85]. For example, both Pivot MDS [15] and Landmark MDS [30] work by having the user select a number of "pivot" or "landmark" points. These particular columns in the distance matrix are

then computed and processed by the algorithm to map the remaining points into low-dimensional space.

The main drawback to this strategy is the manual nature of selecting the proper number of landmark points. The Pivot MDS authors suggest a human-in-the-loop strategy where the user iteratively adds landmarks until visually determining the stability of the layout. The Landmark MDS authors propose an iterative strategy based on cross-validation, but do not present any benchmarks for this termination criterion. Chapter 6 shows a method to automate the selection of the number of landmarks and removes the human-in-the-loop from these techniques.

Force-Directed Algorithms

The Glimmer [56, 57] algorithm and its antecedent, by Chalmers [19], are MDS approximation algorithms that minimize Stress that iteratively sample high-dimensional distances and proportionally nudge the layout points in the direction of the residual distances. The movement of the points is controlled using a dampened force-directed simulation heuristic. The benefits of the force-directed approach include a simple implementation and a rapid convergence to a minimum region of the Stress function in fewer iterations. Force-directed algorithms can also be very scalable; the Glimmer algorithm achieves considerable speed improvement on large datasets by GPU parallelization.

Force-directed algorithms also have drawbacks. Their randomness may induce a visible level of noise in the final layout. Additionally, force-directed methods can converge to a local minimum of the Stress function that may be vastly inferior to the global minimum, though hierarchical force-directed techniques like Glimmer reduce this occurrence.

Force directed algorithms also may compute more distances than are strictly necessary. The algorithms are designed to compute high-dimensional distances prior to each force simulation time step, regardless of whether enough distance information has already been sampled to achieve a quality layout. This oversampling becomes especially inefficient when distances are costly.

Gradient Algorithms

Other MDS techniques use exact gradient information to calculate layout coordinates. Some of these algorithms use backtracking gradient descent on the Stress function [17], while the SMACOF algorithm [29] minimizes a sequence of quadratic functions that majorize the Stress function. These techniques are costly but the most flexible, permitting weights and missing values, while also converging to a lower-error minimum than randomized techniques.

As shown in their application to graph drawing [37, 66], gradient techniques can harness a sparsely populated distance matrix as input with good results in less time than using the full distance matrix. However, like analytic approximation techniques such as Pivot MDS, the precise number of distances to compute in advance to converge to a quality minimum is left up to the practitioner to deduce.

Coordinate-Only Algorithms

When MDS input takes the form of a table of coordinates, the number of input dimensions m is often much smaller than the number of points N. The PLMP [83] and LAMP [63] algorithms build on this assumption to rapidly compute low-Stress layouts for very large datasets by computing mappings for each point derived from a subset of "control" points. The profound acceleration that the algorithms achieve relative to other approaches is hindered when the number of dimensions equals or exceeds the number of points N, forcing the complexity of computing the individual mapping to approach $O(N^2)$. Thus, methods that rely on the relation $m \ll N$ for speed are less suitable than other approaches when the relation does not hold.

2.1.3 Manifold Methods

Manifold distance dimensionality reduction methods preserve distances across local surfaces formed by the data in high dimensional space, building a model of manifold connectivity. Recent work on manifold methods began with the Isomap algorithm [110], Local Linear Embedding [92], and Laplacian Eigenmaps [8] which have been followed by many, many other variants [21, 70, 104, 125].

Manifold methods are computationally intensive and often subject to numerous tuning parameters controlling how the local manifold is inferred from the data.

Furthermore, several methods make assumptions about the sampling density and number of manifolds generating the data points under analysis. In the ideal case, most of the methods are targeted at data with smooth, uniformly sampled, nonlinear structures [98]. Such structures can arise from sampling the state space of dynamical systems as in human motion motion capture [122]. The target datasets that appear in our research are not generated by uniformly sampled, nonlinear processes and therefore unlikely to lie upon a smooth, nonlinear manifold. As a result, algorithms from the class of manifold methods do not make an appearance in our thesis.

2.1.4 Probability-based Methods

Probability-based dimensionality reduction methods such as SNE [50], t-SNE [117], NE [134] and BH-SNE [116] try to minimize the discrepancy between high dimensional and low dimensional probabilities derived from distances. This discrepancy is measured as the Kullback-Leibler divergence

$$\sum_{i}^{N}\sum_{j}^{N} p_{j|i}\frac{p_{j|i}}{q_{j|i}}$$

between the two distributions P and Q representing the distributions for the high-dimensional and low-dimensional cases respectively. Probability methods work by building conditional probability distributions for each point over the other data points based on high-dimensional distances, where probability is interpreted as the likelihood that point i will be chosen as a given point j's nearest neighbor. This approach intuitively assigns points with close distances a high probability value, and points with far distances a low probability value.

The t-SNE probability-based method produces some of the most visually salient cluster visualizations of high-dimensional data. It accomplishes this salience by mapping the high-dimensional Gaussian probability distribution to a heavier-tailed Student-t probability distribution in the low-dimensional space. The mismatch in tails is specifically designed to address the *crowding problem* [117], where mappings of points within a sphere of radius r high-dimensional space quickly exhaust the exponentially smaller volume contained within a corresponding sphere with

identical radius in low-dimensional space. Using the N-body calculation speedup of SNE originally presented by de Freitas et al. [28], both NE and BH-SNE reduce the $O(N^2)$ iteration complexity of t-SNE to $O(N \log N)$ without a quality penalty in certain cases [116, 134].

In Chapter 4, we present a method for improving the speed and efficiency of the NE and BH-SNE algorithms.

2.2 Information Retrieval and the Spatial Analysis of Text

Chapter 4 discusses Information Retrieval and Spatial Analysis methods for processing text data with an emphasis on its high-dimensional structure. Below, we briefly summarize research related to this discussion, including summaries of Information Retrieval, Nearest-Neighbor Search, and Hierarchical Clustering.

2.3 Information Retrieval

The field of information retrieval studies efficient and accurate methods for querying databases, including text databases [75]. Of the several useful models of text data, our work focuses on the vector space model, where documents are mapped to vectors in term space [96]. In particular, the term vectors we study have dimension values assigned by techniques like TF-IDF [95], which assigns each term dimension a value proportional to the frequency of the term in the document and then discounts the value by its frequency of appearance in the database.

Because the number of terms in even a modestly sized database of a few thousand documents often numbers in the tens of thousands, the dimensionality of term vectors is often equal to or greater than the number of documents themselves. In spaces of such extreme dimensionality, effects from the so-called curse of dimensionality [9] become readily apparent. Such effects are known to wreak havoc with spatial search algorithms [55].

Instead, very efficient query algorithms for TF-IDF term vectors eschew spatial metaphors and instead use a data structure called the inverted file [137], thus named because it represents a sparse transpose of the term-vector matrix. In Chapter 4, we present more details about TF-IDF vectors and discuss the spatial geom-

etry of inverted file algorithms.

2.4 Nearest-Neighbor Search

Nearest-Neighbor search algorithms find the nearest set of k distinct points to a query point, where k is a parameter to the algorithm. We first discuss algorithms for general nearest-neighbor search on any dataset, and then for document-based nearest-neighbor search using inverted file indices.

2.4.1 General Nearest-Neighbor Search

General nearest-neighbor search algorithms fall into two classes: spatial-partitioning strategies, and mapping-based techniques.

Spatial partitioning strategies are those techniques that construct hierarchical structures which partition data space. Examples include balance-box decomposition trees [4] and vantage point trees [135]. After constructing hierarchical spatial decompositions of the data space with such methods, a spatial search for nearby points becomes similar to a generic search tree traversal.

Mapping-based techniques, by contrast, build functions that map data points close together with high probability. Using this technique, the search is restricted to the subspace, or set of bins, to which similar points are mapped. The seminal example of mapping techniques is locality sensitive hashing (LSH) [39], which uses random projections to divide the input space into a set of bins.

2.4.2 Inverted-File-Based Nearest-Neighbor Search

The exact nearest-neighbor problem has been tackled in the Information Retrieval and Database literature by using inverted-files as a search data structure. The all-pairs k-nearest-neighbor problem is referred to as a top-k similarity join in the Database literature [12]. Initial work in Information Retrieval reduced the search space of nearest points by partially ordering the inverted file, and then iteratively computing upper bounds on the remaining documents to be processed [80, 105]. Recent work focused on scaling up similarity calculations to massive datasets also uses a bound calculation to compute the exact nearest-neighbors, while also building the inverted file on the fly [6, 129]. In contrast to these exact techniques with

full traversal of possibly disk-resident, partial inverted files, our work in Chapter 4 performs a partial, impact-ordered traversal of a memory-resident inverted file with user-based controls over accuracy.

2.5 Hierarchical Clustering

Cluster Analysis is a rich field with numerous active subfields [32]. One such subfield that intersects this thesis is hierarchical clustering. A hierarchical clustering algorithm creates a binary tree where leaves represent the input data points (and all points are connected to the tree). Agglomerative cluster trees are built from the bottom up: two nodes are joined when they are the most similar. When the similarity measure is the nearest distance between any pair of child nodes, the algorithm is called single-link clustering [106]. Single-link clustering has several nice properties, like computational efficiency [103], and existing algorithms can be augmented with measures to reduce quality problems like cluster chaining [20, 128]. Special single-link hierarchical clustering algorithms exist for document datasets [81, 119]. As in nearest neighbor search, these methods utilize inverted file indices for fast processing of high-dimensional data. Chapter 4 details how an improved traversal of the inverted file can greatly improve the speed of computing a hierarchical clustering without a penalty to cluster quality.

2.6 Software Systems

This thesis presents two software systems, DimStiller in Chapter 3 and the Overview prototype in Chapter 5, designed for different applications: high-dimensional analysis and spatial analysis of text corpora. In this section, we survey other software systems targeted at addressing these two tasks.

2.6.1 Systems for High-Dimensional Analysis

DimStiller is designed to augment high-dimensional analysis tasks with visual guidance of algorithm parameter choices and the construction of analysis pipelines. Many other software systems are designed to be tools for high-dimensional analysis in some shape or another. In this section, we describe those relevant software

systems that provide users with access to the previously described dimensionality reduction and clustering algorithms and produce visualizations of the results.

Programming Environments

We first consider full-fledged programming environments such as MATLAB [112] or R [87]. One strength of these systems is execution speed and breadth of available graphics and analysis packages [126], but at the cost of requiring the user to access functionality through a special programming language. Beyond help files that describe the built-in functions, these systems provide no built-in mechanisms for guidance to non-expert users.

Toolkit Solutions

In specialized toolkits, several algorithms have been packaged together, usually with a GUI front end. For example, the Matlab Toolbox for Dimensionality Reduction[1] gathers together over 30 techniques for dimensionality reduction under one umbrella. Another example is the HIVE dataflow toolkit for dimensionality reduction [90]. While such tools reduce programmer time by providing easy access to a wide variety of analysis techniques, neither local nor global guidance is provided to the user.

Visual Dimensionality Analysis Environments

XmdvTool [123] supports interactive visual exploration of multivariate data sets through many types of views including scatterplot matrices, with interactive controls that include sophisticated linking and brushing techniques. It includes several approaches to collecting and culling dimensions that are based on hierarchical clustering of the dimensions using a variety of metrics, such as DOSFA [130] and VHDR [131]. Its Value and Relation (VaR) technique [132] does use MDS to create a scatterplot of dimensions, encoding information about the dimensions in an information-dense pixel-oriented glyph at each scatterplot point. However, XmdvTool is not primarily designed to support workflows built around reduction through synthesizing new dimensions. In contrast, the GGobi system [24] is a visu-

[1] homepage.tudelft.nl/19j49/Matlab_Toolbox_for_Dimensionality_Reduction.html last visited on 2/01/2010

alization framework for high-dimensional analysis with dimensionality reduction techniques that create synthetic dimensions as a central focus, supporting interaction between multiple kinds of linked views including scatterplot matrices. It also features sophisticated high-dimensional navigation including projection pursuit and grand tours, and a plugin architecture for easy connection with R [87]. The limitation of both of these frameworks is that while they implicitly provide ways to access and explore many relevant paths through table space in useful and novel ways, they lack an explicit framework for local and global guidance. The architecture of these systems is sufficiently orthogonal to the notions of guidance described in this paper that supplying them with such a framework to would require substantial ground-up development.

The rank-by-feature framework of Seo and Shneiderman [100] allows the user to visually inspect and explore dimensional relationships, but only with subsets of the original dimensions, so the huge part of table space that can only be reached via constructing synthetic dimensions cannot be explored. The data exploration environment of Guo [45] has a component-based architecture for finding clusters of the data with unique dimensional relationships.

Johansson and Johansson's [62] system embodies the concept of guiding the user through analysis stages. Users can craft quality metrics from combinations of correlation, clustering, or other measures and cull dimensions according to these measures. However, their system only supports one hardwired global workflow, where the only flexibility is in setting parameter values at each local stage. Another limitation is the inability to construct synthetic dimensions.

2.6.2 Systems for Spatial Analysis of Text Corpora

The Overview prototype, presented in Chapter 5, aids in the analysis of a text corpus with a pair of linked views presenting a clustering and dimensionality reduction of the underlying text corpus. The most relevant previous work is perhaps the Newdle system by Yang et al. [133]. It is also oriented around hierarchical clustering of topics and tag-based analysis. Yang et al. do not incorporate linked dimensionality reduction, and they show only a particular cut through the hierarchy at once, whereas our interface is based on a full hierarchy that allows the user

to explore the entire multiscale structure. Chen et al. [23] also take a sampling approach to dimensionality reduction for large document collections to produce layouts that show clear clusters, but do not discuss any sort of interactive browsing or annotation. Österling et al. compute the contour tree of a density field, which has some conceptual similarities to the hierarchical clustering [82]. The Overview prototype uses a point-based visual encoding rather than their landscape-based approach based on guidelines from previous empirical studies [114, 115].

Chapter 3

DimStiller: Workflows for Dimensional Analysis and Reduction

Many practical questions about a high dimensional dataset require understanding how the dimensions and points relate to each other and to an underlying space: Are my dimensions meaningful? How do my dimensions relate to each other? Are my clusters real?

A combination of known statistical and visualization techniques can help answer the three questions above. For example, the question "how do my dimensions relate to each other?" may be answered using Principal Components Analysis and interpreting the magnitudes of the eigenvalues and eigenvectors of the correlation matrix. Each technique may produce different output with a corresponding specialized interpretation. The sheer proliferation of techniques makes navigating this analysis space a daunting prospect for many users, who do not fully understand how and when to use these techniques correctly. For instance, what parameter settings make sense for a particular dataset? When can the output of one technique be legitimately used as the input for another?

In contrast to the profusion of previous work proposing better or faster techniques for specific aspects of dimensional analysis [49, 57], far less attention has been paid to creating systems that guide their users through the larger process of an-

alyzing high-dimensional data iteratively using a combination of techniques. For instance, many engaged in data analysis, who lack a deep knowledge of dimensional reduction, simply project their data to some 2D space and plot these points using a scatterplot. However, if the intrinsic dimensionality of the dataset is larger than two, clusters or orthogonal axes may be projected on top of each other, occluding relevant structures of interest. Although experts confronted with the problematic result of a single undifferentiated blob could conjecture that there is a mismatch between intrinsic dimensionality of the dataset and the space they chose to project to, less sophisticated users are routinely misled. The reasons for their perplexity include the sheer number of proposed dimensionality reduction techniques [51], the complexity of the mathematics underlying them, the widespread availability of dimensionality reduction tools that create only a single 2D or 3D scatterplot [65], and the lack of clear characterizations in the literature of which techniques are suitable for what kinds of data and tasks. DimStiller was expressly designed to help users avoid this pitfall, with a dimension reduction workflow that includes an estimation of intrinsic dimensionality of the dataset to guide users in making an informed choice about how many dimensions to reduce to, if at all, and having the default view of a table of more than two dimensions be a scatterplot matrix rather than a single scatterplot.

In this chapter we present the design and implementation of the DimStiller visualization framework. DimStiller gathers together a variety of techniques from dimensional analysis and reduction into a coherent framework that emphasizes the underlying dimensions and the relationships between them. For instance, it guides users through estimating the intrinsic dimensionality of a dataset before carrying out reduction. The analysis technique components of DimStiller are outfitted with interactive controls and linked views, allowing users to see and manipulate intermediate results at each analysis step. Section 3.2 describes the task of dimensionality reduction and analysis. We present the DimStiller architecture in Section 3.3, and two case studies showing how it can be used to analyze complex real-world datasets in Section 3.4.

The second contribution of this chapter to the thesis is the notion and implementation of both local and global guidance for navigating through the space of possible data tables during dimensional analysis and reduction, using the abstrac-

tions of operations, expressions, and workflows. *Expressions* instantiate a chaining together, or composition, of transformation *operators* on input data tables. Expressions show which operations have been applied to the data as well as the order in which they occur. *Workflows* are templates for exploration that consist of a specific expression along with the saved parameter values for each operator. Workflows bundle together the sequence of operators of an expression independent of the data on which they operate, permitting DimStiller users to re-use and share common patterns of analysis. While mechanisms such as expressions, operators, pipelines, and macros have been proposed previously in many contexts [48], the novelty in DimStiller is the way in which these are used to walk through a series of operations on data tables, providing guidance during the analysis process. In Section 3.1 we discuss the idea of guidance in further detail, and contrast our approach to previous work in Section 2.

3.1 Local and Global Guidance

Anyone engaged in the dimensional analysis and reduction process must choose from a vast number of possible transformations of the data table at every step – but only a relatively small subset of these transformations will yield meaningful information about the structure of the input dataset. We define providing *guidance* as structuring the exploration process to help users find this small, meaningful set from the huge space of possible transformations. We first describe this abstract analysis space, and then explain the two kinds of guidance, local and global, that DimStiller provides to support effective navigation in this space.

We model the dimensional analysis and reduction process as traversing *table space*, the space of all possible data tables. Conceptually, this space is like a graph where nodes are data tables, connected by edges representing a transformation. A path through the space begins at some node representing the input data table. Intermediate nodes are the tables that result from the transformations applied by the user, and the path terminates at the output data table. We thus must consider how to help system users locate, explore, and traverse relevant regions in table space.

At the global level, we guide system users who must find a path that traverses

the space from some given start point to a useful end point. Workflows are a mechanism to represent entire paths in this space. They represent a chained pipeline of transformations that can be applied to an input dataset. The built-in workflows are a small set of paths intended to span the space between a few landmarks of potential interest. DimStiller is designed to help its users find new, useful paths through table space, and save them as new workflows for later use.

At the local level, users also need to explore neighborhoods in table space: tuning the parameters of an individual transformation operator corresponds to searching for the most informative data table in the region of table space that is reachable with a single transformation that can be carried out by the chosen operator. DimStiller supports this local exploration through a chain of linked operator controls and views, so that users have immediate visual feedback about the effects of parameter tuning.

3.2 Users and Tasks

We now describe in detail the intended target user population for DimStiller, the questions that DimStiller is designed to help these users answer, and common techniques currently used to answer such questions. While these are not the only questions a data analyst might be interested in, we argue that they are a good place to start when considering a new dataset, especially one with unclear provenance that is not necessarily well curated.

3.2.1 Target User Population

DimStiller is aimed at bridging the gap between state of the art techniques in visually oriented dimensionality analysis, and the current practices of many users and potential users who do not already have deep knowledge of their data and the mathematics of reduction. Although dimensional analysis is sufficiently complex that we do not target casual users, we argue that this middle ground between utter novices and fully confident experts is a sizeable group that is underserved by the current set of available systems.

For example, a visualization researcher called on to help somebody analyze a dataset may be completely unfamiliar with the dataset characteristics and the tasks

of the researchers at the beginning of the analysis process. Furthermore, the person might be a visualization generalist rather than a specialist in the mathematical foundations of high dimensional techniques in particular. Another example is end users who have expertise in their own domain and the desire to do some dimensional analysis, but not deep knowledge of reduction mathematics. They might be developing algorithms to generate or process the data, and seek to evaluate the quality of their results or fine-tune parameter settings.

DimStiller is particularly aimed at providing major process improvements for data analysts who must deal with messy datasets that may have unclear provenance. By providing both local and global guidance through table space, we aim to support analyses that might otherwise seem too daunting and decrease the chances that non-expert users draw incorrect conclusions, supporting a qualitatively different analysis process than with previous tools. For those data analysts dealing with curated datasets where the meaning of each row and column are already fully understood, DimStiller may simply speed up a previously feasible, but slow, analysis process by automatically supplying a suite of visual results for the analyst to peruse.

3.2.2 Are My Dimensions Meaningful?

Sometimes an input dimension may actually contain little or no useful information at all. Because of this, it is important for an data analyst to be able to characterize the dataset in terms of which dimensions have useful information versus the "meaningless" ones. This understanding is not critical for downstream analysis algorithms in the same analysis session, since the mathematics of dimension reduction will handle creating the correct lower-dimensional projection. However, discovering that a given dimension is culled could reveal problems with the data source, with major upstream consequences in later iterations of the larger analysis loop: the data might be gathered differently, or the algorithms to generate it might be refined.

One such criterion is simply to check the underlying variance of the input dimensions and cull those beneath a small noise threshold. Another possibility is to use an information entropy cutoff.

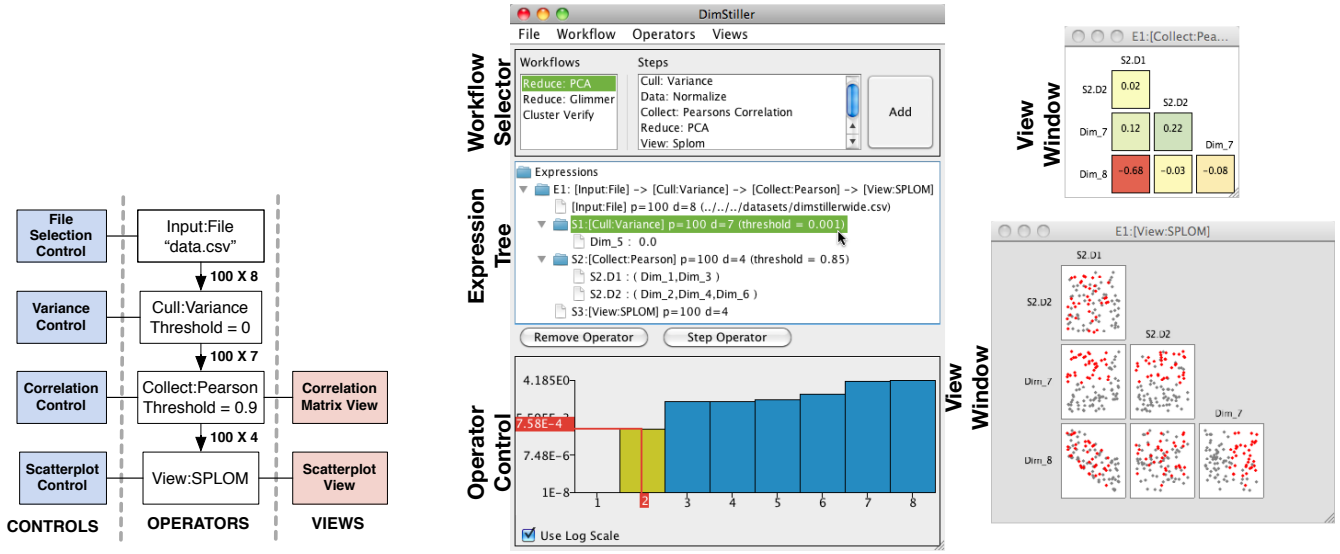


Figure 3.1: Left: The anatomy of a simple DimStiller expression. Input data is fed into a pipeline of operators that alter the dimensionality of the data. Each operator may or may not have controls or views. An Operator's control is displayed when it is selected in the Expression Tree. An operator's view is shown in a separate window, allowing side by side comparison between multiple views. Changing an operator's parameter in the control may produce a change to its output that is propagated across the expression using events that may travel both upstream and downstream from the operator. **Right:** The DimStiller interface for this expression, showing the `Collect:Pearson` and `View:SPLOM` views. The `Cull:Variance` expression is selected in the Expression Tree, so its control is visible.

3.2.3 How Do My Dimensions Relate to Each Other?

Much of multivariate statistical analysis is concerned with how the individual dimensions relate to each other. Many popular metrics such as Pearson's correlation coefficient measure pairwise relationships between individual dimensions. More holistic methods such as Principal Components Analysis determine how all the dimensions may actually express a smaller number of dimensions. Other methods uncover more complex nonlinear relationships between the input data, such as multidimensional scaling or many of the manifold-following variants.

3.2.4 Are My Clusters Real?

While the previous questions are related to dimensions, the question of cluster membership relates to the points. Clustering is the assignment of a unique label to specific regions of the input data's feature space and the points that occupy them. Cluster labels can be computed from any of a myriad of clustering algorithms.

Clustering relates to the input dimensionality in a reciprocal way. If the data analyst trusts the dimensional basis in which the data is represented, then point clusters in such a space will be considered real clusters with higher confidence than without such a trust. Likewise, if the data analyst is given a clustering that is trusted to be real, and the space in which the data is projected maintains the clustering's coherence, then this result increases confidence in the current dimensions. Thus, a clustering can inform the quality of a dimensionality reduction, and vice versa.

3.3 DimStiller Architecture

It is clear that an analysis tool that provides users with the ability to load their data into the system, transform their data with different analysis techniques, and scrutinize their data before and after these transformations would help users answer these questions. What is not clear is how such a tool should organize the results of applying the transformations or how to link these transformations together with visualizations to keep users focused on the analysis.

DimStiller organizes dimensionality analysis and reduction as a pipeline of transformations to a data table and linked views of it at different pipeline stages, as shown in Figure 3.1. The DimStiller model is based on an abstraction called an *ex-*

pression which encapsulates a sequence of transformations, called *operators*, that act upon tables of data where rows are points and columns are dimensions. Operators transform tables by adding, removing, or changing points or dimensions. Operators may have control panels and associated views that provide a visual representation of a table at that pipeline stage. All views are linked, and selections are propagated up and down the pipeline appropriately. A key aspect of the DimStiller architecture is the ability to instantiate expressions from pre-existing *workflows* that capture useful analysis patterns.

The DimStiller expression and operator abstractions were partially inspired by the Expression and Operator information visualization design patterns of Heer and Agrawala [48]. However, DimStiller expressions have a simple, linear topology that defers processing to the operators in contrast to the general tree structure suggested by the Expression pattern. Likewise, DimStiller operators are composable processing units similar to the Operator pattern, but compute general transformations and not necessarily visual mappings.

3.3.1 Input Tables and Dimension Types

DimStiller supports an abstract interface to data via a *table* model. Conceptually, a table can represent anything from physical entries in a disk file, to a cross-network database, or even evaluations of a simulator. The current implementation only supports simple file-based tables. There are two kinds of dimensions: Data, and Attribute. Data dimensions are either Quantitative or Categorical. Internally, both are represented by floating point values, and DimStiller maintains a lookup table to map floating point values to category symbols for Categorical dimensions. Attribute dimensions represent values such as color or selection that are used in views such as scatterplot matrices, but are ignored by purely data-oriented operators such as variance culling or dimension reduction.

3.3.2 Operators

Operators are functions that map an $n \times m$ table to an $n' \times m'$ table. That is, operators may add or delete points or dimensions, or change the value of any existing cell in the table. In the parlance of Section 3.1, they are the edges that connect

two nodes in table space. For example, the `Cull:Variance` operator removes dimensions with low variance from an $n \times m$ table. If any of the m table dimensions has variance below the user-controllable threshold, then the application of this operator would result in a new $n \times m'$ table where $m' < m$.

Every operator may have an associated control and/or an associated view, although neither is mandatory. Operator *controls* are GUI elements that permit users to modify operator parameters. Operator controls afford the user command over local search in table space. For example, Figure 3.2 shows the control panel for the above `Cull:Variance` operator, which has an interactive plot of the variance for each input dimension. The threshold parameter for culling is adjusted by clicking directly on the plot, and this operator does not have a separate view. Operator *views* are visualizations of the data table at that stage of the pipeline. Some operators are purely view-oriented, and do not transform the data table at all. For example, the view for the `View:SPLOM` is a scatterplot matrix, and its control panel only affects this visual display. In contrast, the `Collect:Pearson` operator has both a control panel with a slider to change the threshold, and a separate view with a matrix of colored boxes to show the pairwise correlations encoded with a blue-yellow-red diverging colormap.

The Operator namespace has a two-level structure, where a **family** has specific **instances**. The set of Operator families and instances, with notation `Family:Instance`, is:

- `Attrib:Color` Adds attribute color dim for views
- `Collect:Pearson` Join highly correlated dims
- `Cull:Variance` Identify/remove low-variance dims
- `Cull:Name` Identify/remove dim by name string match
- `Data:Norm` Normalize input dims
- `Input:File` Load comma separated value (CSV) file
- `Reduce:PCA` Estimate/reduce dimensionality with PCA
- `Reduce:MDS` Estimate/reduce dimensionality with MDS
- `View:SPLOM` Plot *n*-dim table using *n*x*n* scatterplot matrix
- `View:Histo` Plot dim distribution with histogram

The current families and operators serve to illustrate the potential of our approach to system architecture; the set of families is not exhaustive, nor is the set of operators within any family. The built-in set of operator families and instances can be extended by implementing new operators.

Cull and Collect Operator Families

Operators in the Cull family compute a specific criterion for each dimension and remove those dimensions that do not satisfy it. The criterion for the `Cull:Variance` operator is variance, and dimensions that fall beneath a user-specified threshold are culled. It can help users locate and eliminate dimensions whose variability is zero, or is small but non-zero because of noise. The `Cull:Name` operator allows users to selectively remove dimensions manually, for example to analyze only a subset of the input dimensions.

While the Cull family acts on individual dimensions, operators in the Collection family use pairwise criteria like covariance and Pearson's correlation coefficient. Rather than removing dimensions whose pairwise measures do not satisfy the threshold, these operators replace them with a single representative dimension for the collection, for example the average.

These operators can help users who may need to refine the processes used to generate their input dataset, as we discuss in Section 3.2.2. They are also useful for those whose analysis needs preclude the creation of synthetic dimensions.

Reduce Operator Family

A critical design choice in the Dimstiller architecture is that the `Reduce` operator family includes estimation of the intrinsic dimensionality of the space in addition to actually performing the reduction. The control for the operator, shown in the lower left of Figure 3.4, has a *scree plot*: a bar chart with the number of dimensions on the horizontal axis, and an estimate of the variability that would not be accounted for if the dataset were reduced to a space of that size on the vertical axis. The user then can make an informed choice when selecting the target dimensionality, by clicking on the plot at the desired threshold. (The definition of "intrinsic dimensionality" that we espouse is the smallest dimensionality of the set

of spaces in which the data can be embedded with distortion *less than some noise tolerance*, rather than *zero* distortion.) DimStiller supports users in experimentally determining the correct noise threshold, which differs between datasets, by interactive threshold adjustment.

Although scree plots are far from new, most previous toolkits do not explicitly couple them to the use of a reduction algorithm: users are simply expected to provide a number as input, with no guidance. Users who are not experts or are dealing with unfamiliar datasets will often have no idea of what a reasonable number might be. Worse yet, a significant number of reduction technique implementations are hardwired to blindly reduce to two (or three) dimensions, with no hint to the user that this choice might be inappropriate or misleading. Even the relatively sophisticated user who knows to run an estimator is often provided with the black-box output of a single number, rather than the detailed information for each possible number of cumulative dimensions shown in a scree plot [49]. Our design also has the benefit that users can see different estimates of intrinsic dimensionality in a lightweight and fast way with the scree plots, rather than the more heavyweight approach of reducing and then viewing the results in a scatterplot matrix. A related design choice is that the `View:SPLOM` view for showing a table is a scatterplot matrix rather than a single scatterplot. When the table has only two dimensions this view does of course show the case of only a single scatterplot, but when it has more the user is guided to see all of the information rather than an arbitrary subset.

While designing the architecture, we considered whether to have the estimation step separate from the reduction step. We ultimately decided that they should be coupled together into one module in service of the goal of providing guidance for the non-expert user. Understanding which estimators are appropriate for which reduction algorithms requires significant knowledge of dimensionality reduction: for example, nonlinear reduction methods should not be used in conjunction with linear estimators. Thus, we do not expect the middle-ground user to make that choice, reserving it for the designer of new operators.

The exact measure shown on the vertical axis of the scree plot depends on the operator instance. The `Reduce:PCA` operator shows the eigenvalues, and the `Reduce:MDS` shows the *stress* values of the embedding in each dimension. We use the CPU implementation of Glimmer [57] for both the MDS reduction and

estimation.

Attribute and View Operator Families

Attribute operators add attribute dimensions to the output data table of the operator. Attribute dimensions are interpreted by view and attribute operators and ignored by other operator families. The Color attribute operator creates an attribute dimension used for coloring to which it assigns values based on the values of numeric or categorical dimensions. The assignment of colors is performed either by linearly interpolating between two endpoint colors or by assigning colors to individual dimension values. The default colormap for categorical data, inspired by the work of Stone [108], has 10 bins; colors are repeated if there are more than 10 values.

View operators provide visualizations of their input data. The two built-in View operators are `View:Hist` for showing the distributions of individual dimensions, and `View:SPLOM` for showing pairwise relationships between dimensions. Both the SPLOM and the Histogram views provide global linked selection by creating an attribute dimension for selection, and displaying points with a nonzero selection attribute value in a default selection color. SPLOM views also use the color attribute dimension to color points.

3.3.3 Expressions

A DimStiller expression is the instantiation of an ordered list of operators applied to an input table. Figure 3.1 Left illustrates the elements of a sample expression, showing the associated views and controls for each operator. As the expression progresses, the data entries change value and the output table changes shape as it is progressively refined. Figure 3.1 Left also shows the relevant pathways for how information about input, view, and parameter changes moves across the expression. In our informal description of the table space graph of Section 3.1, the nodes are data tables and the edges are the transformation operators. However, in the DimStiller user interface, the more natural representation uses the dual graph, where operators are the nodes, and the edges represent the data flowing between them. The user is thus encouraged to focus on manipulating and understanding the transformations of the data.

3.3.4 Workflows

Workflows are templates for entire expressions that can be immediately created with a few clicks. DimStiller has a base set of workflows built in, and users can create their own by saving the list of operators in any active expression as a workflow.

A workflow contains a sequence of individual operator *steps*, and saved parameters associated with each operator. When a user instantiates a workflow, a new expression is produced unique to a given input table. Because many operators may result in time-consuming computations, only the first operator in a workflow computes its output upon workflow instantiation, with subsequent operators greyed out in the user interface. Users choose when to progress to activating the next step, possibly after adjusting parameters at the current step, with a `Step Operator` button. Heavyweight operators downstream will thus only initiate their computations on data tables that may be much more compact than the input table due to reduction at upstream stages.

The built-in workflows are designed to help users begin to answer the set of questions that we identified in Section 3.2.1, as a proof of concept that this style of guidance can help middle-ground users. We do not claim that they are the only way, or even the best way in all cases, to answer these questions. Workflows provide optional global guidance; they are not mandatory. Power users have the flexibility to build up new expressions directly in DimStiller by choosing individual operators from the currently loaded set.

The set of workflows built into DimStiller are:

- **Reduce:PCA.**
  - `Cull:Variance`→
  - `Data:Normalize` →
  - `Collect:Pearson`→
  - `Reduce:PCA`→
  - `View:SPLOM`
- **Reduce:MDS.**
  - `Cull:Variance`→

- `Data:Normalize` →
- `Collect:Pearson`→
- `Reduce:MDS`→
- `View:SPLOM`

- **Cluster Verify.**
  - `Attrib:Color`→
  - `Data:Normalize`→
  - `Reduce:PCA`→
  - `View:SPLOM`

3.3.5 DimStiller Interface

Figure 3.1 Right shows a screenshot of the DimStiller session containing the expression diagrammed in Figure 3.1 Left. The visual structure of the DimStiller interface, with views and controls for each operator, encourages the user to examine the individual operators, adjust their parameters, and observe the effects on the resulting transformations in the visual representations.

The `DimStiller` window on the left contains the Workflow Selector at the top, with the Expression Tree underneath and an operator control panel on the bottom. Two view windows are visible on the right, a scatterplot matrix for the `View:SPLOM` operator and the correlation matrix for the `Collect:Pearson` operator. The Expression Tree shows that the input file `dimstillerwide.csv` contained a table with 100 rows of points and 8 columns of dimensions. In this example dataset, `Dim_1` and `Dim_2` are independently sampled from a uniform distribution between 0 and 1, `Dim_3` is a scalar multiple of `Dim_1`, `Dim_4` and `Dim_6` are scalar multiples of `Dim_2`, `Dim_5` is set to all zeros, `Dim_7` and `Dim_8` are linear combinations of both `Dim_1` and `Dim_2` with a uniform noise term.

The first `E1` operator `S1` is `Cull:Variance`. The user has clicked on the first nonzero dimension in the scree plot, resulting in a threshold value of 0.0007 (rounded to 0.001 for display in the tree). The summary line for `S1` in the Expression Tree shows that the output table of `S1` has 7 dimensions as opposed to the 8 dimensions that were input to the operator, and the expanded details beneath

show that Dim_5 is the one that was culled. The S2 operator collects dimensions that are correlated with the threshold of 0.85, resulting in a table of 4 dimensions whose pairwise correlations are shown with color in the top view. The expanded details shows Dim_1 and Dim_3 are now represented by new synthetic dimension S2.D1, and the remaining three are now represented by S2.D2. The last operator is View:SPLOM, and the bottom view shows the scatterplot matrix. The user selected some points in one plot, and they are colored red in all of the linked plots.

Workflow Selector

The Workflow Selector displays the available workflows and allows the user to select one and create a new expression from it. Selecting a workflow fills the adjacent list box with the sequence of steps for the user to inspect. If the user chooses to activate that workflow by clicking the Add button, DimStiller applies the workflow steps to the currently selected expression, making those operators visible in the Expression Tree.

Expression Tree

The Expression Tree is a three-level tree widget that lists all open expressions. At the top level, expressions are described by a short text summary where each new operator X is appended on the right of the text string as $\rightarrow$ [X], where X is a very terse label. When the user drills down to the next level, the individual operators that comprise the expression are listed with a concise yet complete text summary that includes the size of the output table produced by the operator in terms of points and dimensions, as well as any operator parameters that are set to non-default values. The third and final level of detail is only added if an operator modifies the output dimensionality, namely the list of the dimensions modified by that operator and any details that relate the input and output dimensions. For example, in Figure 3.1 the expansion of the Collect:Pearson operator shows two of the synthetic dimensions and names of the original dimensions collected together.

Operator Control Panel

Each operator may have a control panel that lets the user adjust its parameters. Called operator controls, they afford the user with the means to locally search for a meaningful region in table space. When an operator is selected in the Expression Tree, its control populates the operator control panel region at the bottom of the main DimStiller window. Only one operator can be selected at a time. Operator controls visible in this paper include the `Cull:Variance` control shown in Figure 3.1 and the `Reduce:PCA` control shown in Figures 3.4 and 3.5.

View Windows

Each operator also may have an associated view. When an expression is loaded or created, the associated views open up as individual windows to support side-by-side comparison across operators within the same expression or even across different expressions. All the views are created using the Processing language, but new operator view plugins could be created using any graphical toolkit that can interface with Java. The built-in operators that have views are the `View:Histo`, the `View:SPLOM` shown in Figures 3.1 and 3.2, and the box matrix showing pairwise relationships for the `Collect:Variance` operator in Figures 3.1 and 3.3.

3.4 Case Studies

We now describe how the DimStiller architecture facilitates the task of dimensionality reduction and analysis through case studies on real-world data. We use the built-in workflows to construct expressions that inform users about the character and relationships of the dimensions and clusters of the datasets[1].

3.4.1 Sustainability Simulation: Measuring Variability

Our first case study focuses on a sustainability simulation dataset containing a large collection of simulated results of government policy decision scenarios [78]. The 294 dataset dimensions represent the environmental and societal indicators affected by the policy decisions. A first attempt to analyze this data using pre-existing tools

[1] The video at http://www.cs.ubc.ca/labs/imager/video/2010/dtil4_2.mov shows the look and feel of interactive sessions with these datasets.

fell prey to the *reduce to 2 and plot* pitfall discussed in the introduction to this chapter. The simulation is agglomerated from many subpieces originally designed for varying purposes, rather than being carefully constructed from custom components that would dovetail seamlessly. The simulation designers thus did not have a clear idea of the intrinsic dimensionality of this dataset. They did know what their dimensions were, with meaningful labels for each such as `Cost of Living` and `Air Quality`. However, they thought it was possible that some indicators always had the same value across the entire dataset. They were also interested in learning about how the dimensions related to each other: they suspected that many indicators were highly correlated, but did not know the number of equivalence classes or which indicators were in each group. They were also curious whether automatically computed correlation groups would match with their intuitions about indicator relationships.

The first choice to make when using DimStiller is whether to construct our own expression from scratch by individually choosing operators, or to instantiate a workflow from the existing list. Since we are interested in finding the intrinsic dimensionality of the space as well as any correlations, a workflow in the `Reduce` family seems to be a good match, and we start with `Reduce:PCA`.

Figure 3.2 shows the control of the `Cull:Variance` operator that plots the sorted variances of the dimensions, with the log-scale option selected to emphasize small values. We notice that there are indeed many zero-variance dimensions, and click the scree control to remove these 34 dimensions, leaving 260 in the output table. The researchers could now drill down in the Expression Tree to see the names of the potentially problematic culled dimensions. They now know that either the input policy choices used in this run of the simulator did not effectively span the indicator space, or that there are unforeseen interactions between simulator components.

We click the `Step operator` button to activate the next workflow step, the `Data:Norm` operator. Both reduction workflows include a normalization step to guide users who may be unaware of the effects of transforming dimensions with differing scales of variation. The Expression Tree in Figure 3.4 shows that we chose to normalize using Z scores, so the operator subtracts the mean and divides by the standard deviation. We then step to activate the `Collect:Pearson` oper-

ator, which gathers highly correlated dimensions. Even the most stringent possible threshold setting of 1.0 for perfect correlation results in a drastic reduction of the number of dimensions: from 260 to 147. Figure 3.3 Left shows the correlation matrix view, where only a small fraction of the boxes are visible without scrolling. Relaxing the threshold to a more reasonable value of 0.8 results in the view shown in Figure 3.3 Right, where the number of dimensions in the table is reduced to 22. Again, the simulation designers could now drill down in the Expression Tree to see the names of which dimensions were collected together, in order to check whether the automatic computations match their intuitions about the expected behaviour of the simulator.

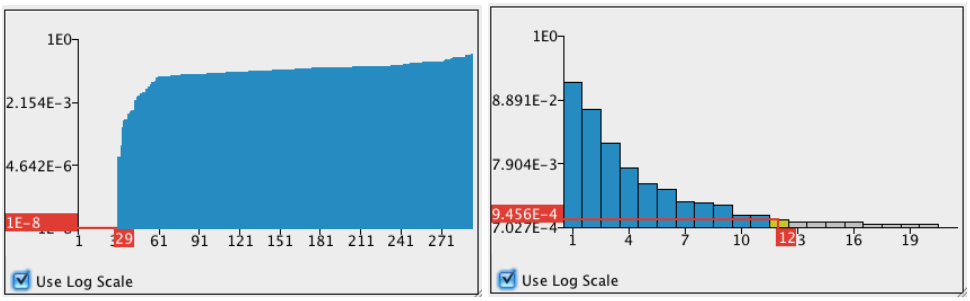


Figure 3.2: Interactive DimStiller controls. **Left:** The `Cull:Variance` operator control displays a sorted list of the dimension variances. Many have zero variance, indicating potential problems in the choice of input variables to the simulator or the operation of the simulator itself. Log-scaling of the variances emphasizes small values. **Right:** The scree plot for the nonlinear `Reduce:MDS` shows an intrinsic dimensionality of 12 dimensions, versus the 16 dimensions found by linear methods shown in Figure 3.4.

Finally, we would like to determine whether the intrinsic dimensionality of this space is even smaller than the 22 dimensions of the table after culling and collecting, and if so reduce to that space. The `Reduce:PCA` operator constructs a linear projection of the data into a subspace that minimizes distortion. Our motivation for doing the reduction is to observe the distribution of the eigenvalues corresponding to the major axes of the simulation output in a subsequent scree plot. The control view in Figure 3.4 shows the scree plot in the PCA control, and we see that the eigenvalues approach zero between 12 and 18 dimensions. Mousing over dimen-

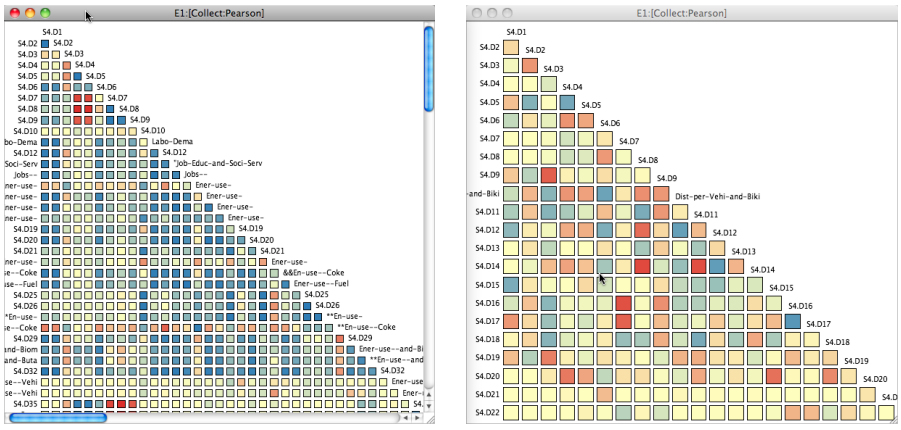


Figure 3.3: The DimStiller `Collect:Pearson` operator view shows the correlation matrix with a diverging color scale ranging from blue at the positive end, through yellow for independent pairs, to red for negative correlation. **Left:** The perfect correlation threshold of 1.0 reduces the table from 260 to 147 dimensions. **Right:** Relaxing to a more reasonable threshold of 0.8 reduces the number of dimensions to just 22, all visible without scrolling.

sion 16 shows that it corresponds to a very small noise threshold of 0.001, and we click to select that value. We then click the `Step Operator` button and activate the `View:SPLOM` operator which brings up the scatterplot matrix view.

In order to facilitate viewing the original lattice structure of the data in the input space, we insert an `Attrib:Color` operator into the expression after the `Input:File`. The `Attrib:Color` creates an attribute dimension containing color information derived from a single, user-selected input dimension. We select the initial dimension as the dimension from which colors are derived. The resulting colored SPLOM is shown in Figure 3.4 in the top right view window, labeled `E1:[View:SPLOM]`.

The original analysis of the dataset was done by projecting the data down to two dimensions using multidimensional scaling. We quickly replicate this analysis in DimStiller so that we can compare the results directly. We reload the same data into a new expression `E2`, adding a `Attrib:Color` Operator and using the `Reduce MDS` workflow. The operators in this workflow are the same as in the

previous analysis except for the Reduce operator, and we use the same settings as before. We check the scree plot for this case, and find that only 12 dimensions suffice with this nonlinear reduction, as shown in Figure 3.2. Then, to illustrate the *reduce to 2 and plot* pitfall, we use the `Reduce:MDS` Operator to select 2 as the output dimensionality. The SPLOM view at the bottom right of Figure 3.4 shows only the single scatterplot. The regular, lattice-like structure visible in the SPLOM above is completely hidden, and we see only an undifferentiated blob.

This analysis session with DimStiller shows that although the simulator produces hundreds of outputs, dozens have zero variance, most of the remainder are highly correlated, and the data can be represented with only around a dozen dimensions without losing information. Although in theory this full analysis could have been carried out with existing tools like MATLAB, and bits and pieces of it were done over the course of a few years, in practice we did not have a complete picture of this messy real-world dataset until we could analyze it with the DimStiller system.

3.4.2 Computational Chemistry: Evaluating Clusters

We now examine a 30-dimensional computational chemistry dataset. The individual dimensions of this data measure physical properties of chemicals such as molecular weight and the number of bonds they possess. The dataset includes a cluster membership dimension with 236 clusters of the data produced by a commercial clustering package. The goal of the chemist who work with this dataset is to evaluate the quality of the clustering.

The `Cluster Verify` workflow is appropriate for this goal, so we instantiate a DimStiller expression from it. After loading the data, we use the `Color` operator control to choose which dimension we use for the categorical colormap. By default, the color operator culls the dimension by which it colors the points; including this cluster membership dimension in the downstream analysis would usually skew the results.

After adjusting color settings, we activate the `Data:Norm` operator which normalizes the dimensions to Z scores. We then activate the `Reduce:PCA` and observe its control. The scree plot of the eigenvalues, visible in Figure 3.5, shows

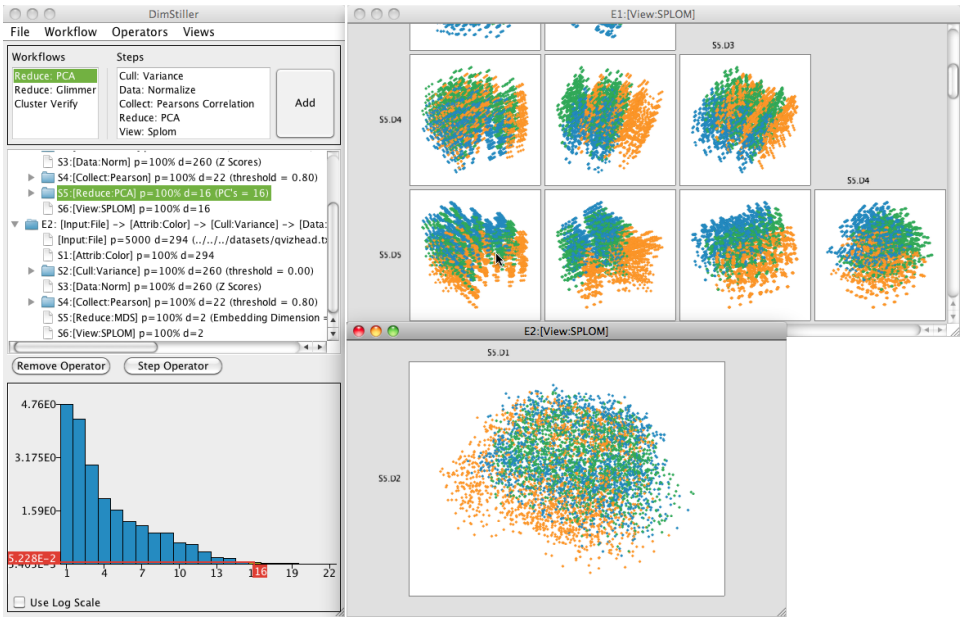


Figure 3.4: DimStiller `Reduce:PCA` control and scatterplot. The scree plot in the `Reduce:PCA` control shows an estimate of the intrinsic dimensionality as between 12 and 18 dimensions, and we have interactively selected 16 as the threshold. The top scatterplot matrix shows a result with a visible lattice structure. In contrast, the bottom view shows the pitfall of reducing to just two dimensions using MDS, where an undifferentiated blob gives a misleading impression of no such structure.

an exponential drop off in magnitudes. This plot strongly suggests that the majority of the variance of the data resides in a lower dimensional space than the input dimensionality. Standard practice is to select the "knee" of the value drop-off curve as a good candidate for target dimensionality. We select 3 and then activate the `View:SPLOM` operator.

The resulting scatterplot matrix of the `View:SPLOM` Operator, also visible in Figure 3.5, reveals several interesting cluster structures in the data. In the bottom row of two scatterplots, we observe clear separation of several clusters of points. In contrast to the spatial separation, some color labels appear to span gaps in the scatterplots. A single categorical color scheme of course cannot possibly show over 200 clusters with distinguishable colors, so DimStiller uses a repeating palette.

To check whether some of the adjacent clusters with differing labels might have the same color by chance, we select the `Attrib:Color` Operator again in the Expression Tree to bring up its control. The `Permute Colors` button permutes the order in which colors are assigned to categories. After trying a few different permutations of the color scheme we conclude that the phenomenon we saw was not just an artifact; several cluster labels do indeed span these observed clusters. This result gives strong, albeit not conclusive, evidence that there may be better clusterings.

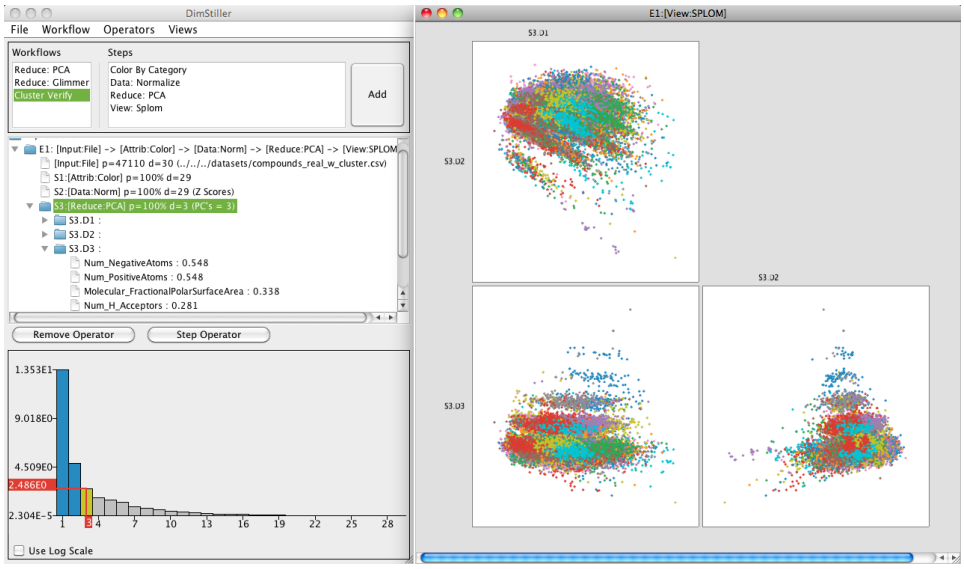


Figure 3.5: Running the DimStiller `Cluster Verify` workflow on a computational chemistry dataset. The scree plot in the lower left shows that most of the variability resides in a low dimensional subspace. We choose a threshold of 3 dimensions at the "knee" in the plot. We see in the colored SPLOM view that while the clusters are spatially coherent, they do not reflect the spatial separation in this projection, suggesting that this clustering is not the most appropriate.

3.5 DimStiller Deployment

DimStiller was produced as part of a larger, multi-pronged research initiative designed to study high-dimensional analysis. In the context this initiative, DimStiller

was our first attempt to architect a software solution to the major issues that we believed were facing working data analysts: selecting appropriate methods and appropriately tuning their parameters. Another key thread of that larger research initiative was a multi-year qualitative field study in which I participated as a co-author [98]. This qualitative study provided an ideal opportunity to deploy DimStiller to working data analysts from a variety of disciplines and informally gauge the utility of the software to their problems.

To deploy the software, we introduced some of the data analysts in the study to DimStiller and encouraged them to try to incorporate using the software in their own data analysis tasks. This limited deployment was instructive in several key ways. First, it highlighted use cases of high-dimensional analysis where DimStiller was not an appropriate tool. Second, it brought to our attention incorrect assumptions of DR algorithms that needed to be addressed before a tool like DimStiller could even be applied. Finally, the deployment shed light on external factors that can greatly affect the success and future development of software systems. In the reminder of this section, we describe these different results of the deployment in more detail.

In one deployment use case, detailed more completely in the qualitative field study as FISHPOP [98], a user applied DR to a technique more appropriate to a different high-dimensional technique called sensitivity analysis. Sensitivity analysis is outside the scope of the goals of both DimStiller and DR. This case represents a mismatch between the user's understanding the goals of existing DR workflows and the user's underlying task of gauging the sensitivity of output dimensions values to small changes in input dimensions.

In two other different deployment cases, users uncovered incorrect assumptions in DR algorithms that needed to be addressed before DimStiller could effectively handle their input. For example, one DimStiller user was a data analyst analyzing distance-matrices resulting from the co-citation analysis of relevant terms, where distances between terms were the result of a costly database calculation. Here, the data analyst was prohibited from doing a full analysis without performing an impractical, months-long database calculation. Another user wanted to use DimStiller to perform DR on vectors derived from document collections. DimStiller was not equipped to process the extremely high-dimensional datasets resulting from ana-

lyzing document data. Both of these issues were less about inherent limitations of DimStiller itself than about DR in general, as any DR algorithm or tool would encounter similar difficulties when applied to the same data.

For these users, important algorithmic obstacles needed to be surmounted before applying the workflows for which DimStiller was designed. By applying DimStiller to perform DR on practical problems, these DimStiller users highlighted the issues that we tackle in the remainder of this thesis. In the case of the co-citation analyst, we developed an adaptive algorithm framework to handle the case of calculating DR in the presence of costly distance functions. For the case of Document Data, we developed DR algorithms to efficiently leverage their high-dimensional structure. Both of these issues, and the algorithms we developed, are covered in more detail in Chapters 6 and 4 respectively.

Missing from our deployment were any unambiguous success stories, where users reported insights and analysis process improvements from using DimStiller. We believe most users in the deployment did not adopt DimStiller into their analysis workflow, using the software only during the period of our interviews. This lack of adoption derives less from the design of the software itself, with which no user in our deployment had issues, than from two important external factors we observed: difficulty of software integration and our own lack of evangelism.

By software integration, we refer to the ease with which the inputs and outputs of software systems connect with other software tools. For example, one of our collaborators in the qualitative study remarked how she would be more likely to use and recommend DimStiller if it could be easily incorporated into her R workflow [86]. DimStiller was implemented as a standalone Java tool with a single generic input and output format. Thus, considerable software engineering and Java expertise is required to build and incorporate new operators to the tool. In the future, we believe guidance-oriented analysis tools should integrate themselves into software platforms with a substantial base of existing techniques and users. Doing so both mitigates the initial barrier to adoption and maximizes the ease with which new or existing techniques can be folded into the tool.

Another factor affecting the adoption of systems like DimStiller is the effect of software evangelism. Beyond the initial deployment and publishing of an article describing DimStiller [58], we made no further effort to deploy to other communi-

ties or improve the software, having shifted our research focus on addressing the issues that appear later in the thesis, such as handling costly distances or document data. We discuss the effects of software evangelism and other ways to increase the impact of research more fully in Section 7.3.2 of this thesis.

Chapter 4

The Geometry of Fast Document Queries: Implications for Visual Analysis Algorithms

The field of Information Retrieval, or IR, focuses in part on improving the accuracy and speed of the task of *querying* document databases [75]. Search querying is performed by millions of users of varying skill each day. To do so, a user poses a set of terms to the query engine, which then provides a 1-dimensional list of results back to the user, sorted in terms of predicted relevance to the query. Search query results computed on databases of billions of text records are often returned with processing time in less than a second, making it one of the more popular and scalable computer algorithms.

One reason Information Retrieval algorithms for search querying can perform so efficiently lies with an important property of the underlying data. We informally name this property the **Mostly Disconnected** property, or **MoDisco** for short. Put briefly, MoDisco datasets are those datasets with a large number of dimensions, but where each point has a nonzero in only a subset of these dimensions, and each dimension is set to nonzero for only a subset of the points. That is, not only are the

points sparse, but so are the dimensions themselves.

The MoDisco property has implications for the efficiency and scalability of query algorithms. High-quality query results can be computed by processing only a small subset of the total dataset without degradation in quality, by organizing the data into data structures such as inverted files and traversing these structures in order of query-term impact [137]. We discuss the MoDisco property in detail and its relationship with query algorithms in Section 4.1.

In contrast to querying, visual analysis often involves a more open-ended *exploration*. The document exploration task remains the purview of expert-level analysts in fields like intelligence analysis, law, and computational journalism, where it is important to quickly understand large, unlabelled collections of text documents [25, 73]. In this context, exploratory methods tend to encounter scaling difficulties, and the characterization of *large* is often ascribed to datasets well under ten thousand documents [52, 133].

We argue that the MoDisco property also has important implications for document exploration algorithm design in visualization. By interpreting the MoDisco property in terms of distance geometry, we can make useful statements about the construction of distance-based analysis algorithms, like clustering and dimensionality reduction, used to explore these large, complex datasets.

As the first contribution of this chapter, we identify and describe the following three implications for algorithm design in this paper.

- Nearest Neighbor search of MoDisco data can more efficiently be done with search-index based queries rather than generic distance-based approaches.
- Distance matrices of MoDisco data can be calculated and stored efficiently with low approximation error.
- Dimensionality Reduction algorithms for MoDisco data should use methods based on local attraction and global repulsion.

While distance matrices, nearest neighbor search, and dimensionality reduction are useful in a variety of analysis pipelines [58], it is not self-evident how such implications can be directly utilized to formulate new algorithms. As a followup

contribution, we also describe three techniques, two new algorithms and a modification of an existing algorithm, for incorporating these implications into the design of algorithms intended for the visual analysis of documents:

- All Pairs Queries - we present a new algorithm for nearest neighbor search of MoDisco data, which we use to construct an efficient Distance Matrix approximation.
- Fast Cluster Tree - we present an algorithm for producing a hierarchical cluster tree from our Distance Matrix approximation in $O(N \log N)$ steps.
- MD-SNE - we introduce the MD-SNE algorithm for dimensionality reduction of MoDisco data.

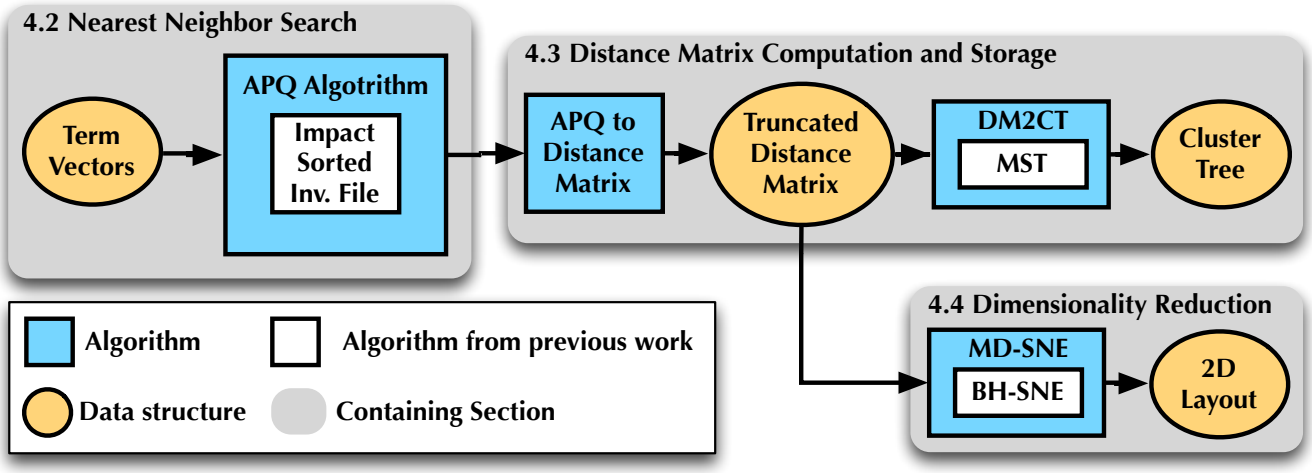


Figure 4.1: Chapter diagram describing the relationship between our algorithmic contributions, the previous work used as algorithm components, and the data structures created and used by these algorithms. **Section 4.2** describes the *All-Pairs Query*, or *APQ*, algorithm, which uses impact-sorted inverted file queries to process a set of term vectors to yield a set of nearest neighbors for each vector. In **Section 4.3**, we describe how this set of nearest neighbors can be input into the *APQ to Distance Matrix* algorithm to produce a Truncated Distance Matrix, a low-space Distance Matrix approximation. We then demonstrate how to use the Truncated Distance Matrix efficiently with the *DM2CT* algorithm which uses the Minimum Spanning Tree (MST) algorithm as a component to produce a Cluster Tree. **Section 4.4** describes the MoDisco-SNE (MD-SNE) algorithm for dimensionality reduction.

Figure 4.1 summarizes the contributions of this paper with a diagram showing the relationship between our implications, our algorithmic contributions, and previous work used as algorithm components.

After first defining the MoDisco property and how it relates to document term-vector datasets in Section 4.1, the remainder of the paper is organized around the three different algorithm design implications. In each of Sections 4.2, 4.3, and 4.4, we explain the design implication, present an algorithm incorporating the design, and compare the results of the algorithm with competing work.

4.1 The Mostly Disconnected Property of Term-Vector Datasets

Query systems often use the vector space model, storing documents as term vectors [96]. Here, document refers to a string containing the content of a single text file. The document string can then be broken into terms: n-grams of words. Term vectors reside in a space whose dimensions are all of the terms in the entire database, and whose weights indicate the importance of a given term in identifying the underlying document. Using such vectors is not only more efficient than processing the varying-length document content during a query, but it permits us to conceive of the documents as points residing in term-space. We can then think about documents spatially, just as we would any other distribution of high-dimensional data points.

In this section we first describe the method by which documents are transformed into term vectors. We then present a set of term-vector datasets derived from real-world document corpora. Next, we indicate how the transformation method induces the mostly disconnected property by measuring properties of our benchmark datasets. Finally, we explain the method by which query algorithms are able to extract performance from this property.

4.1.1 Term-Vector Datasets and TF-IDF

An influential method for determining the weights assigned to term-vector dimensions in the IR literature is the TF-IDF method [95]. TF-IDF, short for Term Frequency times Inverse Document Frequency, is a scheme that assigns a large weight

to terms that appear in a document frequently, but then discounts that weight by how often the term appears in other documents. The weight of common words is reduced and unique words increased. Using the probabilistic model of IR, TF-IDF has already been shown to be an optimal model of document *relevance* under a set of simplifying assumptions [89]. In contrast to this probabilistic perspective regarding document relevance, we develop a spatial intuition of why TF-IDF is efficient in document queries.

Term vectors are typically normalized to unit length, to control for document size by forcing long documents with large term frequencies to carry the same weight as short documents with small term frequencies. This normalization implies that document points reside on the surface of a hypersphere. The fact that term weights are always nonnegative restricts their distribution further to the positive orthant of this sphere.

The common distance metric used between term vectors is the cosine distance metric [95]. The cosine distance between vectors a and b is equal to $1 - a^T b$, or 1 minus the dot product between the vectors. While other related metrics exist, we focus on the cosine metric due to its simple geometric interpretation as one minus the length of the linear projection of vector a onto b.

4.1.2 Benchmark Datasets

Our explanation of the Mostly Disconnected property requires demonstrating and measuring the sparsity characteristics of a set of varying benchmark datasets. The benchmarks in this paper are run on a set of seven datasets. Four of them are collections of MoDisco term vectors generated using TF-IDF term weights from text databases; three are real-world datasets and one is synthetic. The `metacombine` dataset is produced from metadata tags harvested from a disparate collection of digital libraries [67]. The `warlogs` and `cables` datasets are subsets of documents from larger Wikileaks collections [59]. The `conspiracy` dataset is harvested from the conspiracy section of a BBS textfile collection[1]. We include the synthetic `simplex` dataset as an extreme example of a MoDisco dataset, which is a set of 1000 points equally distant from each other in 999-dimensional space. An

[1] http://www.textfiles.com/ Last visited on July 10, 2013

Name	Points (N)	Dims (M)	MoDisco?
metacombine	28K	28K	Y
cables	7K	66K	Y
warlogs	3K	4K	Y
conspiracy	1K	33K	Y
simplex	1K	1K	Y
grid	1K	2	N
mnist	2K	784	N
blobs	6K	5	N

Table 4.1: Benchmark datasets, with size in terms of both points and dimensions. The first five are MoDisco; the second three are not.

$(N-1)$-dimensional simplex dataset is equivalent to an $N \times N$ identity matrix $\mathbf{I}$.

The term vector datasets in this thesis are constructed with the following pipeline. First the text is split into ordered sets of words. These word sets are then stripped of a designated list of stop words. The remaining words in the set are then used to create bigrams, pairs of subsequent words. Term counts for each document are then constructed by counting the occurrences of each bigram in each document. These term counts are finally re-weighted by TF-IDF and then normalized to unit length. Stray has a more detailed description of the term-vector generation process including justifications for not using alternative approaches, such as entity recognition [109].

For contrast, we also include three datasets that are not MoDisco, also a mix of real-world and synthetic. As a synthetic example of a linear manifold, or flat surface, we include a regularly sampled 2-D lattice called `grid`. The real-world `mnist` dataset contains the combined test sets of digits 1 and 2 in the MNIST handwritten digit dataset[2], as an example of a high-dimensional, real-world dataset with two meaningful clusters. Finally, the synthetic `blobs` dataset is a collection of 6 clusters of 1000 points randomly generated from 6 separate 5-dimensional, Gaussian distributions. Table 4.1 provides the characteristics of these benchmark datasets in terms of the number of points N, the term space dimensionality M, and whether it has the MoDisco property.

[2]downloaded from http://yann.lecun.com/exdb/mnist/ Last visited on July 10, 2013

4.1.3 The Mostly Disconnected Property

The intuitive definition of the MoDisco property is that each point mostly resides in a small subspace of the full term space, represented by a small number of key terms. Furthermore, these same key terms weigh highly with only a small subset of the other data points. To formally define the MoDisco Property, we first introduce a descriptive statistic measuring the sparsity of both points and dimensions of the dataset with N points and M dimensions. First, we define the thresholded sparsity of a point i to be

$$P_i(t) = |\{x_{ij} | x_{ij} > t, j \leq M\}|/M$$

and the thresholded sparsity of a dimension j to be

$$D_j(t) = |\{x_{ij} | x_{ij} > t, i \leq N\}|/N$$

for some threshold $0 < t < 1$. The purpose of introducing threshold t is to remove vector terms from the data with very low weight. Low-weight terms often contribute little to the distance calculation between two points, and can potentially make operationally sparse data appear dense. In our analysis, we select $t = 0.1$, but it holds for a wide set of values. We can now define two *worst-case* sparsity measures

$$\begin{aligned} maxP(t) &= \max\{P_i(t) | \forall i \in [1,N]\} \\ maxD(t) &= \max\{D_j(t) | \forall j \in [1,M]\} \end{aligned}$$

Given these worst-case thresholded sparsity measures, we now say that a dataset is MoDisco when **both** $maxP(t) < B$ and $maxD(t) < B$. Here, $B \in (0,1)$ is some bound on the desired sparsity of the data; a smaller B imposes stricter sparsity requirements on the data. These two inequalities are descriptive statements about the *worst-case* sparsity of any single point and dimension in the data. They implicitly describe how the design implications we present in this paper hold true and are not computed explicitly by any of the presented algorithms. As a side note, it is not meaningful to make statements differentiating the *average* sparsity of points versus dimensions of a dataset, because these measures are equivalent.

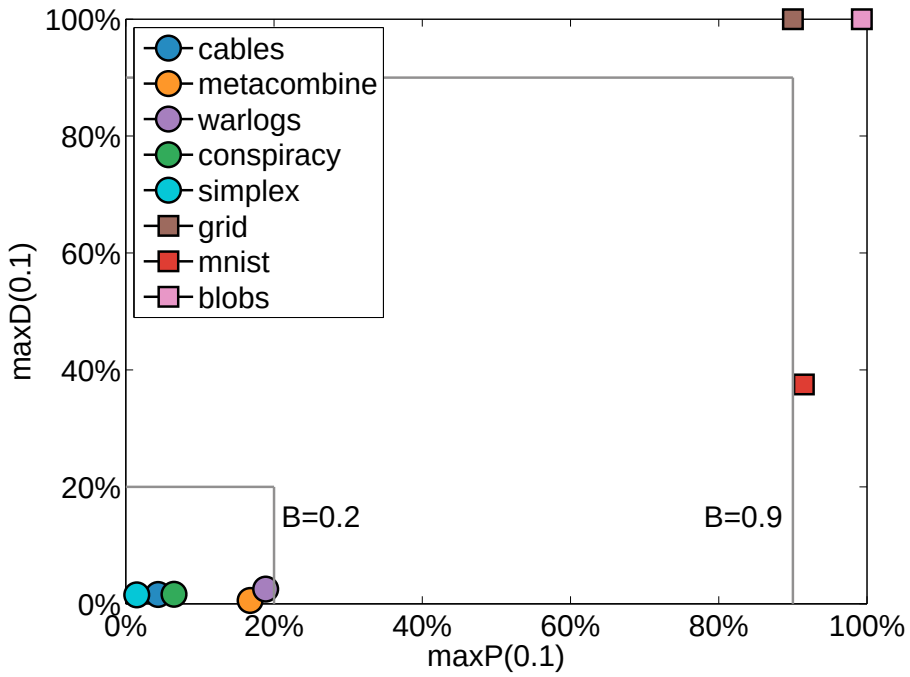


Figure 4.2: Plot of MoDisco statistics $(maxP(0.1), maxD(0.1))$ showing clear separation between MoDisco datasets (circles) and non-MoDisco datasets (squares). The boundary B defines the region of MoDisco datasets, we present two extreme possibilities, $B = 0.2$ and $B = 0.9$, that agree with our datasets. In the MoDisco case, each point has a small number of nonzero dimensions and each dimension is expressed by a small number of points.

Figure 4.2 shows a plot of the points $(maxP(0.1), maxD(0.1))$ for the suite of 7 benchmark datasets. The five MoDisco term-vector datasets, represented as circles, are distinctly separated from the other 3 non-MoDisco datasets, represented as squares. Empirically, the worst case sparsity bound B can take values anywhere between 0.2 and 0.9, demarcated by grey lines on Figure 4.2. Given that we observe significant separation between MoDisco and non-MoDisco datasets, we do not prescribe any specific value of B.

There is an intuitive explanation of why term-vector schemes like TF-IDF induce the MoDisco property. The TF component defines the subspace in which

each vector resides, while the IDF component reduces the subspace weights below t for those dimensions shared by many vectors, effectively reducing the subspace dimensionality and disconnecting most vector subspaces from each other (unless they possess sufficient TF weight to compensate). Without the IDF discounting factor, most of the set of subspaces in which vectors reside would connect by intersecting across the dimensions representing common terms.

By definition, the MoDisco property implies that the distribution of the set of cosine distances between points is highly skewed toward 1. Two randomly sampled documents from a MoDisco data set will share few, if any, terms on average. This phenomenon has been observed and noted in the IR literature on query datasets [84]. Figure 4.3 verifies this phenomenon by showing distance histograms for benchmark datasets: the histogram of the MoDisco `warlogs` dataset is most similar to the high-dimensional simplex case, rather than the other three non-MoDisco examples of a regular grid and the two cluster datasets.

4.1.4 Implication for Efficient Queries

MoDisco data are highly amenable to efficient search queries, due to the mapping of the data into fast, orderable search indices. We now explain how these indices are constructed, and how the MoDisco property affords efficient lookup of relevant documents.

An important search data structure is called an Inverted File [137]. It is so named due to the inversion from the standard term-vector storage that maps a document to a list of terms, to a mapping from a single term to a list of documents that store that term. Query processing using the inverted files need not compare the query to the entire document corpus, but only to those documents that share the query terms.

Improved traversal of the inverted file can be done by performing an impact sort, or simple re-ordering of both the index lists and the order of their traversal. Impact ordering has been shown to compute higher-scoring query results earlier than low-scoring queries [3]. The advantage of computing these higher-impact queries first is that we can truncate the query computation after computing the top-scoring queries.

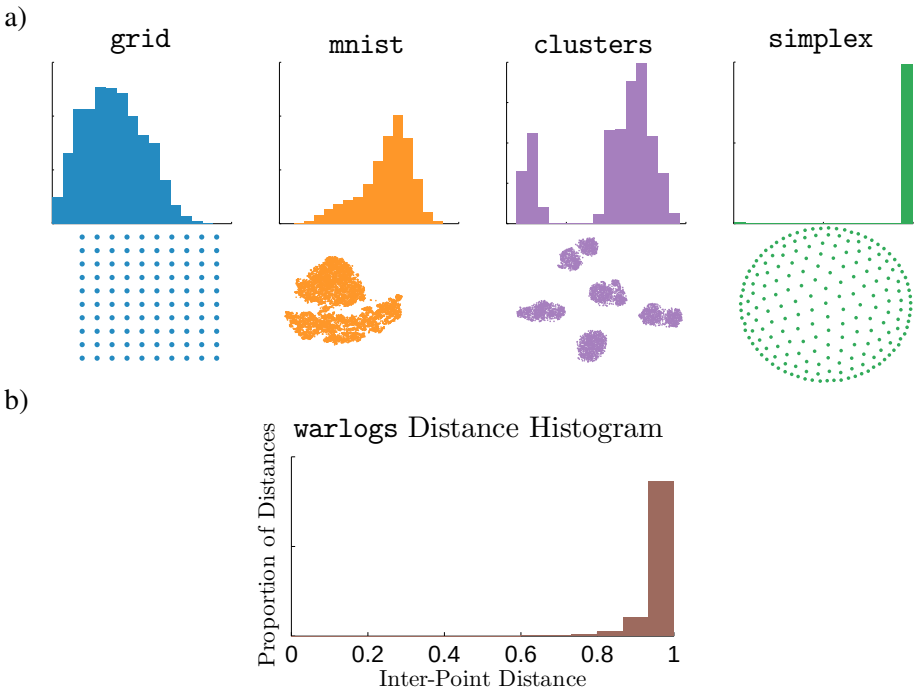


Figure 4.3: Distance Histograms of different dataset types. a) Histograms of distances (top) and layouts showing dataset geometric structures after dimensionality reduction (bottom) for non-MoDisco datasets grid (blue), mnist (yellow), blobs (purple), and the synthetic extreme MoDisco case of the simplex (green). b) Real-world MoDisco dataset `warlogs` (brown) histogram matches the simplex case the best.

The cost of impact-ordered traversal is that the accuracy of the query results are approximate. Many efficient query-evaluation systems exist that produce exact results [34]. In this paper, we focus on impact-ordering techniques for their user-controlled flexibility of the speed and quality tradeoff.

Impact-ordered, inverted file indices exploit the MoDisco property in the following way. The traversal of the documents indexed by the inverted-file terms contained in the query is ordered by the magnitude of their 1-D projections with the query's subspace dimensions. This ordering ensures that we first visit the strongly connected documents, those with large dimension weights intersecting the query subspace.

4.2 Nearest-neighbor Search

The computational task of selecting the nearest neighbors of a point applies to a broad class of problems and is the subject of a significant amount of previous work [72]. For data with the MoDisco property, we suggest that nearest neighbors are most efficiently computed using index-based queries, rather than the two other major categories of nearest-neighbor search techniques that are relevant to our context: spatial partitioning strategies and mapping-based techniques. In this section we discuss the related work, and then explain how index-based queries can be expected to out-perform the state of the art. We then present a new algorithm for all-pairs, approximate k-nearest-neighbor search using an impact-ordered inverted file and show how it achieves high accuracy results in less time than competing approaches.

4.2.1 Implication: Nearest-Neighbor Search with an Impact-Ordered Inverted File

We suggest that when processing MoDisco data for nearest-neighbor search, query indices are more efficient than either spatial partitioning or mapping based techniques. We provide an intuitive justification for this claim in this section, backed up with an empirical analysis of the benefits of using query indices in Section 4.2.3.

Each term vector resides in a low-dimensional subspace of term space, and the inverted file maintains a record of the subspace dimensions for each vector. When using the terms of a given term vector as a query into the impact-ordered index, we first encounter those vectors whose projections onto the query's subspaces dimensions are the largest. These vectors are more likely to be in the set of points closest to our query. In this way, the impact-sorted, inverted file already acts like a mapping function on MoDisco data, but without the need to compute any projections or hash functions.

Spatial partitioning strategies and mapping based techniques, in contrast, operate with no *a priori* knowledge of the subspace structure of the data. They must first resolve the relevant data subspaces by either building the spatial-partitioning data structure or computing enough mapping functions to effectively discretize the term space into relevant regions.

It is instructive to note how using index queries for the general case, when the data does not have the MoDisco property, will fail to improve over other nearest-neighborhood search methods. In the general case, there is no guarantee that most points will reside in a low-dimensional subspace. In the case of a dense dataset, single inverted index entries are *full*, containing references to the entire set of points. When inverted file entries are full, they provide no useful method for reducing the search space for neighboring points. Indeed, in the full inverted-index-entry case, the inverted file devolves into an exhaustive search over all the data points. In the case of sparse rows, but *not* sparse columns, we can guarantee that most points reside in a low-dimensional subspace. However, we still cannot guarantee against the existence of a full inverted index entry.

4.2.2 Algorithm: APQ

Our All Pairs Queries (APQ) algorithm constructs an approximate set of k-nearest neighbors without exhaustively computing all the similarities between data points. We have argued that the fastest way to perform nearest neighbor search on MoDisco data is with index queries, where the query is constructed from the terms of the document itself. Thus, to compute the full set of nearest neighbors, we will want to perform an ensemble of queries across all pairs of data points. Using impact-ordered processing, we can then truncate the query computation when we produce the highest scoring, or nearest results.

The APQ algorithm performs an ensemble of queries in an iterative way using data structures similar to the single query case [137]. The algorithm maintains several lists of fundamental data structures. First, the term vector dataset V. The inverted file I, which we store as a list of arrays of document indices, is sorted in order of term weight. The expression $I[i, j]$ references the document whose weight for term i is the jth largest among documents possessing that term. Figure 4.4 presents a simple two-document term dataset and its corresponding inverted file. The third list Q is the set of terms for each point, where each set of terms is organized as a priority queue. Figure 4.5 describes the values stored in each priority queue element: the queue item *priority* used to position the item in the queue, a *term* used for indexing a row in the the inverted file, a *term pos* recording the po-

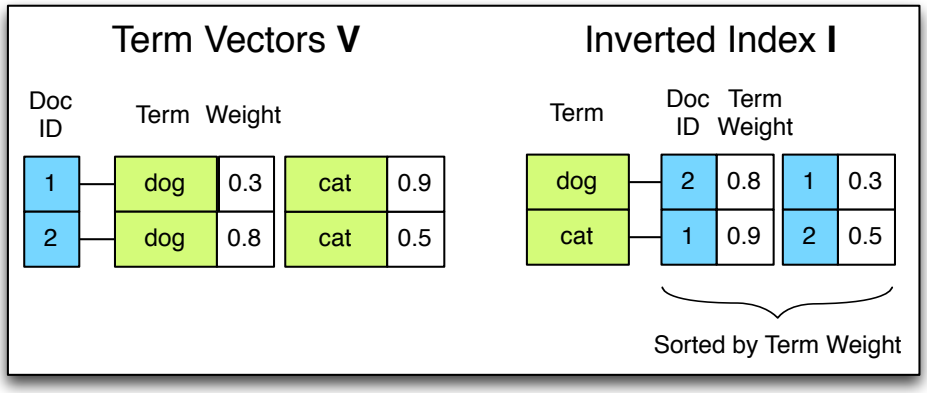


Figure 4.4: A sample term vector dataset and its inverted file. The terms in V become the indices of the rows in I, while the IDs of the documents forming the rows in V become the indices for the weights stored for each term in I. The list of weights for each term in I are stored in descending order of document weight.

sition number of the term in the term vector, and a *counter* used to index a column in the inverted file. The accumulators are a list A of hash sets that map document numbers to accumulated terms. The accumulator $A[i, j]$ stores the accumulated terms that make up the inner product between document i and j up to the current iteration. These sets of accumulators A are the output of the algorithm. We discuss how to interpret A as a distance matrix in section 4.3.1.

The primary difference between our method and the single-query, impact-ordered algorithm is the introduction of the set of priority queues Q. In the single query case, impact ordering is achieved by sorting the set of inverted file term entries using the product of each query term weight and the matching inverted file term weight as a sort key. The priority queue entry for each document Q replicates the same sort ordering by making the priority value for each queue entry the same as the sort key. The priority value for the head queue element is then updated to be the product of the queue entry weight with the next term in the inverted file, thus repositioning the item in the queue. Using the priority queues interleaves the sort ordering and accumulation calculations in a single inner-loop iteration. Figure 4.6 presents the whole APQ algorithm in pseudocode.

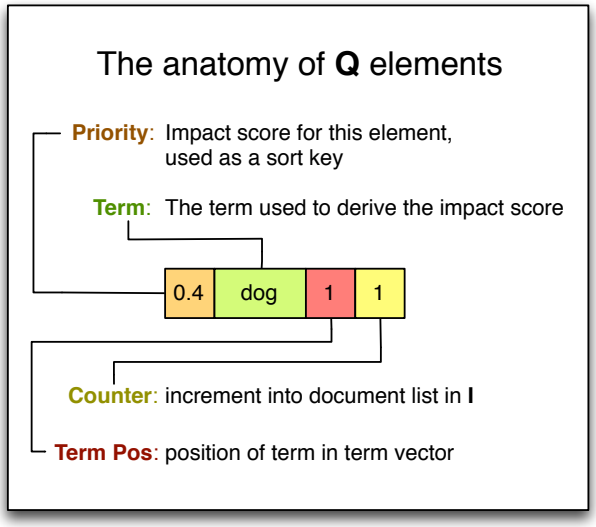


Figure 4.5: Anatomy of Priority Queue elements maintained for each document. The priority queue **Q** is used to sequence the computation of terms in the APQ algorithm in *impact order*.

INPUT
V : Term Vector Database
I : Inverted File
$maxiter$: Maximum number of iterations to run
OUTPUT
A : List of Accumulator Sets
procedure APQ($V, I, maxiter$)
    for $i \leftarrow 1, M$ **do** ▷ Loop over terms
        $I[i].sort()$ ▷ Sort each term index.
    for $i \leftarrow 1, N$ **do** ▷ Loop over documents
        for $j \leftarrow length(V[i,])$ **do** ▷ Loop over terms
            $t \leftarrow V[i,j].term$ ▷ term index
            $a \leftarrow V[i,j].weight$ ▷ document term weight
            $b \leftarrow I[t,1].weight$ ▷ top index term weight
            $qitem \leftarrow (a*b, t, j, 1)$ ▷ Build priority queue item
            $Q[i].insert(qitem)$ ▷ Insert priority queue item
    while $iteration < maxiter$ **do**
        for $i \leftarrow 1, N$ **do** ▷ Loop over documents
            ▷ **Begin A update**
            $qitem \leftarrow Q[i].head$ ▷ Grab the highest priority item
            $t \leftarrow qitem[2]$ ▷ term index
            $c \leftarrow qitem[4]$ ▷ index counter
            $A[i, I[t,c].docid] += qitem[1]$ ▷ accumulate term
            ▷ **Begin Q update**
            $qitem[4] \leftarrow c+1$ ▷ increment index counter
            $j \leftarrow qitem[3]$ ▷ term number
            $qitem[1] \leftarrow I[t,c].weight * V[i,j].weight$ ▷ update priority
            $Q[i].update(qitem)$ ▷ reposition queue item

Figure 4.6: All Pairs Queries algorithm pseudocode.

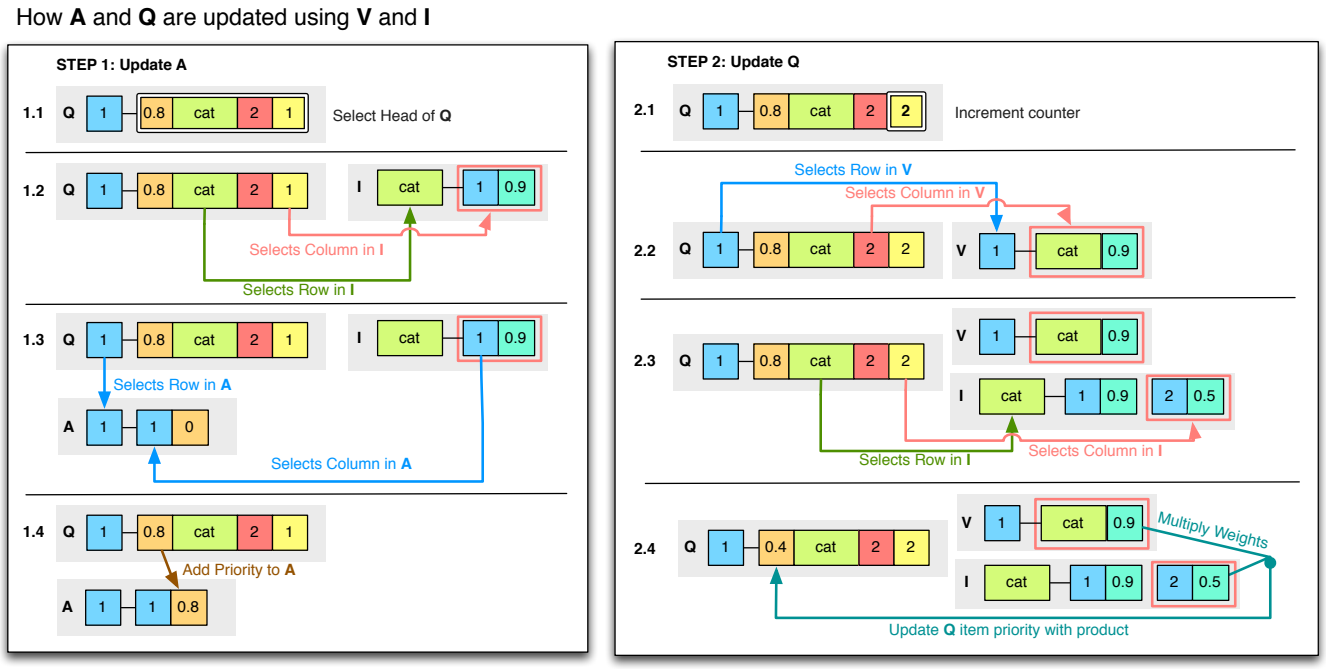


Figure 4.7: Diagram of the accumulator and priority queue update. The accumulator A and the priority queue Q associated with each document are updated using the term-vector dataset V and the inverted file I after each iteration of the APQ algorithm. The left half of the figure diagrams the steps that update A. The right half of the figure diagrams the steps that update Q.

The main loop of the APQ algorithm consists of two steps applied to each document. In the first step, we update the document's accumulator list and in the second, we update its priority queue. Figure 4.7 illustrates these two steps using the example dataset listed in Figure 4.4. The diagram shows the relationship between the data structures in the pseudocode: the term vector database V, the inverted file I, the list of priority queues Q, and the list of accumulator sets A. (Technically, the two-document dataset in the illustration is not MoDisco, but we use this simple example for clarity.)

The update to a document's accumulator list, illustrated in the left box in Figure 4.7, is done in four sub-steps. First, we read the head element from the priority queue. We then use the term and counter members of the queue element to index into the inverted file I and reference the appropriate document-id/term-weight pair. The ID of the document we are currently processing and the ID stored in the referenced pair provide an index into the appropriate entry in the accumulator file A. We then add the priority value of the queue element to the value stored in the accumulator file entry.

After updating the accumulator for a document, we update the priority value of the head queue element. This update is illustrated in the right box in Figure 4.7. First, we increment the counter field of queue element. Then, in the next two steps, entries from V and I are referenced using the queue element's indexing fields. The priority value of the queue element is then set to the product of the weight fields from these referenced entries. After updating the queue item priority, its position in the queue is updated, possibly shifting its location relative to other queue elements.

Note that, before entering the main loop, we must first initially construct the priority queue $Q[i]$ of each document with one queue element for each term. The priority values used by the priority queues are the impact ordering scores. As shown in the right box of Figure 4.7, the impact ordering scores for queue elements are computed by taking the product of the term weight and the corresponding term in the inverted file referenced by the counter of the queue item. At the beginning of the algorithm, the index counters are all 1 and so we reference the first entry in the inverted file list for the indexed term.

The algorithm outer loop may terminate over a variety of possible thresholds. For instance, search engines often simply compute a fixed number of iterations

resulting in a small subset of results for the user to peruse, allowing a user to compute iterations only if interested. This case is presented in the pseudocode in Figure 4.6. In practice, we use a termination threshold based on the change in the largest k accumulators.

4.2.3 APQ Results

We compare our APQ algorithm to state-of-the-art representatives from the spatial-partitioning and mapping classes of nearest-neighbor-search algorithms. For spatial partitioning, we select the Vantage-Point Tree (VPT) algorithm[3] [135]. For mapping algorithms, we select the Locality-Sensitive Hashing (LSH) algorithm[4] [39]. We modified both the VPT and LSH implementations to be able to rapidly store and compute cosine distances of sparse feature vectors.

As a quality measure, we compute the adjusted agreement rate [53] between the true set of nearest neighbors and the nearest neighbors returned by the approximate nearest-neighbor algorithm. Agreement rate measures the average size of the intersection between sets. Adjusted agreement rate subtracts the expected number of intersections due to chance. The formula for adjusted agreement rate is

$$AR_k = \left(\frac{1}{kN} \sum_{i=1}^{N} a_i \right) - \frac{k}{N-1}$$

where N is the number of points, k is the considered number of neighbors, and a_i is the number of *true* k-nearest neighbors appearing in the set of the k-nearest neighbors in the layout.

For APQ, we sample AR_5 measure across a range of compute times. Direct comparison between APQ and the LSH algorithm is complicated by the presence of two parameters: number of hash tables, and the number of mapping functions per table. We regularly sample these two parameters at integer locations over a range between 5 and 80 for hash tables, and 5 and 10 for mapping functions, generating a set of nearest neighbors for each sample and recording the compute time and AR_5 for each sample. We then compute the efficient frontier of these samples with

[3] implementation in C++ provided by van der Maaten.

[4] implementation in C++ as part of the Caltech Large Scale Image Search Toolbox

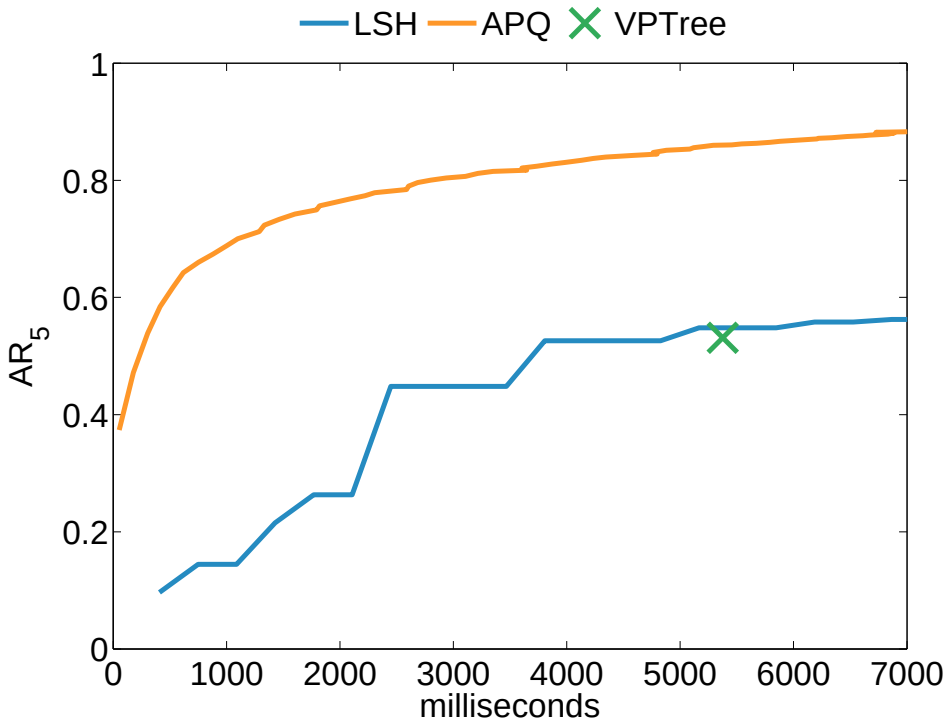


Figure 4.8: Speed vs. Accuracy chart for the APQ 5-nearest-neighbor search algorithm on the `warlogs` dataset.

respect to time and AR_5 to produce a comparison curve. Since VPT is a parameter-free algorithm, we simply ran it once, recording the resulting AR_5 and compute time.

All result timings in this paper are generated on a 13-inch Macbook Pro running OS X 10.8 with a 2.5 GHz Intel Core i5 processor supplied with 8GB memory. Our implementation of APQ and all other implementations in this paper are written in the C programming language.

Figure 4.8 shows the speed-vs-accuracy tradeoff for APQ, LSH and VPT on the task of selecting the 5 nearest neighbors on the `warlogs` dataset. Speed is measured in CPU time, and accuracy is measured as the fraction of correctly labelled nearest neighbors in the 5-nearest-neighbor set. The chart indicates that APQ achieves the same search accuracy in less time than the LSH or Vantage Point Tree techniques. We ran the same analysis with other values of k and other

MoDisco benchmark datasets; the result was identical relative positioning of the algorithms.

4.3 Distance Matrix Computation and Storage

Distance matrices store the distances between points in a data set. In addition to being interesting objects of theoretical study, they often serve as input to practical data analysis algorithms like clustering [1] and dimensionality reduction [118]. Computing and scanning the entire distance matrix is costly due to the O(N^2) unique entries stored in the matrix. In this section we discuss how the MoDisco property implies that we do not need to compute and store this entire database, leading to significant savings in cost. We then demonstrate how to use approximate distance matrices in practice, introducing an algorithm that uses our distance matrix approximation to produce a hierarchical clustering of the input data in less time than using a full inverted file traversal.

4.3.1 Implication: Truncated Distance Matrices

We can greatly reduce the size of storage required for a distance matrix of MoDisco data without suffering from excessive approximation errors. We can accomplish such an approximation by storing distances between term vectors that share high-impact terms and further assuming the remaining distances are 1. This implication is a direct consequence of the distribution of weights among term vectors in MoDisco data where we assume that a vector x_i possesses a small number $X_i(t)$ of high-impact weights, and that the number of vectors $Y_j(t)$ that also possess a high-impact weight for these same dimensions is also small.

An efficient strategy for storing and retrieving these distances is to maintain a hash table for each data point, where distance matrix values are keyed by point index. Using such a strategy, any particular distance d_{ij} can be computed in $O(1)$ by checking if the key j exists in the ith table. If it does exist, the stored distance value is returned; if the key does not exist, it is safe to return 1. We call this efficient distance matrix data structure a **truncated distance matrix**.

Finally, the APQ2DMAT method translates the set of accumulator lists A returned by APQ into a truncated distance matrix. First, we initialize the distance

matrix by constructing an empty hash set for each document. Then, we iterate over all accumulator entries in each accumulator list, adding 1 minus the value of *j*th entry of the *i*th list to the *j*th entry of the *i*th hash set in the distance matrix. To symmetrize the distance matrix, we also insert the same value to the *i*th entry of the *j*th list.

4.3.2 Algorithm: Fast Cluster Tree Calculation with DM2CT

Cluster dendrograms are diagrams that display hierarchical pairwise relationships between clusters in the data. They are visual representations of the cluster tree: a tree whose nodes represent successively higher-density regions of the data, and is computed using hierarchical clustering algorithms. Single-link clustering is a classical hierarchical technique that is isomorphic to the Minimum Spanning Tree (MST) of the complete graph induced by the distance matrix, where edge weights between points are determined by distances between the same points.

We now present the DM2CT algorithm for computing a single-link cluster tree from a truncated distance matrix in $O(N \log N)$ steps. The first step of DM2CT is to convert the dataset into a graph. To do this we first construct a set of graph vertices, one for each document. Then we map the set of distances stored in the matrix into graph edges, where the distance d_{ij} is converted into an undirected edge between vertex i and j with weight d_{ij}. Finally, we perform Kruskal's algorithm in $O(E \log E)$ steps to generate the MST of this graph [68]. The order in which edges are added to the MST is the same order that clusters are joined together in the hierarchy. DM2CT is similar to the Voorhees algorithm for computing exact cluster trees from inverted files [119], which uses Prim's MST algorithm with a full traversal of the inverted file. By using APQ with impact ordering and APQ2DM to generate our distances, we avoid full traversal while still maintaining a good approximation of the distance matrix.

4.3.3 Cluster Tree Results

We compare the speed of DM2CT with the method suggested by Voorhees [119], which performs a full traversal of inverted file structure. We consider the clustering created by this exact method as the ground truth, and compute a Fowlkes-Mallows

index score [35] between each clustering pair to check the quality of our approximation in terms of fidelity to this ground truth. Fowlkes-Mallows scores are a measure of how closely two clusterings correspond for a set number of clusters C. A score close to zero is poor, with random associations between the two clusterings, while a score close to 1 implies identical cluster assignments. We denote the Fowlkes-Mallows index for the clusters produced from cutting the tree at the 500th split as FM_{500}, and for the 1000th split as FM_{1000}.

Our results in Table 4.2 show an order of magnitude improvement in speed using the approximate truncated distance matrix approach over using the exact inverted-file method. The quality is roughly equivalent; we observe high Fowlkes-Mallows scores near 1.0 between the two clusterings, whereas a random clustering would yield a score of 0.0.

4.4 Dimensionality Reduction

In this section we first summarize the extensive related work regarding dimensionality reduction. We then discuss the implications of the MoDisco property for performing dimensionality reduction on term-vector data. Finally, we present an algorithm for scalable dimensionality reduction of term-vector data and measure layout quality relative to competing dimensionality reduction approaches.

4.4.1 Implication: Dimensionality Reduction through Local Attraction and Global Repulsion

The unique geometry of MoDisco data presents several challenges to DR algorithms. The MoDisco property implies that points mostly reside in a low dimensional subspace that is orthogonal to most of the subspaces in which the other data points lie. We would naturally like the local region of a low-dimensional layout of each point to reflect the nearer points with intersecting subspaces. This requirement implies a necessary **local attraction** between a point and its nearest neighbors in term space. We now consider the distant points with orthogonal subspaces. While the cosine distance between two points being orthogonal is 1, the semantic meaning of two documents being orthogonal implies that they are unrelated. Therefore, we argue that it makes more sense for orthogonal points to simply repel each other

Benchmark	N	M	**Distances**			FM_{500} fidelity	FM_{1000} fidelity
			DM2CT time (ms)	**Voorhees** time (ms)	**speedup**		
`metacombine`	28K	28K	2597	14323	6	0.94	0.96
`cables`	7K	66K	2155	114300	53	0.96	0.92
`warlogs`	3K	4K	531	4608	9	0.92	0.7
`conspiracy`	1K	33K	715	5839	8	0.74	N/A
Benchmark	N	M	**Clustering**			FM_{500} fidelity	FM_{1000} fidelity
			DM2CT time (ms)	**Voorhees** time (ms)	**speedup**		
`metacombine`	28K	28K	532	175036	329	0.94	0.96
`cables`	7K	66K	67	104308	1557	0.96	0.92
`warlogs`	3K	4K	34	5745	169	0.92	0.7
`conspiracy`	1K	33K	7	437	62	0.74	N/A

Table 4.2: Comparison of hierarchical clustering timing and accuracy using the truncated matrix with our approximate DM2CT algorithm vs. using the inverted file with exact Voorhees method. We divide runtimes of the two methods into the time it takes to compute the distances and the time it takes to compute the cluster tree. The fourth and fifth columns give timings in milliseconds. Column six gives the speedup achieved using DM2CT over the Voorhees method. The last two columns FM_{500} and FM_{1000} denote the Fowlkes-Mallows indices for clusters at the 500th and 1000th split in the tree respectively; an index score close to 1.0 indicates high fidelity between the two clusterings. The DM2CT method achieves orders of magnitude speed improvement while maintaining high fidelity with the exact ground-truth clustering.

than to attempt to fit precisely to a unit distance apart. Since most of the points in a MoDisco dataset are unrelated, we need a DR method that can apply a **global repulsion** between all points.

In the remainder of this section, we discuss how the MoDisco property and the need for local attraction and global repulsion negatively affects the first three families discussed in Section 2.1 but can be accommodated with the probability-based family.

Candidate Property: Orthogonal Projections

Orthogonal projections are an ineffective solution for reducing the dimensions of MoDisco data because the majority of the orthogonal variance in MoDisco data is captured by the disconnected relationships in the data. Assuming the data is clustered in G clusters, the average distance between these clusters will be 1.0, creating a G-dimensional simplex. The first G principal components will primarily express these unit-length, inter-cluster distance relationships, often without resolving any of the intra-cluster distance relationships that may reside in a subset of orthogonal dimensions. Because G is often larger than 2 or 3, PCA by itself is an ineffective tool for producing low-dimensional visualizations of MoDisco data.

Candidate Property: Global Distances

There is surprising variability to the number of MDS techniques [36], but our arguments here hold for any particular MDS algorithm. The problem that MDS methods have with MoDisco data lies with optimizing the layout to fit the preponderance of unit length distances. MDS methods optimize a function called *stress*, which sums the residuals between the high and low dimensional distance matrices.

Buja and Swayne noted the problem of *indifferentiation* [16] when using MDS for datasets where distances are clustered around a positive constant, with the extreme case being a perfect simplex: the high-dimensional analog of a tetrahedron, where every object is equally distant from every other object. In this case, applying MDS yields low-dimensional layouts with a characteristic shape approximating a uniform ball: in 2D, it is a circular disk with a sharp edge and a low density in the center. They caution data analysts to avoid misinterpreting this artifact as mean-

ingful dataset structure. Figure 4.3 shows the similarity of MoDisco data to their simplex example; both the simplex and the MoDisco distances are largely skewed toward a single value. To our knowledge, their observation is the only close connection between the concept of MoDisco data and a Visual Analytics use case; unfortunately their thoughtful analysis has not percolated into the visualization literature.

Continuing with this line of analysis, we argue that MDS algorithms spend most of their time on nearly useless computations when applied to MoDisco data. Because no simplex can fit without distortion into a low-dimensional projection, the unit-length distances make up the majority of the error of the objective function. Unfortunately, these disconnected unit-length distances dominate the computation in a way that is inversely related to their importance; the important local distances to their nearest neighbors are not given enough weight.

One strategy to improve MDS maps is to modifying the objective function using polynomial re-weighting of the terms inversely proportional to distance, as in Sammon mapping [97]. However, such schemes do not go far enough; they still retain the need to fit all the distances of unit length in the objective function. Our results in Section 4.4.3 show the negative impact that fitting the global distances has on layout quality with MoDisco data.

Candidate Property: Manifold Distances

After observing the deleterious effects of fitting global distances, the notion of stitching together the local term vectors into a useful layout may seem attractive. However, in practice, manifold techniques struggle with building an effective manifold model over MoDisco data. Their connectivity assumptions often do not hold because the sampling of the different subspaces is very sparse, often leading to significant distortions. This phenomenon is well documented by van der Maaten [118], who reports that the performance of manifold methods are consistently thwarted by noisy, real-world examples.

Candidate Property: Probabilities

Probability-based methods like SNE [50] and t-SNE [117] can accommodate both local attraction and global repulsion. To achieve local attraction, we fit a Gaussian probability distribution only over the nearest-neighbor set of each input point. To achieve global repulsion, we can simply assign a zero probability to points that we consider to be disconnected from each other. Unlike the previously described DR techniques, probability methods permit significant flexibility in the relative placement and handling of zero-probability points. Using a spring-force metaphor, zero-probability points are attached to a point with an infinite-length spring that rapidly diminishes in spring force as a function of distance. In contrast, distance-based methods such as MDS always assign a fixed-length spring between points that will invariably contract to some finite length, producing an attractive force between two points that have no meaningful relationship. The crucial property of this family of methods is the ability to assign zero weight to points in a way that is consistent with the mathematical framework; developing different approaches that do not use probabilities to do so would be interesting future work.

In Section 4.4.2 we detail the results of a modified implementation of the BH-SNE algorithm [116]. We choose the BH-SNE algorithm over other probability-based DR methods due to empirical evidence of good cluster separation, as well as its proven $O(N \log N)$ iteration cost. Because the NE algorithm [134] is virtually identical to BH-SNE, the results below remain the same if the NE algorithm is substituted for BH-SNE.

4.4.2 Algorithm: MD-SNE

Our discussion of Implication 3 notes that the t-SNE algorithm minimizes the divergence between the probabilities of the layout and the data [117]. We now describe a scalable, efficient way to construct high-quality probability-based layout of document datasets. We first discuss the implementation details of the BH-SNE algorithm and its shortcomings, and then describe our own modification to the algorithm to adapt it for MoDisco data.

BH-SNE

BH-SNE [116] is a modification of the t-SNE algorithm [117], which is itself a modification of the Stochastic Neighborhood Embedding, or SNE, algorithm [50]. SNE fits Gaussian probability distributions for each point over the other points in the dataset with variances determined by a single user-selected parameter called perplexity. It then positions points in low-dimensional space in such a way that minimizes the Kullback-Leibler divergence (a measure of the difference between probability distributions) between the high and low dimensional Gaussian distributions. The t-SNE algorithm improves the cluster separation of the low-dimensional points by using a Student's t distribution for the low-dimensional probability model. The mismatch between the tails in the Gaussian and Student's t distributions effectively adds more space between dense regions in the layout, reducing the visual crowding problem. Both SNE and tSNE perform dense operations on the distance matrix and therefore do not scale well for datasets beyond roughly 10,000 points.

BH-SNE is a technique for approximating t-SNE efficiently on larger datasets. First, BH-SNE approximates the probability distribution for each point by fitting the high-dimensional probability distribution to only the k-nearest neighbors of any given point, and assigning zero probability to the remaining points. In the published method, nearest neighbor search is performed by first reducing the dimensionality with PCA and then using Vantage Point Trees [135] in this reduced space.

t-SNE optimizes its objective function using a momentum-weighted gradient descent, requiring $O(N^2)$ dense-matrix computations. For BH-SNE to perform this same optimization efficiently, it re-expresses the gradient of its objective cost function in two parts: an attractive part and a repulsive part. The attractive part of the gradient is calculated in $O(N)$ time using the set of nearest neighbors, valid since the zero-probability points exert no attraction. The repulsive part of the gradient, which involves all pairs of points, is computed in $O(N \log N)$ time using a Barnes-Hut quadtree approximation. The Barnes-Hut quadtree discretizes space in such a way that the algorithm can approximate repulsive forces with an accuracy dependent on the distance [5]. This approximation is valid because, in the case of t-SNE, the magnitude of the repulsive force often decays as a function of distance.

MD-SNE

Applying BH-SNE directly to MoDisco data is straightforward. The primary shortcoming of the BH-SNE procedure when applied to MoDisco data is its ineffective nearest-neighbor strategy: using PCA to reduce to 50 dimensions followed by Vantage Point Tree (VPT) search. While this method succeeds for small MoDisco datasets, our results in Section 4.4.3 show that as the size of the term spaces increase, the PCA-VPT method breaks down. Instead, we can use our APQ algorithm to generate a truncated distance matrix of k distances per row, where k is the number of nearest neighbors requested by BH-SNE. Then, we simply substitute the indices and distances of the truncated distance matrix in place of the nearest neighbors set computed using their strategy. The BH-SNE algorithm then proceeds normally. Unlike the PCA-VPT method, the quality of the APQ method is independent of the size of the input on MoDisco data.

4.4.3 MD-SNE Results

We compare with representative algorithms from different classes of dimensionality reduction. For linear projection methods, we select Principal Component Analysis, with eigenvectors computed using a fast SVD algorithm [47]. For distance-based methods, we select the hierarchical, stochastic Glimmer MDS algorithm [57]. Finally, we compare MD-SNE to results produced by BH-SNE.

The speeds between the different DR methods are commensurate. The MD-SNE algorithm produced a layout of our largest distance matrix, `metacombine`, in approximately 5 minutes. Because of its $O(N \log N)$ complexity and fixed-iteration optimization, it is not unreasonable to expect MD-SNE to handle million-scale datasets with hour-length processing times.

We claim that probability maps are better suited to produce low-dimensional visualizations of MoDisco data over competing techniques, and that our MD-SNE algorithm is better suited to MoDisco data than standard BH-SNE. Our argument for using probability-based methods over other dimensionality reduction techniques are based on quality, not speed. We validate this claim of quality improvement both quantitatively and qualitatively.

Neighborhood Agreement

We first present a quantitative validation, where we compare a single quality measure across different layouts of the same dataset. Many quantitative evaluations of DR algorithms use the final Normalized Stress measure [57, 63, 83] when comparing against other algorithms. The Stress measure, however, incorporates all pairs of distances in its calculation, while we are specifically attempting to remove disconnected distances from the equation. Furthermore, *metric-oriented* objectives like Stress are concerned with measuring precise distances, while Probability-based techniques subject these distances to a nonlinear transformation. We thus compare the rank ordering discrepancy between points using different techniques, rather than the precise distance discrepancy.

As with the nearest-neighbor search discussed in Section 4.2.3, we select *adjusted agreement rate* as a suitable, rank-oriented measure given that the target task is local exploration. Here, we measure the agreement of the local neighborhood of points in high-dimensional space and in the layout for different neighborhood sizes. Instead of measuring AR_k for a single k, we vary k from 1 to 100 for each layout method, on each dataset.

The results are displayed in Figure 4.9. Both PCA and MDS report lower agreement rates on each dataset, while MD-SNE and BH-SNE show similar behavior on all benchmarks except for `metacombine`. For this dataset, BH-SNE shows worse agreement rates for larger neighborhoods. We hypothesize that this result is due to the `metacombine` dataset having a larger feature space than the other benchmarks; it is an order of magnitude larger than the others. The BH-SNE nearest-neighbor search technique of reducing to 50 dimensions with PCA before using Vantage Point Trees is more likely to fail to adequately separate points in such a large feature space. In contrast, MD-SNE exhibits the same pattern of neighborhood-agreement behavior as other MoDisco datasets.

The values for agreement rate in Figure 4.9 are much lower than those in Figure 4.8 on the same dataset. We suspect this discrepancy arises from the challenge of combining the different point neighborhoods into a single planar region. Despite the difference in magnitude, the ordering of the different neighborhood-agreement curves corresponds to the ordering in visual quality between the different tech-

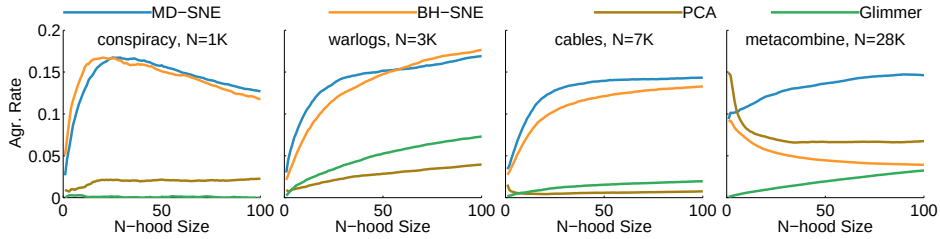


Figure 4.9: Adjusted agreement rates AR_k among dimensionality reduction techniques on our benchmark datasets over a range of neighborhood sizes k. Higher numbers indicate higher agreement between the layout neighborhood and the true neighborhood. MD-SNE and BH-SNE exhibit roughly equivalent performance on smaller MoDisco datasets, but BH-SNE breaks down on the benchmark with the larger feature space while MD-SNE maintains performance.

niques.

Visual Quality

We now present a qualitative validation based on discussion of result images. Figure 4.10 presents layouts of the four MoDisco datasets for each of the four dimensionality reduction techniques. The PCA layouts, as expected, have a difficult time separating points using linear projections, and clusters tend to be expressed in filament-like structures projected on top of each other. The Glimmer MDS algorithm applied to MoDisco data produces noisy, cloud-like visualizations with large areas of uniform density, and fails to produce much visual separation between dense regions.

On the smaller first three datasets, BH-SNE and MD-SNE both produce acceptable layouts, and we see a tradeoff between cluster separation and local fidelity. On the `cables` and `warlogs` dataset, BH-SNE produces excellent cluster separation, but within the local cluster region MD-SNE layouts show better neighborhood agreement. However, with the larger `metacombine` result, there is a clear quality improvement for MD-SNE over BH-SNE: the BH-SNE nearest neighbor strategy fails to achieve the same cluster cohesion as MD-SNE approach.

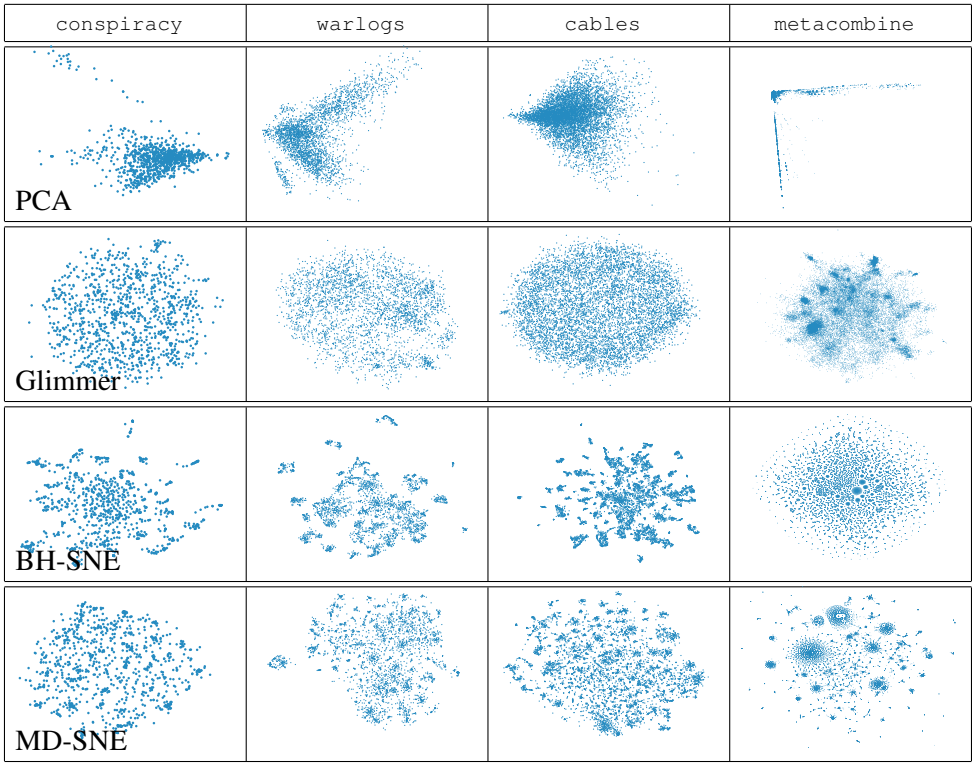


Figure 4.10: 2D layouts of the four MoDisco benchmark datasets across the four tested dimensionality reduction algorithms. PCA yields too much overplotting and Glimmer fails to provide clear cluster separation. BH-SNE and MD-SNE both produce acceptable layouts for the first three smaller datasets, but BH-SNE breaks down with the large `metacombine` dataset.

Chapter 5

Overview Prototype

Chapter 3 presented a software system designed with the goal of providing guidance to non-expert users and Chapter 4 presented a set of algorithm guidelines with the goal of efficiently processing text data for visual analysis. In this chapter, we combine these two goals to present a software system, the Overview Prototype, designed to provide guidance to journalists who are analyzing large text datasets. In the course of designing this tool, we worked in close collaboration with a journalist from the Associated Press who was specifically interested in helping other journalists to understand a large set of documents of mostly unknown content, when indices, summaries, and other knowledge organization aids are not available.

We define the underlying task of our target users as *document set exploration*: the computer-assisted human construction of a categorization that is instance-specific, that is, tailored to a specific use of a specific document set. We assume that full text search of the dataset is available to users, but not completely helpful because they do not know precisely what they are looking for. Our target users are not strictly confined to journalism, and may also arise in fields such as business, law and intelligence.

To process and categorize our document set, we must encode documents in some way that preserves semantics. The *vector space representation* of documents described in Chapter 4 yields very high dimensional spaces with thousands or tens of thousands of dimensions, corresponding to the natural language vocabulary of the document set. Many automatic clustering algorithms that carry out computa-

tions in these spaces have been proposed [10] as ways to categorize large document collections. The output of these algorithms is a list of the documents in each cluster, which does not necessarily provide a sense of the overall semantic structure of the document set. Nor is it obvious whether any particular clustering, out of the combinatorially huge number of partitions of objects into disjoint sets, captures the application-specific semantics of interest. For this reason, many sensemaking systems attempt to directly visualize the high-dimensional cluster structure of the documents through low-dimensional layouts created with dimensionality reduction (DR) techniques [127] [23]. Our first attempt at such a system involved a single interactive layout of a document dataset using a modification of the Glimmer algorithm.

However, we found that users of DR for document set visualization, including our collaborator, have the persistent unease that there is often but not always structure in their datasets that is not revealed; that is, that they see false negatives in many cases but true negatives in others. In the case of document sets, structure most often means clusterings of related documents. But due to the extremely high-dimensional nature of the underlying data, the dimensionally reduced plot can become crowded, with local densities of documents resulting from chance (a false positive). In a similar way, true densities of related documents may be "interrupted" by spurious, unrelated documents (a false negative). The tool developed in this chapter, the Overview prototype, is designed to mitigate these two related problems by augmenting and complementing a dimensionally reduced view of the data with additional information in the form of a hierarchical decomposition of the data and an infrastructure for manual annotation.

The remainder of this chapter is organized as follows: we first describe the Overview prototype application that combines a hierarchical clustering dendrogram and an MDS view to support text analysts in exploring and annotating document collections through tagging clusters. We then justify our design by discussing how the different prototype components work to support our target analysis task. Next, we show the prototype at work in two separate data analysis cases; the first case demonstrates a hypothetical use of the tool, and the second case describes the results of actually giving the tool to a real-world data analyst. Finally, we describe the results of the tool's public deployment to users outside of the experimental

setting.

5.1 Overview Protoype Description

The Overview prototype, shown with labeled components in Figure 5.1, is a proof-of-concept application to address computer-assisted human classification through tagging items. It is based on interactively combining clustering, tagging, and dimensionality reduction. In this section we describe the different data views and how to interact with the prototype.

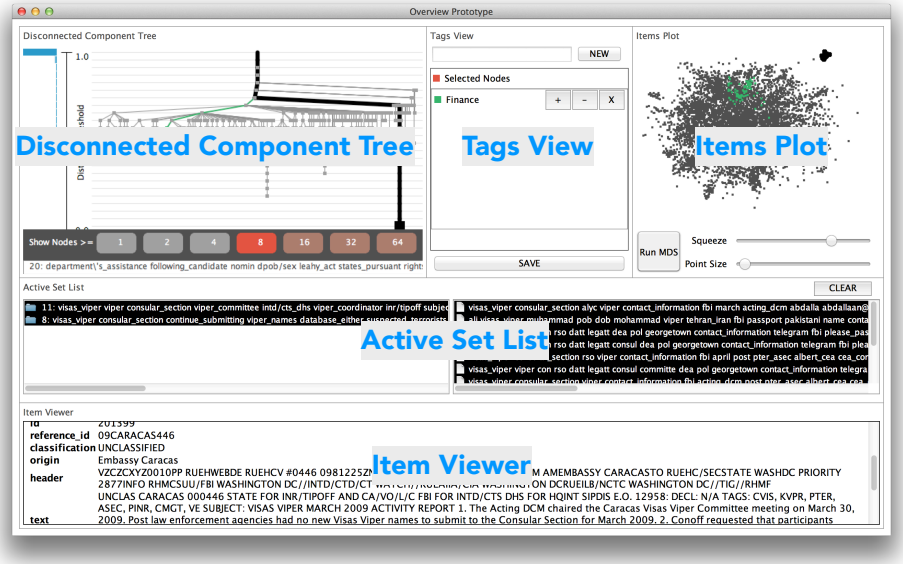


Figure 5.1: The components of the Overview prototype. The Overview prototype consists of five main components, indicated in this figure with blue labels. The `Disconnected Component Tree` displays an interactive dendrogram, the `Tags View` displays and manages user-created tags, the `Items Plot` displays a scatterplot of the data, the `Active Set List` displays the tags and clusters contained in the current selection, and the `Item Viewer` presents the text of the highlighted item in the selection.

In the top half of the application are three different views of the data. The *left* cluster view features an interactive representation of the hierarchical clustering

dendrogram (hereafter called the `Disconnected Component Tree`, or DiscoTree), and the *right* DR view, called the `Items Plot`, shows a 2D scatterplot of a low-dimensional embedding created with the Glimmer MDS technique [57]. The `Tags View` in the *middle* allows tags to be created or highlighted. The views are all linked to support cross-filtering [124].

Below these views is the `Active Set List`. The active set is the set of documents highlighted by the user and their associated nodes. The active set is constructed through a selection interaction, such as clicking a node or lassoing a set of items, within a particular view. The `Active Set List` shows DiscoTree nodes on the left and items on the right, where the labels are the top terms within a document, or of all documents within the cluster. At the very bottom is `Item Viewer`, which displays the underlying text of a specific item selected in the right items list of the `Active Set List`.

A user interacts with the prototype by selecting nodes in the DiscoTree, by highlighting regions of the `Items Plot`, or by clicking a Tag in the `Tags View`. This brings all items and nodes that match the selection criteria into the `Active Set List`. The user can then peruse the items that make up the selection in the `Item Viewer`. Finally, having collected related items into an active set, the user can apply a tag to those items in the `Tags View`. Section 5.3 provides more details on how to use the different components of the prototype by walking through an analysis task on a real-world dataset. As a supplement, our collaborator produced an online tutorial video explaining the different components of the software[1].

5.2 Clustering, Tagging, and DR for Sensemaking

We now justify the design of the Overview prototype by articulating the document set exploration task more precisely as supporting text analysts in building an application-specific hierarchical categorization schema through tagging, starting from the scaffolding of a schema automatically created through hierarchical clustering. Dimensionality reduction fits into this workflow both to enable direct visualization of document set structure, where possible, and as a way to evaluate

[1] http://overview.ap.org/blog/2012/03/video-document-mining-with-the-overview-prototype/ Last accessed July 17th, 2013

the relationship between the distance metric used in algorithmic clustering and the semantic content of the dataset.

5.2.1 Why Clustering?

Clusters of points in high-dimensional data sets are interesting because they often have meaning; that is, cluster structure frequently represents semantics of interest to the user. This statement posits a strong connection between a mathematical property and an abstract, high-level notion of human knowledge.

The idea of representing document collections as point sets in high-dimensional space began with the work of Luhn in the 1950s [74] and was developed into the vector space model by Salton et al. [96], originally designed for information retrieval tasks. Section 4.1 contains a thorough description of the vector space model.

Information retrieval and dimensionality reduction applications rely on a similarity or distance function, defined over every pair of documents, which gives rise to a metric space and associated topology. Spatially compact clusters of document vectors in cosine distance space were recognized by early information retrieval researchers as semantically interesting structures, giving rise to the *cluster hypothesis* [61], a modern version of which is articulated as "documents in the same cluster behave similarly with respect to relevance to information needs" [75]. The cluster hypothesis is widely assumed and has been shown to hold in the case of web-scale information retrieval [26].

Sensemaking differs from information retrieval in that the user does not know beforehand what type of information is sought [94]. However, because the documents within a cluster are conceptually similar, representing a document corpus by its clusters may be a useful form of information reduction. The intuition is that if the text analyst has read a few documents in a cluster, they can assume that the rest will contain a similar type of information.

5.2.2 Why Tagging?

A cluster is, in the end, just a set of documents. While there is evidence that machine-extracted clusters capture interesting semantics, that does not help the user to understand what any given cluster means, much less a tree which may

include hundreds of clusters and sub-clusters. Cluster labeling is the crucial next step in sensemaking.

There have been many more or less sophisticated attempts at automatic cluster labeling, ranging from displaying the most frequently occurring words to attempting to extract a single key sentence from a text corpus [136]. A related problem is the naming of topics extracted by topic modeling algorithms such as Latent Dirichelet Analysis (LDA) [11]. Each *topic* found by such an algorithm is a probability distribution over words, sometimes visualized with a word cloud as shown by the TextFlow tool of Cui et al. [27]. Yet a word cloud is not a substitute for a good topic name, as researchers working with LDA-based methods tacitly acknowledge when they compose short, human-generated labels to refer to the semantics of extracted word distributions.

A deeper problem is the suitability of the classification schema implied by the distance metric. In what sense are two documents really "similar"? In practice similarity depends on the context of the analysis. For example, should the documents be grouped by location, specific event, type of incident, or actors involved? There is no reason to assume that the particular encoding and distance metric used to generate clusters necessarily partitions the documents in the most semantically useful way, especially given that, for the sensemaking task, the user may not know beforehand what ways are going to be interesting.

Grimmer and King approach this problem by visualizing the space of all possible clusterings, populated by executing a variety of clustering algorithms on the same data [43]. They are able to directly explore some of the different semantically interesting categorizations on the same set of documents.

In principle, the sensemaking loop should include adjustments to the vector encoding and distance metrics so as to explore different categorization schemas, but very little is known about how to automatically make these adjustments. Instead, we consider the automatically-generated clusters as starting points for human classification. We argue for a tagging system that allows the user to summarize and annotate the content of a cluster, by applying a label to some or all of its documents. These tags allow the construction of a manual classification scheme that follows or cuts across the cluster structure as desired, and has greater or lesser resolution around particular concepts and subtrees.

5.2.3 Why Dimensionality Reduction?

Dimensionality reduction methods such as MDS also use distance metrics, just as k-means and other clustering methods do, to map items to a lower dimensional space. The promise of MDS and other DR methods is visual confirmation of the existence and size of clusters. MDS as a visualization technique is often used to verify a proposed clustering by coloring the points according to the clusters and checking if the spatial proximity relationships in the low dimensional view match up the coloring, or to visually check for cluster structure in unclassified data by noticing if there are any visually separated clusters in the view. The visual display of dimensionality reduction results allows the user to cross-check whether the relationships between the visible clusters that arise from the distance metric match their mental model of semantic content.

5.2.4 Why All Three?

In this framework, the sensemaking task is the construction of a set of tags which capture the concepts of interest — perhaps newly discovered — in the document collection. The tags, presented in the tool in the `Tags View`, act as an annotation layer to get human semantic understanding into the exploration loop. An automatically created clustering, which can be navigated and examined in the `DiscoTree View`, serves to accelerate the process of constructing meaningful annotations. Finally, dimensionality reduction, as presented in the `Items Plot`, provides a parallax view of the cluster structures.

The Overview prototype then combines these three capabilities as three very different linked views of the document dataset. Using the views, an Overview user cycles between examining clusters, writing annotations, and examining local neighborhoods and outliers until the process converges to a satisfactory categorization of the document dataset.

5.3 Overview Prototype Results

We show results of using the Overview Prototype with two real-world journalism datasets from WikiLeaks, `Warlogs` and `Cables`. In both, documents were encoded as a vector using the TF-IDF term weighting scheme [96] applied to all

vocabulary words plus automatically detected common two-word phrases, or bi-grams.

5.3.1 Afghan War Logs

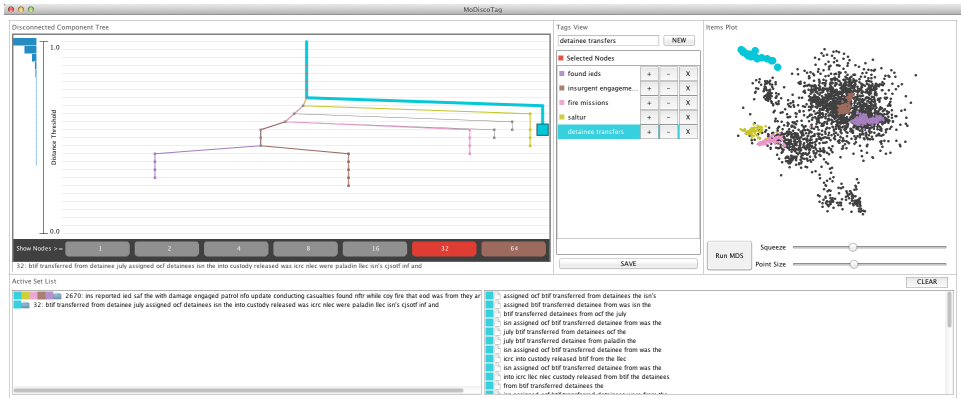


Figure 5.2: The Overview Prototype loaded with the `Warlogs` dataset and having five nodes of the DiscoTree tagged. The colors reveal that the documents they contain fall into local neighborhoods in the MDS `Items Plot`, but most of this structure cannot be seen from proximity relationships alone because there is no visible separation from the other data points.

The `Warlogs` dataset is the subset of the WikiLeaks Afghan Warlogs dataset from July 2009. It contains 3077 points and 4286 dimensions, where a single point corresponds to a document from the dataset. These documents are military after-action reports with an extremely terse format and specialized jargon, so they are not trivial for non-experts to read.

Figure 5.2 shows the results. We began with the most compact pruning level where only nodes of 64 or more items are visible in the DiscoTree, revealing seven main clusters. Exploring those with the combination of quickly reading labels in the `Active Set List` and using the `Item Viewer` to read the full text of documents led us to quickly confirm five of these as semantically meaningful categories and tag them with the following names and colors: *found ieds* (purple), *insurgent engagements* (brown), *fire missions* (pink), missions containing a *saltur* report (size, activity, location, time, unit, result) following an enemy contact (gold),

and *detainee transfers* (teal).

We can see that these groups do form neighborhoods in the MDS `Items Plot`, but would be difficult to identify without the coloring because the regions are not visually separated from the rest of the points by regions of low density. The exception is the well separated *detainee transfers*.

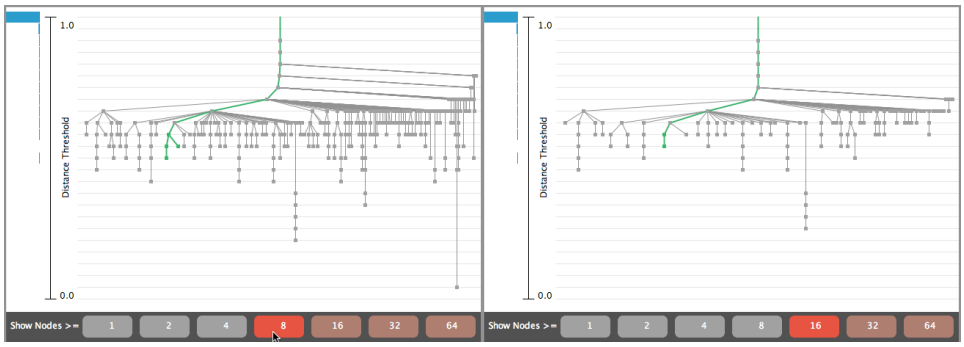


Figure 5.3: The DiscoTree control at different prune levels. Using the prune controll at the bottom of the `DiscoTree View` permits the user to see the cluster tree at several levels of abstraction. Here we show the view of the `Cables` dataset at prune level 8 (left) and 16 (right).

For clarity, the `DiscoTree View` is interactively prunable so that only nodes past a particular size are visible, using the logarithmic `Show Nodes >=` buttons along the bottom. Figure 5.3 shows the `DiscoTree View` of the `Cables` dataset at varying prune levels. Each node in the DiscoTree contains many document items, and due to the hierarchical clustering the same item will appear in every node along a path from the singleton leaf node at the bottom of the tree up to the root. When an item is selected through one of the other views, this entire branch is highlighted through an edge color.

The user can use the `Tags View` to create and assign an arbitrary set of named and colored tags. Clicking on a tag's name selects all items which have been assigned to that tag. While the user interface supports the useful shortcut of tagging all items in a node, in general tags may be arbitrarily distributed across items. Nodes are assigned a tag's color when all items within that node contain that tag. Due to the hierarchical nature of the tree, once a node is colored in, all of its child nodes will also be filled in.

Each item in the MDS `Items Plot` is a single document, represented by a point. An individual item can have many tags attached to it, and a node has many items and thus many tags as well. We do not attempt to show all tags at once with any sort of glyph, since items and nodes cover small regions of the screen. Instead, the last selection color takes priority over the others.

5.3.2 Caracas Cables

The `Cables` dataset comes from a different WikiLeaks source, the diplomatic cables. We analyzed the subset consisting of cables sent to or from the US Embassy in Caracas, Venezuela, or containing the word "Caracas". It has 6849 points and 65,777 dimensions.

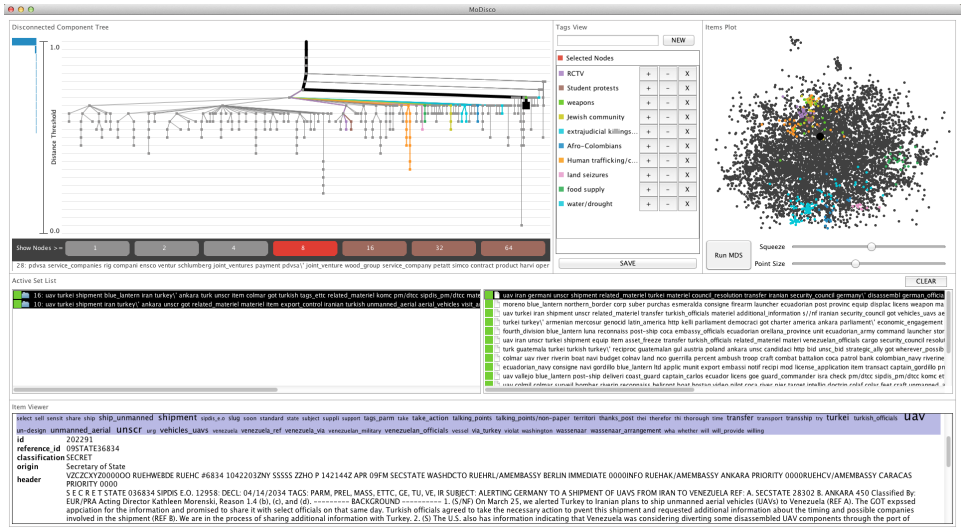


Figure 5.4: Cable alleging Iranian drone plans. The journalist found a cable alleging Iranian plans to ship unmanned aerial vehicles to Venezuela.

We provided this dataset and the Overview prototype to the Associated Press Caracas bureau chief, who created several dozen tags over a session of a few hours. These tags, listed in Table 5.1, cross-cut the data in different ways including geography, politics, and events; the supplemental video[2] shows the prototype with each of these three tags sets. Figure 5.4 shows a different subset of ten interesting

[2] http://www.cs.ubc.ca/labs/imager/video/2012/modiscotag.mov Last accessed July 17th, 2013

Tag Name	# Tagged	Tag Name	# Tagged
Colombia/rebels	570	RCTV	89
Ven Opposition	424	Student protests	64
Oil industry	428	Suriname/Dutch territories	287
Chavez politics	383	Paraguay	95
Finance	382	Human trafficking/child labor	236
extradition	678	land seizures	43
vatican	283	weapons	79
China	64	food supply	93
Caribbean politics	923	Russia	117
Peru	1405	Brazil	180
Portugal	109	Guyana	143
Banks	59	Jewish community	108
Elections	138	extrajudicial killings/human rights	374
Argentina	72	Afro-Colombians	94
Cuba	66	health care	39
ALBA countries	213	education	39
Nicaragua	357	nationalizations	64
Ecuador	728	Spain	86
Airlines	99	El Salvador	113
Chile	104	Israel	39
Honduras	234	Trinidad	39
Jamaica	99	water/drought	44
Colombia politics	207	Belarus	39

Table 5.1: List of manually constructed document tags applied to the `Cables` dataset.

tags. Tags are applied to documents, not nodes, because the automatically generated tree does not necessarily categorize the documents in a way that is meaningful to the user. For example, several different branches contain documents concerning *Ecuador* in Figure 5.5. Figure 5.6 shows hierarchical cluster structure, where the parent-child relationships between the branch tagged with *Finance* and its children about banking and oil are also visible as spatial nesting in the MDS view.

The journalist found several topics that he had not previously found through unassisted inspection of the dataset, including arms shipments to Ecuador shown in Figure 5.7. He noted that the application allowed him to quickly spot subject

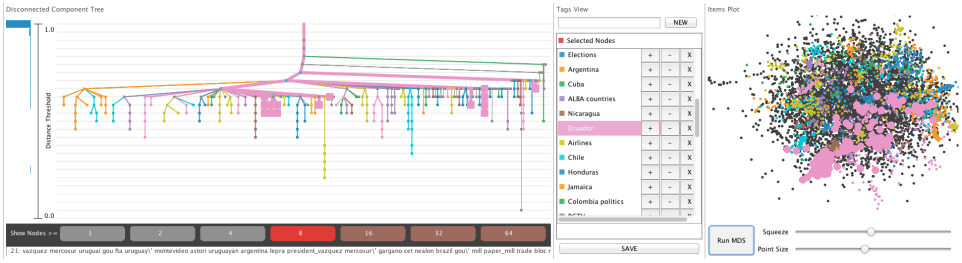
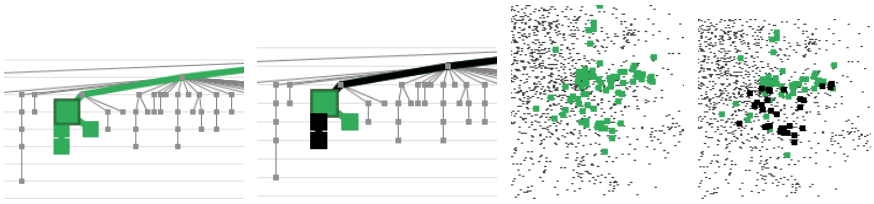


Figure 5.5: The `Cables` dataset, with the full set of categories listed in Table 5.1 from the AP Caracas Bureau Chief.



(a) (b) (c) (d)

Figure 5.6: Hierarchical structure in the `Disconnected Component Tree` and `Items Plot`. The branch tagged with *Finance* has children concerning banking and oil. `DiscoTree View` detail when finance tag selected (a) and one child node selected (b); `Item View` detail for finance alone (c) and child (d).

areas that could be of greater news interest, such as information on Colombian rebels. The tool also helped him find several interesting individual documents, for example claims that Chavez was giving millions to a particular Jamaican politician's election campaign shown in Figure 5.8, and the cable alleging Iranian plans to ship unmanned aerial vehicles to Venezuela shown in Figure 5.4.

5.4 Overview Deployment

The Overview prototype was deployed on a website maintained by the Associated Press[3]. The deployment was evangelized by our collaborator, Jonathan Stray, who promoted the tool through social media and journalism conferences, and built a

[3] http://overview.ap.org Last accessed July 17th, 2013

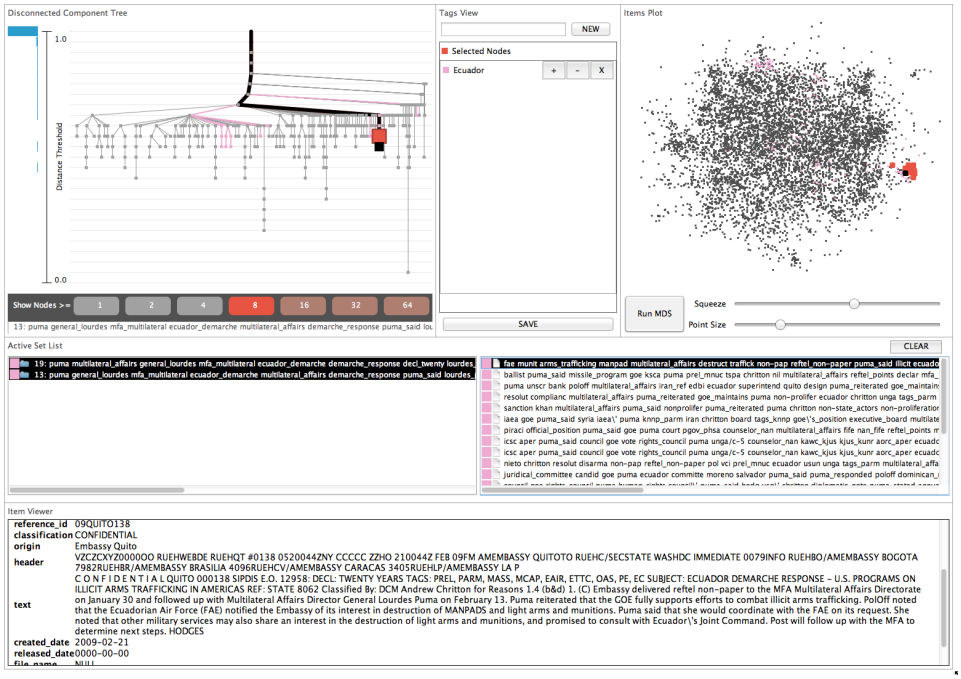


Figure 5.7: Cable with description of alleged arms trafficking. The journalist found cables with a description of such trafficking in Ecuador.

comprehensive tutorial video of the tool on the website[4]. The prototype was then converted to a public, web-based analysis tool[5], complete with user accounts for saving document analyses, and the ability to import data from DocumentCloud, a public document repository for warehousing primary sources. In its various incarnations, Overview has been used by working journalists to assist in published news stories. Here, we briefly present two such use-cases as further validation of our design.

The first use-case is that of reporter Jarrel Wade, working for the Tulsa World newspaper [120]. He used Overview to analyze a collection of emails from the Tulsa Police Department revealing problems with the in-car computer system purchased by the department. In his detailed user experience writeup[6], Wade describes

[4] http://overview.ap.org/blog/2012/03/video-document-mining-with-the-overview-prototype/ Last accessed July 17th, 2013

[5] overviewproject.org Last accessed July 17th, 2013

[6] http://overview.ap.org/blog/2012/09/how-i-used-overview-to-report-on-8000-police-department-emails/

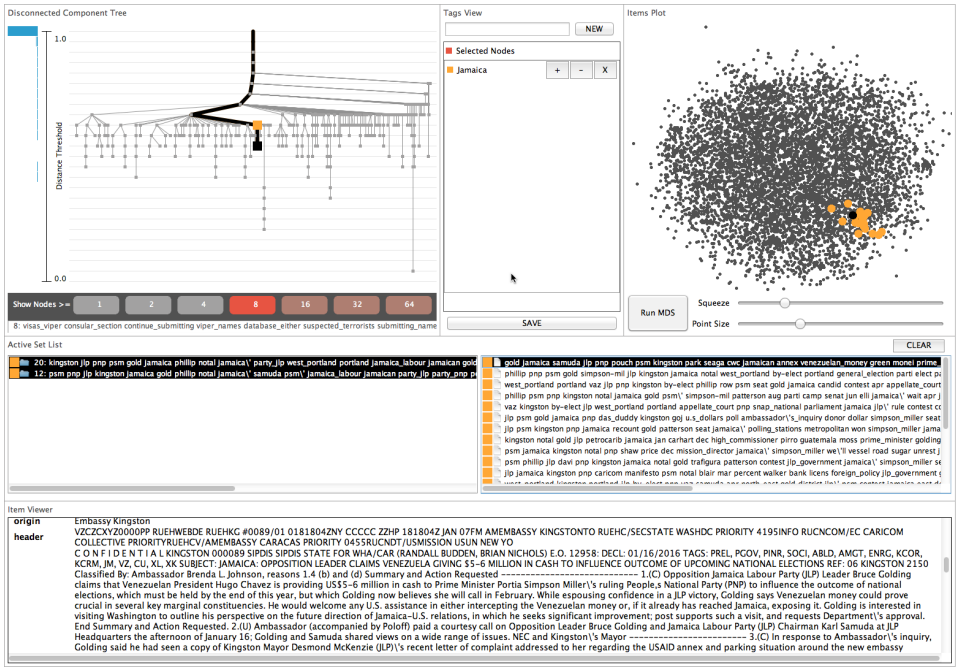


Figure 5.8: Cable alleging Venezuelan influence in Jamaican politics. The journalist found cables alleging campaign financing of a Jamaican political candidate by Chavez.

using the tool as a method not for avoiding reading all the documents, but for *organizing* his reading of the email conversations into categories. He praises the efficacy of Overview, stating, "In the end, I'm guessing it would have taken four reporters splitting up emails into stacks of a few thousand to do the work I did in two weeks."

The second use-case story we describe occurred during the 2012 US Presidential election campaign. Reporter Jack Gillum used Overview in building a story to reveal that vice-presidential candidate Paul Ryan requested and used federal money from government programs he publicly criticized [38]. Gillum describes his use of the tool in a writeup by Jonathan Stray on the Overview website[7]. In contrast to Wade using Overview's clustering as an *organizational* tool, Gillum reports using

Last accessed July 17th 2013

[7] http://overview.ap.org/blog/2012/11/document-mining-shows-paul-ryan-relying-on-the-the-programs-he-criticizes/ Last accessed July 17th, 2013

the clustering as a *filtering* tool to specifically avoid having to read all the documents in his analysis. Gillum knew that large swaths of the documents in the dataset contained no useful information for his story, and so he used the clustering algorithm to identify and cull these swaths.

These two use-cases provide evidence that our tool design accelerated the target task of document analysis for non-experts users: working journalists having no experience with high-dimensional analysis. We were encouraged that these analyses led to published articles. Furthermore, we discovered our design is flexible enough to accommodate different analysis styles of our target users. As future work, we plan to present a detailed account of several different use-cases of the Overview tool, with the hopes of improving the design of analysis tools for non-expert users.

Chapter 6

Glint: An MDS Framework for Costly Distance Functions

All MDS algorithms work by minimizing an objective function quantifying the distortion of the points in the low-dimensional space relative to their original input configuration. Though the different MDS algorithms compute coordinates in a wide variety of ways, in each case the computational work can be divided into two parts: **distance calculation**, where the inter-point distances are calculated from the input points, and **layout calculation**, which reads the computed high-dimensional distances and positions the points in the low-dimensional space.

The focus of this chapter is Glint, an iterative algorithm framework for automatically minimizing distance calculation in MDS. Structurally, Glint forms an outer loop around a modified MDS algorithm. It starts with an empty distance matrix, densifying the matrix as the outer loop iterates, automatically terminating when the MDS layout is stable. Glint separates the distance calculation portion of the MDS algorithm from layout calculations and provides an automated termination procedure.

The time cost of individual high-dimensional distance calculations have a profound effect on the run time of an MDS algorithm. Even for an efficient metric like the 10-dimensional Euclidean distance function, the time spent calculating high-dimensional distances occupies almost 80% of the algorithm run time using the Glimmer force-directed MDS algorithm [57]. Many real-world problems where

MDS is used require more costly distance functions than the Euclidean case. In these more expensive cases, total distance costs occupy more than 99% of MDS run time using the same algorithm. Thus, an efficient MDS algorithm should seek to minimize the *total* work done, minimizing the sum of both the distance and layout work.

Previous work has assumed that individual distance computations are fast to calculate and thus has not sought to automatically resolve the balance between distance and layout work. Current fast MDS algorithms that handle distance matrices either compute many more distances than necessary [57], or leave the total number of distances to compute as a tuning parameter and so do not have a fully automatic way to terminate [15, 30]. The Glimmer algorithm is an example of the overcomputation shortcoming [57]. It computes an *iterative* MDS approximation using force-directed heuristics. The Glimmer minimization strategy defines a cheap iteration and then iterates until convergence is detected. Within each iteration, both distance calculations and layout calculations are done. Glimmer automatically chooses the number of distance calculations to make before terminating, but computes more than are strictly necessary. The Pivot MDS algorithm is an example of the termination shortcoming [15]. It computes a one-step *analytic* MDS approximation. The MDS work is cleanly divided between distance calculation up front followed by a single contiguous layout calculation. Pivot MDS computes all the distances it uses up front, but does not know how many to select.

The above examples motivate a synthesis of the benefits of the two algorithms, keeping the automatic termination of algorithms like Glimmer while separating the distance work from the layout work as in algorithms like Pivot MDS. The goal of Glint is thus to not only compute far fewer distances than the iterative approximation, but also to remove the tuning parameter from the analytic approximation.

To demonstrate the generality and robustness of the Glint approach, we devise Glint instantiations for three very different classes of MDS algorithm: force-directed, analytic, and gradient-based. We present the design of the Glint components for each instantiation, where each is tailored to the requirements of the underlying MDS algorithm. We then show that these Glint instantiations drastically reduce total run time on datasets with costly distance functions without penalizing the final layout quality.

6.1 Distances In MDS

The distances between the points in a low-dimensional MDS solution are intended to closely model those in the high-dimensional input dataset. The core premise of MDS is that the input contains redundant information, allowing for correct output even with an incomplete set of distances as input. Glint exploits this redundancy by iteratively constructing a subset of distances that is as small as possible. This section describes two issues concerning these distances: the existence and effect of expensive distance functions, and how sparse the input distance matrix can be.

6.1.1 Expensive Distance Functions

Minimizing the total number of distances computed is especially important when the time spent computing distances dominates the time spent computing the layout. Many real-world applications involve datasets with expensive distance functions. Even the straightforward Euclidean distance metric can be costly if the number of dimensions is large enough, for example in the millions. In image processing, the Earth Mover's Distance, or EMD, compares the similarity of color distributions between images and is useful for ranking images for querying and nearest-neighbor-type calculations [93]. Its calculation requires solving a linear program, often a costly operation relative to the layout calculation per point. Computational complexity is not the only reason for distance calculation cost. Distances based on database lookups are costly due to the relative speed of disk I/O to memory reads. Distances that involve elicitation of human judgement can be the most costly of all, because the time scales of human response are so much longer than of automatic computation. Human-elicited distances are of interest in many domains; in a marketing example, a single distance is derived from the averaged similarity judgements elicited from survey takers comparing two items [71]; in a psychophysics example, distances are derived from just noticeable differences in haptic stimuli [111].

In all of these cases, distance calculations can comprise well over 99.9% of the total time to compute the MDS layout. We will show that using the Glint framework can drastically reduce the time spent computing distances without compromising the final quality of the MDS layout.

6.1.2 Experimental Analysis of Sparse MDS Solutions

Spence and Domoney conducted a series of data experiments to determine if there could be an *a priori* way to select an optimal subset of distance matrix entries to compute prior to MDS layout [107]. Their experiments investigated the effect of controlling three factors pertaining to layout quality. The first two factors, the amount of noise in the distance measurement and the number of input data points, cannot be manipulated by an MDS algorithm. The last experimental factor they tested, which an algorithm can indeed control in practice, is distance matrix density, or how densely sampled the approximation of the distance matrix is compared to the full version.

The experiments resulted in two key findings that pertain to our work. First, only a fraction of the matrix, ranging from 20% to 60% of the distances on their example data, needed to be computed to accurately approximate the full layout. This finding verifies that the goal of minimizing distance computations is a reasonable one. Second, their results imply that there is no direct way to assess in advance exactly how many distances need to be computed. We thus designed Glint to run online, determining the optimal number of distances to compute on the fly.

6.2 Glint Algorithm Framework

Glint is an algorithm framework: an algorithm with modular components that are themselves algorithms. Figure 6.1 shows a diagram of these three components; each corresponds to a step in the Glint outer loop. Glint starts with an empty distance matrix and a random layout and then loops over the following three main steps to determine a final layout. First, in the *Densify Matrix* step, it selects a new subset of the distance matrix to compute with the distance matrix densification strategy **DS** and then updates the matrix with the computed values. In the *Lay Out Points* step, it updates the layout using the new distance information as input to the MDS layout algorithm **M**. Finally, in the *Check Convergence* step, it checks to see if the change in the objective function **S** is below a threshold ε. If convergence is detected, the last layout is returned, otherwise the loop repeats.

The MDS layout algorithm **M** takes as input a low-dimensional point configuration as the starting point and a sparse distance matrix. To qualify for use in Glint,

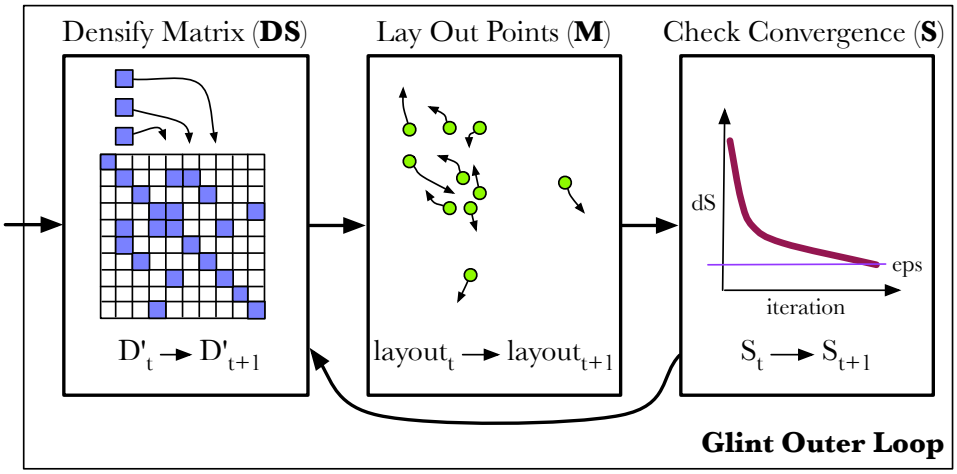


Figure 6.1: Diagram of Glint execution.

M must possess three characteristics. First, it must be able to compute a layout given a distance matrix. Next, it must be able to handle an incomplete – that is, sparse – distance matrix, given the Glint strategy of gradual densification. Finally, **M** must compute a layout from a given starting position rather than starting from scratch each time, so that subsequent outer loop iterations start **M** from a state closer to the final layout configuration. We discuss **M** further in Section 6.3.1.

Controlling the density rate and pattern of the distance matrix is the job of the densification strategy **DS**. Some MDS algorithms, such as Pivot MDS and Landmark MDS are able to compute layouts with incomplete matrices, but the precise sparsity pattern of the incomplete distance matrix may be constrained. Because matrix sparsity pattern requirements vary from algorithm to algorithm, we must tailor the selection of computed distances **DS** to the MDS algorithm **M**. We discuss **DS** further in Section 6.3.2.

Glint requires a cheap, monotonic objective function **S** in order to measure layout convergence; it must also be tailored to the MDS algorithm **M**. It should not invoke a costly full stress function that requires computing all the high- and low-dimensional distances, which would obviate all performance benefits of the system. We discuss **S** further in Section 6.3.3.

6.2.1 Glint Outer Loop

Algorithm 1 Pseudocode for Glint, with variable definitions.

function GLINT(ε)

layout$_0 \leftarrow$ RANDOMLAYOUT
$t \leftarrow 0$
$s_{old} \leftarrow$ MAXVALUE
while !converged **do**
$P_{t+1} \leftarrow$ DS(P_t)
$D'_{t+1} \leftarrow$ DISTANCE(D'_t, P_{t+1}, d)
layout$_{t+1} \leftarrow$ M(D'_{t+1}, layout$_t$)
$s_{new} \leftarrow$ S(P_t, D'_{t+1}, layout$_{t+1}$)
converged $\leftarrow |s_{old} - s_{new}|/s_{old} < \varepsilon$
$s_{old} \leftarrow$ S(P_{t+1}, D'_{t+1}, layout$_{t+1}$)
$t \leftarrow t+1$
return layout

Variable	Description
t	the current Glint iteration
ε	termination threshold
MAXVALUE	maximum possible value for scalar objective function
d	the distance metric $d(i,j)$
s_{old}	scalar objective function value on the previous layout
s_{new}	scalar objective function value on the current layout
layout$_t$	layout coordinates at iteration t
P_t	the set of computed point pairs at iteration t
D'_t	the sparse distance matrix with nonzeros specified by P_t

The Glint algorithm consists of a single threshold-controlled loop, similar to algorithms like gradient descent where the algorithm loops until the change in measured progress becomes very small. Figure 1 lists pseudocode for Glint. The algorithm initializes with a random configuration of points `layout` and then iterates through the main loop. In the main loop, we first call the densification strategy **DS**. On the first call it constructs the initial sparsity pattern P_t of the distance matrix D'_t, and on subsequent calls it densifies the pattern by filling in more nonzero entries. Specifically, the sparsity pattern P_t contains the set of nonzero indices of

the D'_t at time t. After selecting the precise entries to change, Glint updates the sparse distance matrix D'_t by invoking the distance function for each pair of points contained in P_t. Next, the MDS algorithm **M** runs to termination with the starting point `layout` and the input distance matrix D'. Glint itself terminates when the change in the objective function **S** is less than the termination threshold ε.

The objective function **S** takes three parameters: the sparsity pattern P that specifies the pairs of points over which we compare high-D and low-D distances, the distance matrix D' from which we read the distances specified by pairs in P, and the low-dimensional layout coordinates `layout` from which we compute the low-D distances. The reason for including the pattern P as an input instead of simply summing over the entirety of D' is subtle, but important. Glint terminates when the objective function converges; that is, when it stops changing between subsequent iterations. Thus, the objective function must compare results at time $t+1$ to results at time t. However, not only do the points in the layout change between iterations, but the number of terms in the distance function changes, because there are more nonzero entries in D'_{t+1} than in D'_t. To properly measure convergence, we need to compare functions with the same number of terms. Including the same sparsity pattern in the objective calculation ensures that we compare objective functions with equivalent terms at each iteration, by specifying which entries of the matrix to use. Thus, in the Algorithm 1 pseudocode, s_{new} is computed with the sparsity pattern from the previous iteration, P_t, to determine which entries to include in the computation, while using the actual values derived from the current layout at time $t+1$.

6.3 Glint Instantiations

A Glint *instantiation* substitutes implementations of three concrete components into the abstract framework of the Glint algorithm. We describe three Glint instantiations, one for each of the three different MDS algorithm families described in Section 2.1.2: force-directed, analytic, and gradient.

Several of the Glint instantiations require choosing input parameters, as discussed in detail below. Table 6.1 summarizes the default value of each parameter and our method for selecting it. It also includes our analysis of the tradeoffs, with

the results for setting the parameter too small or too big.

Parameter Name	Instantiation	Default	Selection Method	If Too Small		If Too Big	
ε	all	0.001	benchmark	T:slower	Q:better	T:faster	Q:worse
`numIters`	gradient-based	100	benchmark	LT:faster	DT:slower	LT:slower	DT:faster
`numDists`	all	$\log N$	parameter doubling	T:faster	Q:worse	T:slower	Q:better
`numRunsF`	force-directed	5	benchmark	LT:faster	Q:worse	LT:slower	Q:better
`numRunsA`	analytic	10	benchmark	LT:faster	Q:worse	LT:slower	Q:better
`trainSize`	force-directed, analytic	3	benchmark	T:faster	Q:worse	T:slower	Q:better

Table 6.1: Parameters used in Glint instantiations, their default values, how they were chosen, and the tradeoffs in setting them too small or too big. T is total time, LT is layout time, DT is distance calculation time, and Q is layout quality.

6.3.1 Component M: MDS Algorithm

The **M** component takes as input the low-dimensional input coordinates and places them in a new configuration based on the current distance matrix D' as output. For the analytic instantiation we substituted the Pivot MDS algorithm [15] for **M**, and for the gradient implementation we substituted the SMACOF algorithm [29] for **M**. The Pivot MDS algorithm is used without change, but the other instantiations require algorithm parameter choices or internal modifications which we detail in the following subsections.

Gradient-Based Instantiation

For the gradient-based instantiation, the SMACOF MDS algorithm has two tuning parameters: the inner termination threshold, ε, and the maximum number of inner-loop iterations before termination, `numIters`. We use the same value for ε as in the main Glint algorithm.

We observed that the gradient of the stress function for very sparse input matrices quickly shrinks in proximity to a minimum. Setting `numIters` too large results in over-optimizing with incomplete distance information, while setting it too small leads to computing more distances than are necessary. We select 100 as a good balance over all our benchmarks between these two extremes.

Force-Directed Instantiation

In the force-directed instantiation, we substitute a modified version of the Glimmer [56, 57] algorithm for **M**. We used the version of Glimmer that supports distance matrix calculations in addition to handling points [56]. To make the Glimmer algorithm suitable as the **M** component, we must alter the randomized sampling regime used by the algorithm. In Glimmer, sampling is uniform and unconstrained over the entire distance matrix. Glint, however, only feeds a sparse subset of the distance matrix D' to **M** for each outer loop iteration. To compensate, we constrain Glimmer sampling to be uniform over the given nonzeros of the sparse distance matrix D'.

6.3.2 Component DS: Densification Strategy

The **DS** component determines which distances to compute at each Glint iteration. For each instantiation, we follow a strategy of adding `numDists` new distances per point to the matrix D'. By default, the `numDists` parameter is initially set to $\lceil \log_{10} N \rceil$.

Setting the `numDists` parameter to an overly small value would result in an objective function **S** change that is less than the termination threshold ε and thus an incorrect algorithm termination after the first iteration. A small `numDists` is analogous to performing gradient descent with too small a gradient step-size. To ensure `numDists` is large enough, we follow a simple strategy of doubling `numDists` during the first iteration until we achieve a change in the objective function **S** greater than ε.

The distribution of new distances across the matrix D' varies for each instantiation. We describe these distributions on a per-instantiation basis.

Gradient Instantiation

The gradient instantiation **DS** is the simplest of the densification strategies. At each iteration, the **DS** uniformly samples `numDists` distances per point without replacement.

Force-Directed Instantiation

The force-directed **DS** is similar to the gradient instantiation, except for a single modification addressing Glimmer point hierarchies. The Glimmer algorithm divides points into a pyramid of levels, with the fewest points contained in the top level and increasingly larger sets of points at lower levels [57]. Sampling uniformly without replacement from the distance matrix would often lead to the case that, at the top level, several points will not have any distances computed between any of the other points in the top level, only distances computed to points in lower levels. To solve this problem, the force-directed **DS** samples `numDists` distances without replacement once for the points contained in each level. The sampling for a given level is constrained to be uniform over only the points contained in that level.

Analytic Instantiation

Pivot MDS works by operating on a subset of complete columns of the distance matrix. The uniform sampling of distances per point used by the other instantiations would violate this constraint by allowing zeros within columns. We instead compute `numDists` new columns of the distance matrix at each iteration. New columns are chosen using the *MaxMin* strategy described in the Pivot MDS paper [15] starting from a single column chosen uniformly at random.

6.3.3 Component S: Objective Function

Glint objective functions **S** are fast approximations of the true objective functions **F** that are far more costly. In each of the Glint instantiations, **S** fits the following template:

$$S < hi, low, sel > (P, D, layout) = \frac{\sum_{(i,j) \in sel(P)} (lo(i,j) - hi(i,j))^2}{\sum_{(i,j) \in sel(P)} hi(i,j)^2}$$

Here we use the $< \cdot >$ template notation from the C++ language to indicate parameters to **S** that do not change at runtime. The template parameters $hi(i,j)$ and $lo(i,j)$ are functions defining the high and low-dimensional distances between points i and j. The *hi* function varies from dataset to dataset, while lo is always the Euclidean distance function operating on the *layout* input parameter. The *sel* template parameter is an index-selection function that selects a subset from the set of nonzero distance matrix indices P. Intuitively, this function just measures the normalized sum of distance residuals between the layout points and the data, but only for a small set of point pairs instead of all pairs of points.

Because they are stress-based techniques that minimize distance residuals, the force-directed and gradient-based instantiations use D'_{ij} for $hi(i,j)$ and the low-dimensional Euclidean distance for $lo(i,j)$. The analytic instantiation is *strain*-based, minimizing the inner-product residuals. To measure strain, we set $hi(i,j)$ to be the inner product of ith and jth rows of the double-centered matrix C and set $lo(i,j)$ to be the inner product of the ith and jth layout coordinates. The interested reader should refer to the original Pivot MDS paper for more details on the definition of double-centering and the efficient computation of C [15].

For the analytic and gradient-based instantiations, the index-selection function *sel* selects the entirety of the nonzero matrix indices P. In contrast, the force-directed instantiation selects a subset of P. The precise subset of P is the set of point indices contained in the union of per-point random sample caches used by the Glimmer algorithm.

Each instantiation employs randomized sampling of new distance matrix indices after each Glint iteration, as mentioned in Section 6.3.2. In the case of the gradient-based instantiation, this random sampling does not impart enough random noise to the observed values of $\mathbf{S}$ to induce an unexpected termination. However, in the force-directed and analytic cases, we observed enough noise in the sequence of $\mathbf{S}$ values that early termination was regularly observed. The sequence noise was observed to be Gaussian distributed (we confirmed normality with a Shapiro-Wilk test result of $p = 0.55$ [102]). In this section we describe our strategy for creating a smooth $\mathbf{S}$ from the noisy series of raw objective function values.

The obvious first choice for smoothing in Glint would be to convolve the signal with a noise filter, as is done in the force-directed Glimmer algorithm. However, while that approach works well for Glimmer, it is not adequate for detecting convergence reliably within Glint. First, the smoothing filter size used in Glimmer is much bigger than the expected number of outer loop iterations. Reducing the filter window size would change the frequency response of the filter, allowing noise to leak into the signal. Furthermore, moving averages correspond to an impulse response filter that is finite. However, the noise in the sparse stress signal is Gaussian white noise, with equal power across all frequencies, so it will manifest itself after filtering any bandwidth.

Rather than attempt to filter our the noise using convolution or averaging, we instead take a more direct approach and model the observed noise in the sparse stress signal explicitly. Stochastic processes where any subset of process samples are normally distributed are known as Gaussian processes and can be accurately modelled by the machinery of Gaussian process regression (GPR) [88].

In order to perform GPR we must select the forms of the two functions that completely determine a Gaussian process, the mean and the covariance function. The mean of the Gaussian process encodes information about the shape of the underlying process, for example whether it is linear or constant. We chose a mean

prior of zero, indicating that we have no advance knowledge about the signal. We select the squared exponential function, one of the most commonly chosen covariance functions [88], because it models smooth transitions between adjacent values of **S**, a behavior that matches our expectations for the convergence curve.

We can improve our smooth estimate of the mean of **S** by increasing the number of samples computed at each outer loop iteration. In the force-directed case, we compute more samples by restarting **M** with the same initial layout and a different random seed. Since the analytic case proceeds deterministically, the same technique cannot be used. To compute a set of random samples for the analytic case we were inspired by bootstrap resampling methods [31]. In order produce a single sample we select `numDists` columns uniformly at random to leave out of *P*.

For the parameter designating the number of computed samples per Glint iteration, there is a parameter tradeoff between the fidelity of the estimated mean, which affects the likelihood of observing a false termination, and the speed of algorithm. We empirically find that computing 5 runs for the force-directed `numRunsF` parameter and 10 runs for the analytic and `numRunsA` parameter yields good results over all our benchmark datasets.

Using GPR requires initialization of the so-called process *hyperparameters* of the squared exponential covariance function. These include the length scale, or degree of smoothness, and the noise level. The hyperparameters can be efficiently learned from a small set of observations computed during the first `trainSize` iterations of the Glint outer loop, by optimizing a likelihood function using conjugate gradients. We empirically find that using 3 iterations for training yields good results over all our benchmark datasets.

6.3.4 Instantiation Design Summary

Table 6.2 summarizes the Glint component design decisions, emphasizing the underlying algorithm features that crosscut the three instantiations. Consideration of these features could guide designers of future instantiations. For example, an algorithm using the entire sparse input distance matrix, like Pivot MDS and SMACOF, can remain unaltered for **M**. Algorithms with objective functions **S** that are noisy,

Alg. Class	**M**	**DS**	**S**
force-directed	altered sampling	uniform pointwise for each hierarchy	GPR smoothed stress-based across sample-cache sets
gradient-based	unchanged	uniform pointwise	stress-based across P
analytic	unchanged	uniform columnwise	GPR smoothed strain-based across P

Table 6.2: Glint component design summary for each MDS algorithm class.

such as Pivot MDS and Glimmer, can employ GPR smoothing.

6.4 Results

We present the results in terms of a benchmark performance comparison and an assessment of convergence. We first describe the benchmark datasets in detail. We compare the efficiency and quality of Glint instantiations against the standard algorithms in terms of time and stress using these benchmarks. We then discuss convergence issues and demonstrate convergence behavior of each instantiation.

6.4.1 Dataset and Distance Function Description

The `molecule` dataset contains 661 points representing polymer-based nanocomposites. The distance function is cheap: it is the Euclidean distance metric where the number of dimensions m is 10. We include this dataset as a baseline where the Glint requirements are not met and unmodified algorithms should be employed instead. The 4000 points in the `concept` dataset are biomedical terms where the distance function to determine their co-occurrence in journal articles requires running database queries. The `Flickr` dataset contains 1925 images culled from the author's public photo collection, with distances computed using the Earth Mover's Distance (EMD) [93]. The `BRDF` dataset is an example from the computer graphics literature, where computations involving 100 points representing images use the Euclidean distance function. The number of dimensions m is four million [76]; this function is expensive despite being Euclidean because of the huge number of dimensions. The `videogame` dataset was created by gathering human judge-

ments in response to survey of questions about 96 games. While the exact timing information for the judgments was not reported [71], our conservative estimate is that the sum of the response times of the human participants took an average of 10 seconds for each pairwise comparison. Table 6.3 summarizes our benchmark distance functions and costs.

d cost (sec)	Distance Calculation	Benchmark
0.00001	Euclidean $m = 10$	`molecule`
0.001	DB Query	`concept`
0.01	Earth Mover 8^3 signature	`flickr`
1.0	Euclidean $m = 4$M	`brdf`
10.0	Human Elicited	`videogame`

Table 6.3: The cost d of a single distance calculation for the benchmark datasets in seconds rounded to the nearest power of 10. Here m represents the number of dimensions of the input data in the case of using a Euclidean distance function.

6.4.2 Benchmark Speed and Quality Comparison

We validate Glint by comparing the benchmark performance of our implementations against the previous work in terms of speed and quality. Speed is measured in seconds to termination and quality is measured in terms of the full objective function **F** using the entire distance matrix D. For the force-directed and gradient-based instantiations, **F** is the full normalized stress function [13]. For the analytic instantiation, **F** is the full normalized strain function. We compute **F** only for performance validation; it is never computed in practice. All recorded values are averaged over 5 runs on an Intel Core 2 QX6700 2.66 GHz CPU with 2 GB of memory.

For the original approach in the force-directed and gradient-based performance comparison, we ran the Glimmer and SMACOF algorithms, respectively, with the same ε for these as used in Glint. For the original approach used in the analytic performance comparison, we know of no algorithms with termination criteria. Instead, we used a human-in-the-loop Pivot MDS setup, where the first author added `numDists` pivots at a time with a keystroke, and manually halted the process after visually assessing layout convergence. The Pivot MDS algorithm is unable to han-

dle incomplete distance matrix columns, so we omit the `videogame` benchmark, which possesses many missing matrix entries, from the analytic results.

Figure 6.2 and Table 6.4 compare the execution time and final layout quality of Glint to the original approaches.

The speedup of the force-directed instantiation ranges from 20 to 115 for the costly target cases, while the original Glimmer algorithm is several times faster for the cheap baseline. The main benefit of the fully automatic analytic Glint instantiation is the elimination of the need for manual monitoring and intervention. The Glint instantiation was faster than Pivot MDS with a manual operator in the loop for `molecule` and `flickr`, but slower for `concept` and `brdf`. The speedup of the gradient-based Glint instantiation is dramatic: several orders of magnitude in the target cases, and a factor of two in the baseline case of `molecule` where the distance function is cheap.

The quality values for Glint are roughly the same magnitude and variability for each benchmark in the force-directed case. For the analytic instantiation, the quality values are equal or better than the manual Pivot MDS method. In the gradient case, most of the final quality values, except `molecule`, are slightly worse than the standard approach using the full distance matrix. The gradient Glint instantiation provides a speed and quality compromise between the extremely costly but accurate full gradient approach, and the fast but approximate force-directed Glint instantiation.

6.4.3 Convergence

We illustrate the convergence behavior of each Glint instantiation in Figure 6.3. Each log-scale plot displays two curves: the blue curve represents the value of the full, slow objective function **F** of the layout after each Glint iteration, while the orange curve shows the value of the smoothed, fast objective **S**. For those instantiations that employ GPR smoothing, we also plot the random samples used in the regression as gray dots. Similarly, for those instantiations that employ an iterative layout algorithm **M**, we plot the values of **S** after each **M** iteration. As in the benchmark comparison, **F** is the full normalized stress function for Glimmer and SMACOF, and **F** is the full strain function for Pivot MDS.

Benchmark	Glint **F**	Orig. **F**	Glint Time	Orig. Time	Speed up
Force-Dir.					
`molecule`	0.03	0.03	14	4	0.2
`concept`	0.18	0.18	49	1016	20
`flickr`	0.08	0.09	2.4K	98K	40
`brdf`	0.03	0.04	3K	304K	115
`videogame`	0.45	0.45	23K	482K	20
Analytic					
`molecule`	0.35	0.42	3	23	9
`concept`	0.93	0.94	96	63	0.7
`flickr`	0.48	0.59	1.2K	2.9K	2
`brdf`	0.078	0.233	40K	6K	0.2
Gradient					
`molecule`	0.01	0.03	360	700	1.9
`concept`	0.18	0.18	0.1K	113K	880
`flickr`	0.06	0.04	8K	71M	8.8K
`brdf`	0.008	0.005	4K	859K	200
`videogame`	0.16	0.13	19K	430K	220

Table 6.4: Comparison of full objective functions, time (in seconds), and speedup between Glint instantiations and original MDS algorithms.

The magnitude of the change in the cheap objective **S** approximates that of the change in costly **F** function. In the case of Pivot MDS, the smoothed **S** series is slightly offset from the gray random samples due to the effect of using sparsity patterns from the previous iteration. These benchmarks validate the claim that setting ε to a given termination threshold will terminate Glint when the corresponding change in **F** falls below the threshold modulo some sampling noise.

Figure 6.2: Comparison of speed (top) and quality (bottom). In each pair, the top blue bar is the original MDS algorithm, and the bottom orange bar is the Glint instantiations. The black lines indicate 95% standard error bars.

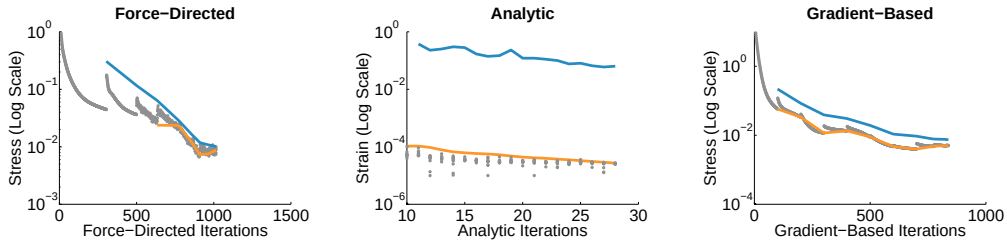


Figure 6.3: Log-scale Glint convergence curves on each instantiation generated using the `brdf` dataset. The orange **S** curve is derived from the noisy grey samples. **S** is designed to match the convergence behavior of the costly **F** series in blue.

Chapter 7

Conclusion and Future Work

We conclude the thesis by summarizing the finer points of the different thesis chapters. We then describe a set of future research directions that build upon and expand our work. We draw the thesis to a close with a set of lessons learned from our research.

7.1 Conclusions

Difficulties arise when dimensionality reduction is applied to the important use-cases of non-expert users, document data, and costly distance functions. In this thesis we identified the obstacles associated with each of these cases and explored ways to address the underlying problems. For non-expert users of dimensionality reduction, we identified the need for two kinds of user guidance, local and global, and designed a system, DimStiller, that encapsulates both. In the case of document data, we have identified the mostly-disconnected property of the data and its connection with query algorithms from Information Retrieval. We then presented algorithms for high-dimensional analysis, including dimensionality reduction, that take advantage of the mostly disconnected-property for improved efficiency and accuracy. For the case of costly distance functions, we identified an inefficiency in the design of multidimensional scaling algorithms with respect to this case and presented an algorithm framework, Glint, to minimize total running time without a penalty to output quality.

7.1.1 DimStiller

In Chapter 3, we presented DimStiller, a data analysis system that uses a set of abstractions to structure and navigate dimensional analysis and reduction: data resides in tables, operators modify and visualize tables, expressions chain together operators, and workflows permit pattern re-use. DimStiller uses these mechanisms to provide both local and global guidance through the analysis space of possible data tables. In both of the presented case studies in Section 3.4, we showed how individual operators probe the input dimensions and produce values like variance, correlations, and principal components and visualizations such as scree plots and scatterplots. It is the data analyst's task to turn these quantitative figures into answers to qualitative questions about the data. DimStiller builds target users' trust in these answers by providing an intuitive pipeline architecture that visually guides users through making algorithm and parameter choices.

7.1.2 Algorithms for the Visual Analysis of MoDisco Data

Chapter 4 showed how the MoDisco property of real-world document datasets is an important consideration in the design of algorithms for data analysis. This property has not been previously addressed in the visualization literature; we generalize algorithms and data structures originally designed for information retrieval for visualization applications. MoDisco data has both sparse vector columns and rows; this property has important implications for the design of search query algorithms. We showed how impact-ordered query algorithms implicitly use this property to compute the higher-scoring queries faster. When data analysis algorithms, such as dimensionality reduction, target TF-IDF document data, they can leverage the MoDisco property to efficiently compute important data structures for data exploration: nearest-neighbor sets, distance matrices, cluster trees, and 2D layout coordinates. We then presented three scalable algorithms for computing each of these important data structures, with results that improve over the state-of-the-art.

7.1.3 Overview

In Chapter 5, we presented the Overview prototype application for this task that combines a hierarchical clustering view and a traditional MDS view. We vali-

dated the application with two complex, real-world datasets from the journalism domain: subsets of the diplomatic cables and Afghan war logs from WikiLeaks. The Overview prototype application allowed direct comparison of different item encoding, clustering, and layout algorithms with respect to each other and a fixed set of human-assigned tags, opening up a line of research into the semantic relationships between these different algorithmic stages in the visual exploration pipeline.

7.1.4 Glint

Chapter 6 illustrated how expensive distance calculations change the efficiency of existing MDS algorithms like Glimmer and SMACOF. Such algorithms compute more distances than are required for an existing quality of layout, while analytic algorithms require manually tuning the number of distances to compute as an input parameter. We solve both these problems with Glint, an algorithm framework with three components: a distance matrix densification strategy **DS**, an algorithm **M**, and an inexpensive objective measure **S**. Given these components, Glint samples distances from the distance matrix in fixed batches, updating the low-dimensional layout with new information until the layout quality converges. We showed how careful design of termination criteria can overcome the noise effect of random sampling on convergence. We presented and validated Glint instantiations for three separate types of previous MDS algorithms: the force-directed Glimmer, the analytic Pivot MDS, and the gradient-based SMACOF.

The Glint instantiations presented give essentially equivalent layout quality in all cases. The analytic instantiation was roughly equal in time performance to Pivot PDS, with some cases of speedup and some of slowdown; the main contribution of Glint in this situation is to remove the need for manual monitoring and intervention. The iterative instantiations showed substantial speedups against Glimmer and SMACOF in all of our target cases with costly distance functions, ranging from 20 to 115 with the force-directed Glint instantiation and from 200 to 8800 with the gradient-based Glint instantiation.

7.2 Future Work

We now present a series of stepping-off points for future research based on the contents of this thesis.

7.2.1 User Guidance

A form of guidance not yet explicitly provided by DimStiller is to help the user fully understand the inner workings of the supported operators. For example, users could be given visual feedback highlighting relevant regions of scree plots in the reduce operator control panels. While the DimStiller architecture should in theory support this kind of targeted exploration, it would require significant future work to design a system that truly sheds light on all black-box algorithms. Another direction of future work will be to add more encoding and interaction techniques previously shown to be effective for high-dimensional analysis, for example sorting the `Collect` operator matrix view for the rank by feature capability suggested by Seo and Shneiderman [100].

7.2.2 Efficient DR with Costly Distances

The Glint system specifically targeted MDS algorithms. As discussed in section 2.1, there are other types of dimensionality reduction algorithms, like probability based methods, that also rely on distance information. It would be interesting work to apply the Glint framework to these other types of DR algorithms.

Another interesting avenue of future work is to explore different types of densification strategies. The examples we provide in Glint all use uniform sampling, but a more sophisticated sampling based on proximities of points in low-dimensional space might also be employed to good effect.

7.2.3 Mostly Disconnected Data

There are several different avenues for possible future work with respect to the the analysis of mostly-disconnected data. One is to determine if data other than term-vector databases can exhibit the MoDisco property. We conjecture that using an IDF-like transformation of high-dimensional vector data in other domains could induce MoDisco structure for great effect. Another area for future work is

modification of the APQ nearest-neighbor algorithm to include other innovations from query algorithms like impact transformations of the input data and accumulator limiting and pruning [2]. It may also be fruitful to devise ways to incorporate efficient use of truncated distance matrices and inverted files into other clustering algorithms like k-medoids [91].

7.3 Lessons Learned

Shifting the discussion to a higher level of abstraction, we end the thesis with a discussion of some of the overarching lessons learned about conducting interdisciplinary research in the course of writing this thesis.

Many of the projects in this thesis involve bringing together techniques from different disciplines. For example, Chapter 4 draws connections between Information Retrieval and Visualization. The Overview prototype combine ideas from data mining, visualization, and data journalism. Glint attempts to build an algorithm framework that crosscuts algorithm work in Visualization and Statistics. Dimensionality Reduction itself is a subject of interest in several different research communities: Statistics, Machine Learning, Data Mining, and Visualization. Each of these disciplines functions like a lens, focusing on those aspects of algorithm design most beneficial to their own particular objectives.

Below, we first relate the experience of drawing connections between the algorithmic foci of two different research communities. Then, we discuss what factors of an interdisciplinary collaboration can aid in the dissemination of research, leading to greater impact.

7.3.1 Making an Algorithmic Connection

Revealing algorithmic connections between High-dimensional data analysis and Information Retrieval was one of the more challenging and intellectually satisfying aspects of this thesis. In this section, we recount the somewhat roundabout story of how these connections were made. It is our hope that this story functions as an informative exercise in revealing how research across disciplines is often a mix of wrong-turns, back-tracking, and (hopefully) breakthroughs. This contrasts with the standard academic presentation of work as a finished product resulting from a

set of ordered, logical conclusions. We then discuss how a multifaceted research process can actually work to create multiple connections between different fields and enrich the underlying algorithms developed along the way.

Glimmer and Sparse Docs

One of the more compelling set of results produced from our validation of the Glimmer MDS algorithm was from the `docs` dataset [57]. This dataset, referred to as `metacombine` in Chapter 4, was our first encounter with analyzing term-vector data using techniques from high-dimensional analysis. The discrepancy between the `docs` result from PivotMDS, a Classical Scaling algorithm, and the result from Glimmer, a Distance Scaling algorithm, motivated an in-depth analysis of when Distance Scaling algorithms are more appropriate than Classical Scaling algorithms. Our intuition was that `docs` was so intrinsically high-dimensional, that its structure was largely simplicial. Therefore, linear projection techniques like Classical Scaling were inappropriate for producing dimensionally reduced layouts because much of the data variance was orthogonal.

Power Transformations and Local Graph Layout

Our success with using Glimmer to produce large document dataset layouts with visible clustering attracted our collaborator, Jonathan Stray, who was interested in producing the tool that would become Overview. We began experimenting with using Glimmer on what would become the `warlogs` dataset in Chapter 5. These experiments revealed that improved cluster structure could be observed with power transformations of the distance function, as described by Buja et al [16]. But one troubling visual artifact of using Glimmer on our data was the persistence of an overall circular layout, inside which the clusters were being forced together. Further analysis of the data revealed this circularity to be an artifact of the predominance of unit length distances in the data.

Unit distances are a result of the cosine distance between two data points sharing no features. Semantically, though, the purpose of a unit distance is actually to imply no connection at all, not a connection of a specific length as was being reported by the cosine distance function. Our next strategy was to attempt to

visually encode the absence of a connection between documents. This encoding was achieved by replacing unit length distances with infinite distances and employ a Box-Cox transformed version of the stress function as specified by Chen and Buja [22]. But this strategy of replacing distances led to many practical complications: the data often broke into multiple disconnected components which would repel each other in the layout, the energy model optimization was prone to getting stuck in inferior local minima, and the Box-Cox force model itself had many sensitive parameters.

Contour Trees and DiscoTrees

At this point, we shifted our focus away from finding a better layout engine, and toward visually encoding the different densities of points in high-dimensional space. Focusing on densities was directly informed by the observation that removing all unit length "edges" in the graph constructed from the distance matrix often resulted in meaningful disconnected components. The disconnected components were in fact densities of points separated by an appreciable distance in terms-space. If densities separated by unit-distance were meaningful, we surmised that many other distances could produce meaningful decompositions. We set about answering the question of whether there are even more interesting component decompositions of the data determined by gradually relaxing the distance threshold.

In our first analysis of the problem, we considered the Contour Tree [18] of the distance field. If one imposes a distance field over the high-dimensional data space, then the value at each point in space is determined by its distance to the nearest data point. Our unit-length decomposition of the data then resulted from the disjoint sets of points contained in the iso-contours of the distance field at a threshold of 1. The Contour *Tree* then represents all possible sets of components produced by the union of all iso-contour values between one and zero. We then set about computing and visualizing this decomposition.

The DiscoTree described in Chapter 5 is the result of an approximate algorithm to calculate this Contour Tree. Our DiscoTree algorithm hinged upon the observation that the maximum distance between points within a contour is bounded by the value of the contour [59]. By comparing the Contour Tree generated by this pro-

cess to other hierarchical decompositions of points, we then concluded that there is an isomorphism between this Contour Tree of a scalar distance field and the hierarchy produced by Single-Link clustering.

Single-Link to Impact-Ordered Inverted Files

Realizing that our DiscoTree algorithm efficiently computes a Single-Link hierarchy, we set about comparing our strategy to previous work computing single-link hierarchies. This comparison revealed a subset of clustering work using inverted-file index structures when clustering documents [81]. Rather than simply compare against this work, we examined current techniques for building and processing inverted files [137]. It was in this deeper comparison that we were able to draw the connection between the approximations we made in our DiscoTree algorithm with approximations made by impact-ordered, inverted-file indices. The connection between inverted files and high-dimensional space then permitted us to connect many of the previously-visited dots and suggest the algorithm design implications in Chapter 4.

Making Connections: Breadth-First vs. Depth-First Research

In developing new techniques, there is a tension between a breadth-first and depth-first approach to researching a solution to a problem. In the breadth-first approach, a researcher constructs an abstract description of their problem and then makes a broad survey of possible techniques, selecting the best overall fit for the solution at hand. In the depth-first approach, the researcher picks a familiar technique, developing and deepening its features until it is appropriate for solving the underlying problem. Both approaches have their merits and risks. Breadth-first connects the problem to a tried and true solution to a similar problem, but at the risk of making only a superficial connection between problem and algorithm. Depth-first often leads to novel work, pushing the frontiers of a technique to new venues, but at the risk of being inferior to algorithms in alternative fields.

Our experience shows there may be benefit in combining both breadth and depth. For example, our work in making modifications to Glimmer and the Contour Tree algorithm were a depth-oriented approach, taking spatial approaches from

visualization and computer graphics and deepening them to better analyze high-dimensional term-vector data. Simultaneously, our drawing connections across dimensionality reduction, hierarchical clustering and information retrieval were breadth-oriented, selecting algorithms appropriate for the tasks of layout-generation, groupings, and querying respectively. The utilization of these two styles simultaneously allowed us to organize a set of disparate fields together (breadth) and then draw low-level connections between them (depth), ultimately leading to deeper insights and improved results.

7.3.2 Research Impact and Collaboration

The other high-level lesson learned in the course of developing this thesis was that of the benefits of a cross-domain collaboration, especially with respect to research impact. Real-world impact, where non-experts derive benefit from the insights or results of research, can be difficult to achieve. A paper by Wagstaff [121], addressing research impact in Machine Learning, suggested that impact is greatly affected by *follow-through*, loosely defined as both "Publicize results to relevant user community" and "Persuade users to adopt technique." In this section, we discuss our experiences with collaborators in developing software systems, and how a collaborator's own goals can have significant effect on the impact of one's research. Sedlmair et al. have presented a detailed approach to collaborator selection during the course of visualization design studies [99]. Our discussion here is focused specifically on *impact* through user adoption, and can be considered a complementary set of suggestions.

DimStiller Collaboration

In Chapter 3, we presented a high-dimensional analysis tool, DimStiller, aimed at providing non-expert data-analysts with guidance in using sophisticated tools and workflows. After developing this software, we collaborated with different researchers and elicited their questions and analysis needs in response to the tool. The motivations and goals of these collaborators were focused on solving their domain analysis problems under aggressive time constraints. For them, taking part in an iterative design process might do more harm than good if the tool doesn't have

immediate benefits to their work process. Our collaboration with these domain analysts led to insights in other research projects [98], but did not lead to further development of the tool, or any popular adoption of the DimStiller software.

Overview Collaboration

Our collaboration with Jonathan Stray on the Overview project contrasts with the DimStiller collaboration. In particular, as a Knight Foundation grant recipient, he was motivated to see our collaboration succeed. In addition, his profile was aligned with two aspects of follow-through highlighted by Wagstaff: publicizing results, and evangelizing the technique for adoption. That is, he had access to high-profile venues, such as the PBS Idea Lab website[1], and he was professionally well-connected enough with our target group to make headway evangelizing the technique to real journalists. As a result, we were able to see the tool put to use by journalists outside of our collaboration. We detailed some of this use in Section 5.4 of the thesis.

Ingredients of High-Impact Collaboration

The Overview project then presents a study in some of the right ingredients for high-impact research. We summarize these ingredients as:

- Collaborator has a stake in the positive outcome of the project.
- Collaborator has a noted platform for publicizing good results.
- Collaborator is well-connected to high-profile users in the community.

Our collaborations on the DimStiller project possessed none of these ingredients, making user adoption more unlikely through this specific collaboration. In our experience, if the ultimate goal of a project is community adoption of a tool, we surmise that project collaborations without these ingredients face an uphill struggle. We do not claim that these factors are either necessary or sufficient for high-impact research. To do so would cynically suggest only seeking out community leaders for

[1] http://www.pbs.org/idealab/2013/04/how-a-computer-can-organize-thousands-of-documents-for-a-reporter110.html Last accessed July 17th, 2013

any collaboration. But we do mean to imply that, in our experience, the existence of any or all three of these facets can go a long way toward seeing greater adoption rates in a target community.

Bibliography

[1] C. Aggarwal and C. Zhai. A survey of text clustering algorithms. *Mining Text Data*, pages 77–128, 2012. → pages 66

[2] V. Anh and A. Moffat. Impact transformation: effective and efficient web retrieval. In *Proc. ACM Conf. Information Retrieval (SIGIR)*, pages 3–10. ACM, 2002. → pages 116

[3] V. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proc. ACM Conf. Information Retrieval (SIGIR)*, pages 35–42. ACM, 2001. → pages 55

[4] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998. → pages 16

[5] J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324:446–449, 1986. → pages 73

[6] R. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th International Conference on World Wide Web*, pages 131–140, 2007. → pages 16

[7] R. Becker and W. Cleveland. Brushing scatterplots. *Technometrics*, 29(2): 127–142, 1987. → pages 2

[8] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in Neural Information Processing Systems*, 14:585–591, 2001. → pages 13

[9] R. Bellman. *Adaptive Control Processes: a Guided Tour*, volume 4. Princeton University Press, 1961. → pages 15

[10] P. Berkhin. A survey of clustering data mining techniques. *Grouping Multidimensional Data*, pages 25–71, 2006. → pages 79

[11] D. Blei. Probabilistic topic models. *Communications of the ACM*, 55(4): 77–84, 2012. → pages 83

[12] C. Böhm and H. Kriegel. A cost model and index architecture for the similarity join. In *Conf. on Data Engineering (ICDE)*, pages 411–420. IEEE, 2001. → pages 16

[13] I. Borg and P. Groenen. *Modern Multidimensional Scaling Theory and Applications*. Springer-Verlag, 2nd edition, 2005. → pages 108

[14] I. Borg and P. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer, 2005. → pages 10

[15] U. Brandes and C. Pich. Eigensolver methods for progressive multidimensional scaling of large data. In *Graph Drawing*, volume 4372, pages 42–53. Springer, 2007. → pages 11, 94, 102, 104

[16] A. Buja and D. Swayne. Visualization methodology for multidimensional scaling. *Journal of Classification*, 19(1):7–43, 2002. → pages 70, 117

[17] A. Buja, D. Swayne, M. Littman, N. Dean, H. Hofmann, and L. Chen. Data visualization with multidimensional scaling. *Journal of Computational and Graphical Statistics*, 17(2):444–472, 2008. → pages 13

[18] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry: Theory and Applications*, 24(2): 75–94, 2003. → pages 118

[19] M. Chalmers. A linear iteration time layout algorithm for visualising high dimensional data. In *Proc. IEEE Visualization*, pages 127–132, 1996. → pages 12

[20] K. Chaudhuri and S. Dasgupta. Rates of convergence for the cluster tree. *Advances in Neural Information Processing Systems*, 23:343–351, 2010. → pages 17

[21] L. Chen and A. Buja. Local multidimensional scaling for nonlinear dimension reduction, graph drawing, and proximity analysis. *Journal of the American Statistical Association*, 104(485):209–219, 2009. → pages 4, 5, 13

[22] L. Chen and A. Buja. Stress functions for nonlinear dimension reduction, proximity analysis, and graph drawing. *Journal of Machine Learning Research*, 14:1145–1173, 2013. → pages 118

[23] Y. Chen, L. Wang, M. Dong, and J. Hua. Exemplar-based visualization of large document corpus. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1161–1168, 2009. → pages 20, 79

[24] D. Cook and D. Swayne. *Interactive and Dynamic Graphics for Data Analysis: With Examples Using R and GGobi*. Springer, 2007. → pages 18

[25] K. A. Cook and J. J. Thomas, editors. *Illuminating the path: the research and development agenda for visual analytics*. National Visual Analytics Center, 2005. → pages 47

[26] F. Crestani and S. Wu. Testing the cluster hypothesis in distributed information retrieval. *Information Processing & Management*, 42: 1137–1150, 2006. → pages 82

[27] W. Cui, S. Liu, L. Tan, C. Shi, Y. Song, Z. Gao, X. Tong, and H. Qu. TextFlow: Towards better understanding of evolving topics in text. *Proc. IEEE Symp. Information Visualization (InfoVis)*, 17(12):2412–2421, 2011. → pages 83

[28] N. de Freitas, Y. Wang, M. Mahdaviani, and D. Lang. Fast krylov methods for n-body learning. In *Advances in Neural Information Processing Systems*, pages 251–258, 2005. → pages 15

[29] J. de Leeuw and P. Mair. Multidimensional scaling using majorization: SMACOF in R. *Journ. Statistical Software*, 31(3):1–30, 8 2009. → pages 13, 102

[30] V. de Silva and J. Tenenbaum. Sparse multidimensional scaling using landmark points. Technical report, Stanford, 2004. → pages 11, 94

[31] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*, volume 57. CRC Press, 1993. → pages 106

[32] B. Everitt, S. Landau, M. Leese, and D. Stahl. *Cluster Analysis, 5th Edition*. John Wiley & Sons, Ltd, 2011. → pages 17

[33] R. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, 7(2):179–188, 1936. → pages 1

[34] M. Fontoura, V. Josifovski, J. Liu, S. Venkatesan, X. Zhu, and J. Zien. Evaluation strategies for top-k queries over memory-resident inverted indexes. *Proceedings of the VLDB Endowment*, 4(12):1213–1224, 2011. → pages 56

[35] E. Fowlkes and C. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383): 553–569, 1983. → pages 68

[36] S. France and J. Carroll. Two-way multidimensional scaling: A review. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 41(5):644–661, 2011. → pages 10, 70

[37] E. Gansner, Y. Koren, and S. North. Graph drawing by stress majorization. In *Graph Drawing*, pages 239–250, 2004. → pages 13

[38] J. Gillum. Ryan asked for federal help as he championed cuts. http://bigstory.ap.org/article/ryan-asked-federal-help-he-championed-cuts. Accessed: 2013-06-04. → pages 91

[39] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, pages 518–529, 1999. → pages 16, 64

[40] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965. → pages 11

[41] J. Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53(3-4):325–338, 1966. → pages 11

[42] J. Gower. Measures of similarity, dissimilarity, and distance. *Encyclopedia of Statistical Sciences*, 5:397–405, 1985. → pages 3

[43] J. Grimmer and G. King. General purpose computer-assisted clustering and conceptualization. *Proc. Natl. Acad. Sciences (PNAS)*, 2010. → pages 83

[44] P. Groenen and W. Heiser. The tunneling method for global optimization in multidimensional scaling. *Psychometrika*, 61(3):529–550, 1996. → pages 11

[45] D. Guo. Coordinating computational and visual approaches for interactive feature selection and multivariate clustering. *Information Visualization*, 2 (4):232–246, 2003. → pages 19

[46] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003. → pages 3

[47] N. Halko, P. Martinsson, Y. Shkolnisky, and M. Tygert. An algorithm for the principal component analysis of large data sets. *SIAM Journal on Scientific Computing*, 33(5):2580–2594, 2011. → pages 10, 74

[48] J. Heer and M. Agrawala. Software design patterns for information visualization. *Proc. IEEE Symp. Information Visualization (InfoVis)*, 12(5): 853–860, 2006. → pages 23, 28

[49] M. Hein and J. Audibert. Intrinsic dimensionality estimation of submanifolds in R^d. In *Proc. Intl. Conf Machine Learning (ICML)*, pages 289–296. ACM, 2005. → pages 21, 31

[50] G. Hinton and S. Roweis. Stochastic neighbor embedding. *Advances in Neural Information Processing Systems*, 15:833–840, 2002. → pages 14, 72, 73

[51] R. Holbrey. Dimension reduction algorithms for data mining and visualization. *University of Leeds/Edinburgh*, 2006. → pages 22

[52] S. Huang, M. Ward, and E. Rundensteiner. Exploration of dimensionality reduction for text visualization. In *Proc. Coordinated and Multiple Views in Exploratory Visualization (CMV)*, pages 63–74, 2005. → pages 47

[53] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2 (1):193–218, 1985. → pages 64

[54] M. Hubert, P. Rousseeuw, and K. Branden. ROBPCA: a new approach to robust principal component analysis. *Technometrics*, 47(1), 2005. → pages 10

[55] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613. ACM, 1998. → pages 15

[56] S. Ingram. Multilevel multidimensional scaling on the GPU. Master's thesis, University of British Columbia Department of Computer Science, 2007. → pages 6, 12, 102

[57] S. Ingram, T. Munzner, and M. Olano. Glimmer: Multilevel MDS on the GPU. *IEEE Transactions on Visualization and Computer Graphics*, 15(2): 249–261, 2009. → pages 5, 12, 21, 31, 74, 75, 81, 93, 94, 102, 103, 117

[58] S. Ingram, T. Munzner, V. Irvine, M. Tory, S. Bergner, and T. Möller. Dimstiller: Workflows for dimensional analysis and reduction. In *Proc. IEEE Symp. Visual Analytics Science and Technology (VAST)*, pages 3–10, 2010. → pages 44, 47

[59] S. Ingram, T. Munzner, and J. Stray. Hierarchical clustering and tagging of mostly disconnected data. Technical Report TR-2012-01, University of British Columbia Department of Computer Science, May 2012. URL http://www.cs.ubc.ca/labs/imager/tr/2012/modiscotag/. → pages 51, 118

[60] A. Inselberg and B. Dimsdale. Parallel coordinates. In *Human-Machine Interactive Systems*, pages 199–233. Springer, 1991. → pages 2

[61] N. Jardin and C. van Rijsbergen. The use of hierarchical clustering in information retrieval. *Information Storage & Retrieval*, 7(5):217–240, 1971. → pages 82

[62] S. Johansson and J. Johansson. Interactive dimensionality reduction through user-defined combinations of quality metrics. *Proc. IEEE Symp. Information Visualization (InfoVis)*, 15(6):993–1000, 2009. → pages 4, 19

[63] P. Joia, D. Coimbra, J. Cuminato, F. Paulovich, and L. Nonato. Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2563–2571, 2011. → pages 13, 75

[64] I. Jolliffe. *Principal component analysis*. Wiley Online Library, 2005. → pages 3, 4, 9

[65] F. Jourdan and G. Melançon. Multiscale hybrid MDS. In *Proc. Intl. Conf. on Information Visualization (IV'04)*, pages 388–393, 2004. → pages 22

[66] M. Khoury, Y. Hu, S. Krishnan, and C. Scheidegger. Drawing large graphs by low-rank stress majorization. *Comp. Graph. Forum*, 31(3pt1):975–984, June 2012. → pages 13

[67] A. Krowne and M. Halbert. An initial evaluation of automated organization for digital library browsing. In *Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries, 2005.*, pages 246–255. IEEE, 2005. → pages 51

[68] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. American Mathematical Society (AMS)*, 7(1): 48–50, 1956. → pages 67

[69] J. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964. → pages 11

[70] N. Lawrence. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *The Journal of Machine Learning Research*, 6:1783–1816, 2005. → pages 13

[71] J. P. Lewis, M. McGuire, and P. Fox. Mapping the mental space of game genres. In *Proc. ACM SIGGRAPH Symp. Video Games*, pages 103–108, 2007. → pages 95, 108

[72] Y. Lifshits. Algorithms for nearest neighbor search. Tutorial, Russian Summer School in Information Retrieval (RuSSIR), 2007. → pages 57

[73] Y. Liu, S. Barlowe, Y. Feng, J. Yang, and M. Jiang. Evaluating exploratory visualization systems: A user study on how clustering-based visualization systems support information seeking from large document collections. *Information Visualization*, 12(1):25–43, 2013. → pages 47

[74] H. Luhn. A statistical approach to the mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4): 309–317, 1957. → pages 82

[75] C. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008. → pages 15, 46, 82

[76] W. Matusik, H. Pfister, M. Brand, and L. McMillan. A data-driven reflectance model. *ACM Trans. Graphics (Proc. SIGGRAPH 2003)*, 22(3): 759–769, 2003. → pages 5, 107

[77] L. Molina, L. Belanche, and À. Nebot. Feature selection algorithms: A survey and experimental evaluation. In *Proceedings 2002 IEEE International Conference on Data Mining*, pages 306–313. IEEE, 2002. → pages 4

[78] T. Munzner, A. Barsky, and M. Williams. Reflections on questvis: A visualization system for an environmental sustainability model. *Scientific Visualization: Interactions, Features, Metaphors*, 2:240–259, 2011. → pages 36

[79] K. Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, 2012. → pages 3

[80] F. Murtagh. A very fast, exact nearest neighbor algorithm for use in information retrieval. *Information Technology: Research and Development*, 1(4):275–283, 1982. → pages 16

[81] F. Murtagh. Clustering in massive data sets. In *Handbook of Massive Data Sets*, pages 501–543. Kluwer Academic Publishers, 1999. → pages 17, 119

[82] P. Öesterling, G. Scheuermann, S. Teresniak, G. Heyer, S. Koch, T. Ertl, and G. Weber. Two-stage framework for a topology-based projection and visualization of classified document collections. In *Proc. IEEE Symp. Visual Analytics Science and Technology (VAST)*, pages 91–98, 2010. → pages 20

[83] F. Paulovich, C. Silva, and L. Nonato. Two-phase mapping for projecting massive data sets. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1281–1290, 2010. → pages 13, 75

[84] S. Perry and P. Willett. A review of the use of inverted files for best match searching in information retrieval systems. *Journal of Information Science*, 6(2-3):59–66, 1983. → pages 55

[85] J. Platt. FastMap, MetricMap, and Landmark MDS are all Nyström algorithms. In *Proc. Intl. Workshop on Artificial Intelligence and Statistics*, pages 261–268, 2005. → pages 11

[86] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL http://www.R-project.org/. Accessed: 2013-09-27. → pages 44

[87] R Development Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2008. URL http://www.R-project.org. → pages 18, 19

[88] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. → pages 105, 106

[89] S. Robertson. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520, 2004. → pages 51

[90] G. Ross and M. Chalmers. A visual workspace for hybrid multidimensional scaling algorithms. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pages 91–96, 2003. → pages 18

[91] L. Rousseeuw and L. Kaufman. Clustering by means of medoids. *Statistical Data Analysis Based on the L1-norm and Related Methods*, pages 405–416, 1987. → pages 116

[92] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000. → pages 13

[93] Y. Rubner, C. Tomasi, and L. Guibas. The Earth Mover's Distance as a metric for image retrieval. *Intl. Journ. Computer Vision*, 40(2):99–121, 2000. → pages 95, 107

[94] D. Russell, M. Stefik, P. Pirolli, and S. Card. The cost structure of sensemaking. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 269–276. ACM, 1993. → pages 82

[95] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988. → pages 15, 50, 51

[96] G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975. → pages 15, 50, 82, 84

[97] J. Sammon. A nonlinear mapping for data structure analysis. *IEEE. Trans. Computers*, 100(5):401–409, 1969. → pages 71

[98] M. Sedlmair, M. Brehmer, S. Ingram, and T. Munzner. Dimensionality reduction in the wild: Gaps and guidance. Technical Report TR-2012-03, UBC Computer Science, June 2012. → pages 4, 14, 43, 121

[99] M. Sedlmair, M. Meyer, and T. Munzner. Design study methodology: Reflections from the trenches and the stacks. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2431–2440, 2012. → pages 120

[100] J. Seo and B. Shneiderman. A rank-by-feature framework for interactive exploration of multidimensional data. *Information Visualization*, 4(2): 99–113, 2005. → pages 19, 115

[101] F. Shahnaz, M. Berry, V. Pauca, and R. Plemmons. Document clustering using nonnegative matrix factorization. *Information Processing & Management*, 42(2):373–386, 2006. → pages 6

[102] S. Shapiro and M. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965. → pages 105

[103] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973. → pages 17

[104] V. Silva and J. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. *Advances in Neural Information Processing Systems*, 15:705–712, 2003. → pages 13

[105] A. Smeaton and C. Van Rijsbergen. The nearest neighbour problem in information retrieval: an algorithm using upperbounds. *ACM Special Interest Group on Information Retrieval (SIGIR) Forum*, 16(1):83–87, 1981. → pages 16

[106] P. Sneath. The application of computers to taxonomy. *Journal of General Microbiology*, 17(1):201–226, 1957. → pages 17

[107] I. Spence and D. Domoney. Single subject incomplete designs for nonmetric multidimensional scaling. *Psychometrika*, 39:469–490, 1974. → pages 96

[108] M. Stone. Color in information display. IEEE Visualization 2006 Course Notes. `http://www.stonesc.com/Vis06`, Oct 2006. → pages 32

[109] J. Stray. How overview can organize thousands of documents for a reporter. http://overview.ap.org/blog/2013/04/how-overview-can-organize-thousands-of-documents-for-a-reporter/. Accessed: 2013-06-04. → pages 52

[110] J. Tenenbaum, V. de Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, Dec 22 2000. → pages 13

[111] D. Ternes and K. MacLean. Designing large sets of haptic icons with rhythm. In *Intl. Conf. Haptics: Perception, Devices, and Scenarios (EuroHaptics)*, pages 199–208. Springer LNCS 5024, 2008. → pages 95

[112] The MathWorks Inc. *MATLAB*. Natick, Massachusetts, 2010. → pages 18

[113] W. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17:401–419, 1952. → pages 11

[114] M. Tory, D. Sprague, F. Wu, W. So, and T. Munzner. Spatialization design: Comparing points and landscapes. *Proc. IEEE Symp. Information Visualization (InfoVis)*, 13(6):1262–1269, 2007. → pages 20

[115] M. Tory, C. Swindells, and R. Dreezer. Comparing dot and landscape spatializations for visual memory differences. *Proc. IEEE Symp. Information Visualization (InfoVis)*, 16(6):1033–1040, 2009. → pages 20

[116] L. van der Maaten. Barnes-Hut-SNE. In *Proceedings of the International Conference on Learning Representations*, 2013. URL http://arxiv.org/abs/1301.3342. → pages 14, 15, 72, 73

[117] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008. → pages 14, 72, 73

[118] L. van der Maaten, E. Postma, and H. Van Den Herik. Dimensionality reduction: A comparative review. *Journal of Machine Learning Research*, 10:1–41, 2009. → pages 66, 71

[119] E. Voorhees. Implementing agglomerative hierarchic clustering algorithms for use in document retrieval. *Information Processing & Management*, 22 (6):465–476, 1986. → pages 17, 67

[120] J. Wade. TPD working through flawed mobile system. http://www.tulsaworld.com/article.aspx/TPD_working_through_flawed_mobile_system/20120603_11_a1_cutlin136616. Accessed: 2013-06-04. → pages 90

[121] K. Wagstaff. Machine learning that matters. *Proc. Intl. Conf Machine Learning (ICML)*, pages 529–536, 2012. → pages 120

[122] J. Wang, D. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):283–298, 2008. → pages 14

[123] M. Ward. XmdvTool: Integrating multiple methods for visualizing multivariate data. In *Proc. IEEE Visualization*, pages 326–333, 1994. → pages 18

[124] C. Weaver. Cross-filtered views for multidimensional visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):192–204, 2010. → pages 81

[125] K. Weinberger and L. Saul. An introduction to nonlinear dimensionality reduction by maximum variance unfolding. In *Proceedings of the National*

Conference on Artificial Intelligence, volume 21, pages 1683–1686. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006. → pages 13

[126] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer New York, 2009. → pages 18

[127] J. Wise, J. Thomas, K. Pennock, D. Lantrip, M. Pottier, A. Schur, and V. Crow. Visualizing the non-visual: spatial analysis and interaction with information from text documents. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pages 51–58, 1995. → pages 79

[128] D. Wishart. Mode analysis: A generalization of nearest neighbor which reduces chaining effects. *Numerical Taxonomy*, 76:282–311, 1969. → pages 17

[129] C. Xiao, W. Wang, X. Lin, and H. Shang. Top-k set similarity joins. In *Conf. on Data Engineering (ICDE)*, pages 916–927. IEEE, 2009. → pages 16

[130] J. Yang, W. Peng, M. Ward, and E. Rundensteiner. Interactive hierarchical dimension ordering, spacing and filtering for exploration of high dimensional datasets. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pages 105–112, 2003. → pages 18

[131] J. Yang, M. Ward, E. Rundensteiner, and S. Huang. Visual hierarchical dimension reduction for exploration of high dimensional datasets. In *Proc. Eurographics/IEEE Symp. Visualization (VisSym)*, pages 19–28, 2003. → pages 18

[132] J. Yang, A. Patro, S. Huang, N. Mehta, M. Ward, and E. Rundensteiner. Value and relation display for interactive exploration of high dimensional datasets. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pages 73–80, 2004. → pages 18

[133] J. Yang, D. Luo, and Y. Liu. Newdle: Interactive visual exploration of large online news collections. *IEEE Computer Graphics & Applications*, 30(5): 32–41, 2010. → pages 19, 47

[134] Z. Yang, J. Peltonen, and S. Kaski. Scalable optimization of neighbor embedding for visualization. In *Proc. Intl. Conf Machine Learning (ICML)*, pages 127–135, 2013. → pages 14, 15, 72

[135] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 311–321. Society for Industrial and Applied Mathematics, 1993. → pages 16, 64, 73

[136] H. Zha. Generic summarization and keyphrase extraction using mutual reinforcement principle and sentence clustering. In *Proc. ACM Conf. Information Retrieval (SIGIR)*, pages 113–120. ACM, 2002. → pages 83

[137] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys (CSUR)*, 38(2):6, 2006. → pages 15, 47, 55, 58, 119