SIGRAD 2012
A. Kerren and S. Seipel (Editors)

Glint: An MDS Framework for Costly Distance Functions

S. Ingram[†1] and T. Munzner[1]

[1]University of British Columbia, Canada

Abstract

Previous algorithms for multidimensional scaling, or MDS, aim for scalable performance as the number of points to lay out increases. However, they either assume that the distance function is cheap to compute, and perform poorly when the distance function is costly, or they leave the precise number of distances to compute as a manual tuning parameter. We present Glint, an MDS algorithm framework that addresses both of these shortcomings. Glint is designed to automatically minimize the total number of distances computed by progressively computing a more and more densely sampled approximation of the distance matrix. We present instantiations of the Glint framework on three different classes of MDS algorithms: force-directed, analytic, and gradient-based. We validate the framework through computational benchmarks on several real-world datasets, and demonstrate substantial performance benefits without sacrificing layout quality.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Human-centered Computing]: Visualization—Visualization systems and tools

1. Introduction

Multidimensional Scaling, or MDS, is a method for positioning the points of a dataset into a user-specified, low-dimensional space. The technique is used when the given description of the points is overly verbose, making visual analysis unwieldy or algorithmic analysis intractable. Input dataset descriptions processed by MDS come in two types: **points**, where each point is described by an equal number of spatial coordinates, or a **distance matrix**, where the rows and columns of the matrix represent a nonnegative value computed by a distance function of the two points.

Plotting the low-dimensional MDS output enables visual analysis of the proximity relationships between data points that are obscured in high-dimensions. This technique is employed in psychophysics, marketing research, and unsupervised learning [BG05, Gre75, HTF09]. MDS visualizations can be readily incorporated into other interactive applications [IMS12], providing a rich overview of the data.

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 contribution of this paper 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 [IMO09]. Many real-world problems where MDS is used require more costly distance functions than the

† E-mail: {sfingram,tmm}@cs.ubc.ca

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 [IMO09], or leave the total number of distances to compute as a tuning parameter and so do not have a fully automatic way to terminate [BP07, dST04]. The Glimmer algorithm is an example of the overcomputation shortcoming [IMO09]. 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 [BP07]. 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, our contribution includes 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 show that these Glint instantiations drastically reduce total run time on datasets with costly distance functions without penalizing the final layout quality.

2. 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.

2.1. 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 [RTG00]. 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 [LMF07]; in a psychophysics example, distances are derived from just noticeable differences in haptic stimuli [TM08].

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.

2.2. 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 [SD74]. 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, are given in practice. 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.

3. Related Work

MDS refers to an entire family of algorithms with different objective functions, computational complexities, and qualitative results [BG05, FC11]. 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. Here, we discuss the four major classes of MDS algorithms in terms of their shortcomings in handling costly distances.

3.1. Coordinate-Based Algorithms

MDS input can take the form of a table of coordinates or a distance matrix. When the points are given as coordinates, the number of input dimensions m is often much smaller than the number of points N. The PLMP [PSN10] and LAMP [JCC*11] algorithms build on this assumption to rapidly compute low-distortion layouts for very large datasets. The profound acceleration that the algorithms achieve is hindered when the number of dimensions equals or exceeds the number of points, as is precisely the case when the input format is a distance matrix. Because Glint is designed for the distance matrix use case, coordinate-based algorithms are not suitable as components for Glint.

3.2. Analytic Algorithms

The original MDS algorithm, now called Classic MDS [Tor52], computed a one-step analytic minimum of an objective function called Strain. The algorithm relies on computing the full SVD of a dense N^2 matrix, 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 [Pla05]. For example, both Pivot MDS [BP07] and Landmark MDS [dST04] 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.

3.3. Force-Directed Algorithms

The Glimmer [Ing07, IMO09] algorithm and its antecedent, by Chalmers [Cha96], are MDS approximation algorithms 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.

While force-directed algorithms typically exhibit a rapid convergence to a minimum, they often suffer from computing 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. As we show later in the paper, force-directed algorithms can sample fewer high-dimensional distances and suffer little to no degradation in quality.

3.4. 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 [BSL*08], while the SMACOF algorithm [dLM09] minimizes a sequence of majorizing quadratic functions. These techniques are more 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 [GKN04, KHKS12], gradient techniques can harness a sparsely populated distance matrix as input with good results. 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.

4. Glint Algorithm Framework

Glint is an algorithm framework: an algorithm with modular components that are themselves algorithms. Figure 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

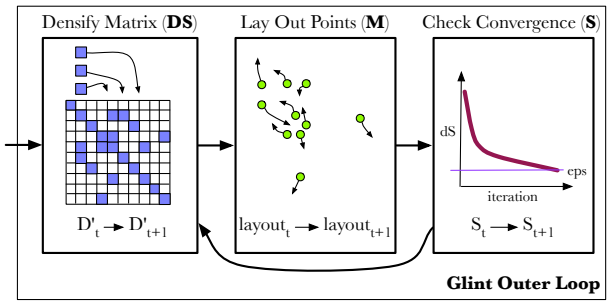

Figure 1: *Diagram of Glint execution.*

sparse distance matrix. To qualify for use in Glint, **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 5.1.

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 5.2.

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 5.3.

4.1. Glint Outer Loop

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,

Algorithm 1 Pseudocode for Glint, with variable definitions.

function GLINT(ε)

layout ← RANDOMLAYOUT

while !converged **do**

$P_{t+1} \leftarrow \mathrm{DS}(P_t)$
$D'_{t+1} \leftarrow \mathrm{DISTANCE}(D'_t, P_{t+1}, d)$
$\mathrm{layout}_{t+1} \leftarrow \mathrm{M}(D'_{t+1}, \mathrm{layout}_t)$
$S_{new} \leftarrow \mathrm{S}(P_t, D'_{t+1}, \mathrm{layout}_{t+1})$
$\mathrm{converged} \leftarrow |S_{old} - S_{new}|/S_{old} < \varepsilon$
$S_{old} \leftarrow \mathrm{S}(P_{t+1}, D'_{t+1}, \mathrm{layout}_{t+1})$

return layout

Variable	Description
P_t	the set of computed point pairs at iteration t
D'_t	the sparse distance matrix with nonzeros specified by P_t
S_t	scalar objective function value at iteration t
d	the distance function
t	the current Glint iteration
ε	termination threshold

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 is 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 Figure 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 $t+1$.

5. 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 3: force-directed, analytic, and gradient.

Several of the Glint instantiations require choosing input parameters, as discussed in detail below. Table 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.

5.1. 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 [BP07] for **M**, and for the gradient implementation we substituted the SMACOF algorithm [dLM09] 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.

5.1.1. 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 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 high leads to computing more distances than are necessary. We select 100 as a good balance over all our benchmarks between these two extremes.

5.1.2. Force-Directed Instantiation

In the force-directed instantiation, we substitute a modified version of the Glimmer [Ing07,IMO09] algorithm for **M**. We used the version of Glimmer that supports distance matrix calculations in addition to handling points [Ing07]. 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'.

5.2. 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.

5.2.1. 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.

5.2.2. 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 [IMO09]. 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.

5.2.3. 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 [BP07] starting from a single column chosen uniformly at random.

5.3. 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

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 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.*

the Glint instantiations, **S** fits the following template:

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

Here, $hi(i,j)$ 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. The *sel* function is an index-selection function that selects a subset from 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 original Pivot MDS paper for more details on the computation of C [BP07].

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. In the force-directed instantiation, **S** is equal to the sparse stress function computed at the end of each Glimmer run [IMO09].

Each instantiation employs randomized sampling of new distance matrix indices after each Glint iteration, as mentioned in Section 5.2. In the case of the gradient-based instantiation, this random sampling does not impart enough random noise to the observed values of **S** to induce an unexpected termination. However, in the force-directed and analytic cases, we observed enough noise in the sequence of **S** values that early termination was regularly observed. In this section we describe our strategy for creating a smooth **S** from the noisy series of raw objective function values.

A simple approach would be to filter the sequence of raw values using a moving average. Since the noise in the signal is white noise, with equal power across all frequencies, it would manifest itself after filtering any bandwidth, so this approach would not solve the problem.

Fortunately, the observed noise can be modelled by a Gaussian distribution. 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) [RW06]. (We confirmed normality with a Shapiro-Wilk test result of $p = 0.55$ [SW65].)

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 [RW06], 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, for each 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 an-

alytic 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.

5.4. Instantiation Design Summary

Table 3 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, such as Pivot MDS and Glimmer, can employ GPR smoothing.

6. 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.

6.1. 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 first author's public photo collection, with distances computed using the Earth Mover's Distance (EMD) [RTG00]. 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 [MPBM03]; this function is expensive despite being Euclidean because of the huge number of dimensions. The videogame dataset was created by gathering human judgements in response to survey questions about 96 games. While the exact timing information for the judgments was not reported [LMF07], 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 2 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 2: *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.*

6.2. 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 [BG05]. 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 handle incomplete distance matrix columns, so we omit the videogame benchmark, which possesses many missing matrix entries, from the analytic results.

Figure 2 and Table 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

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 3: *Glint component design summary for each MDS algorithm class.*

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

`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.

6.3. Convergence

We illustrate the convergence behavior of each Glint instantiation in Figure 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.

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.

7. Conclusion

We have illustrated how expensive distance calculations change the efficiency of existing MDS algorithms. Algorithms like Glimmer and SMACOF 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 show how careful design of termination criteria can overcome the noise effect of random sampling on convergence. We present and validate 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 provide 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 func-

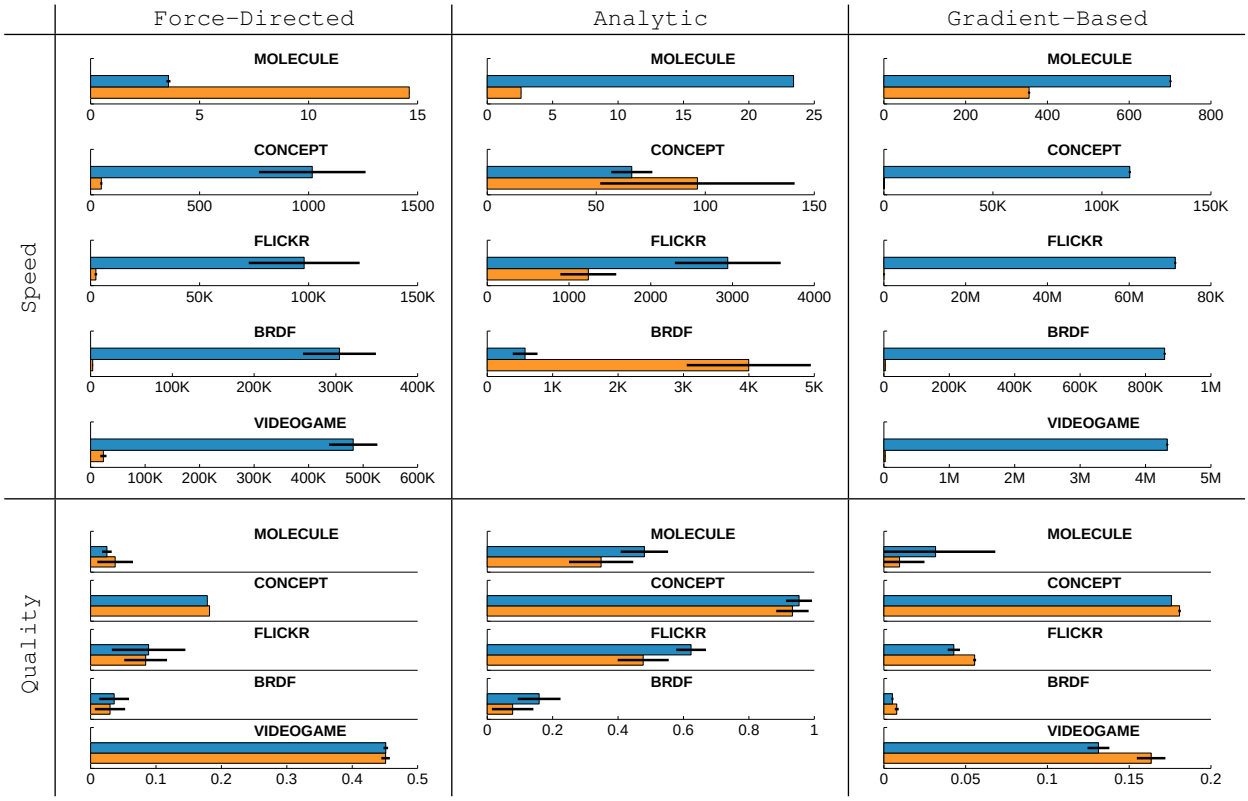

Figure 2: *Comparison of speed (top) and quality (bottom). In each pair, the top blue bar is the original MDS algorithm, and the bottom orange bar is the Glint instantiations. The black lines indicate 95% standard error bars.*

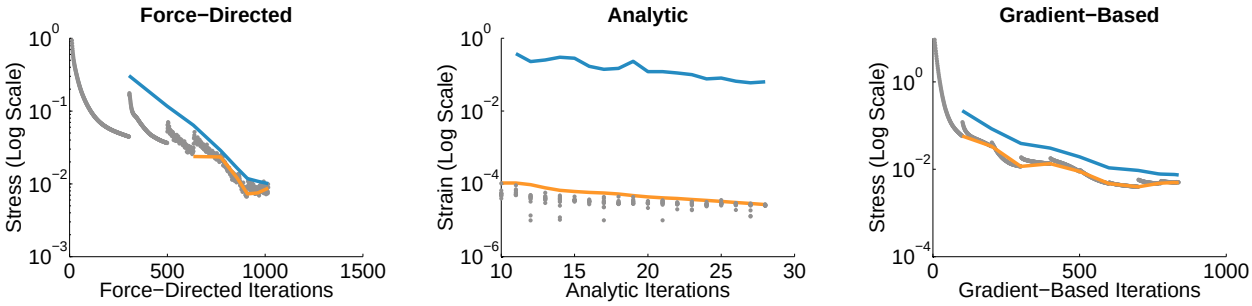

Figure 3: *Log-scale Glint convergence curves on each instantiation generated using the* `brdf` *dataset. The orange* ***S*** *curve is derived from the noisy grey samples.* ***S*** *is designed to match the convergence behavior of the costly* ***F*** *series in blue.*

tions, ranging from 20 to 115 with the force-directed Glint instantiation and from 200 to 8800 with the gradient-based Glint instantiation.

The Glint framework uses modified versions of existing algorithms to more efficiently compute low-dimensional layouts on problems with costly distance functions. Glint reduces the total time spent computing distance information by automatically selecting a reduced number of distance matrix entries to compute based on monitoring layout quality. Glint reduces the time and cost of analyzing distance-based datasets with MDS, opening the door for practitioners to apply MDS to problems with expensive distance functions at an entirely new scale.

8. Acknowledgements

This work was supported through a NSERC Strategic Grant, with partial travel support from UBC ICICS and FOGS.

References

[BG05] BORG I., GROENEN P. J. F.: *Modern Multidimensional Scaling Theory and Applications*, 2nd ed. Springer-Verlag, 2005. 1, 3, 7

[BP07] BRANDES U., PICH C.: Eigensolver methods for progressive multidimensional scaling of large data. In *Graph Drawing*, Kaufmann M., Wagner D., (Eds.), vol. 4372 of *Lecture Notes in Computer Science*. Springer, 2007, pp. 42–53. 2, 3, 5, 6

[BSL*08] BUJA A., SWAYNE D., LITTMAN M., DEAN N., HOFMANN H., CHEN L.: Data visualization with multidimensional scaling. *Journal of Computational and Graphical Statistics 17*, 2 (2008), 444–472. 3

[Cha96] CHALMERS M.: A linear iteration time layout algorithm for visualising high dimensional data. In *Proc. IEEE Visualization* (1996), pp. 127–132. 3

[dLM09] DE LEEUW J., MAIR P.: Multidimensional scaling using majorization: SMACOF in R. *Journ. Statistical Software 31*, 3 (8 2009), 1–30. 3, 5

[dST04] DE SILVA V., TENENBAUM J.: *Sparse multidimensional scaling using landmark points*. Technical report, Stanford, 2004. 2, 3

[FC11] FRANCE S., CARROLL J.: Two-way multidimensional scaling: A review. *IEEE Trans. Systems, Man, and Cybernetics, Part C: Applications and Reviews 41*, 5 (2011), 644–661. 3

[GKN04] GANSNER E. R., KOREN Y., NORTH S. C.: Graph drawing by stress majorization. In *Graph Drawing* (2004), pp. 239–250. 3

[Gre75] GREEN P.: Marketing applications of mds: Assessment and outlook. *The Journal of Marketing* (1975), 24–31. 1

[HTF09] HASTIE T., TIBSHIRANI R., FRIEDMAN J.: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer, 2009. 1

[IMO09] INGRAM S., MUNZNER T., OLANO M.: Glimmer: Multilevel MDS on the GPU. *IEEE Trans. Visualization and Computer Graphics (TVCG) 15*, 2 (2009), 249–261. 1, 2, 3, 5, 6

[IMS12] INGRAM S., MUNZNER T., STRAY J.: *Hierarchical Clustering and Tagging of Mostly Disconnected Data*. Tech. Rep. TR-2012-01, University of British Columbia Department of Computer Science, May 2012. 1

[Ing07] INGRAM S.: *Multilevel Multidimensional Scaling on the GPU*. Master's thesis, University of British Columbia Department of Computer Science, 2007. 3, 5

[JCC*11] JOIA P., COIMBRA D., CUMINATO J. A., PAULOVICH F. V., NONATO L. G.: Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2563–2571. 3

[KHKS12] KHOURY M., HU Y., KRISHNAN S., SCHEIDEGGER C.: Drawing large graphs by low-rank stress majorization. *Comp. Graph. Forum 31*, 3pt1 (June 2012), 975–984. 3

[LMF07] LEWIS J. P., MCGUIRE M., FOX P.: Mapping the mental space of game genres. In *Proc. ACM SIGGRAPH Symp. Video Games* (2007), pp. 103–108. 2, 7

[MPBM03] MATUSIK W., PFISTER H., BRAND M., MCMILLAN L.: A data-driven reflectance model. *ACM Trans. Graphics (Proc. SIGGRAPH 2003) 22*, 3 (2003), 759–769. 7

[Pla05] PLATT J. C.: FastMap, MetricMap, and Landmark MDS are all Nyström algorithms. In *Proc. Intl. Workshop on Artificial Intelligence and Statistics* (2005), pp. 261–268. 3

[PSN10] PAULOVICH F., SILVA C., NONATO L.: Two-phase mapping for projecting massive data sets. *IEEE Trans. Visualization and Computer Graphics 16*, 6 (2010), 1281–1290. 3

[RTG00] RUBNER Y., TOMASI C., GUIBAS L.: The Earth Mover's Distance as a metric for image retrieval. *Intl. Journ. Computer Vision 40*, 2 (2000), 99–121. 2, 7

[RW06] RASMUSSEN C. E., WILLIAMS C. K. I.: *Gaussian Processes for Machine Learning*. MIT Press, 2006. 6

[SD74] SPENCE I., DOMONEY D.: Single subject incomplete designs for nonmetric multidimensional scaling. *Psychometrika 39* (1974), 469–490. 2

[SW65] SHAPIRO S., WILK M.: An analysis of variance test for normality (complete samples). *Biometrika 52*, 3/4 (1965), 591–611. 6

[TM08] TERNES D., MACLEAN K. E.: Designing large sets of haptic icons with rhythm. In *Intl. Conf. Haptics: Perception, Devices, and Scenarios (EuroHaptics)* (2008), Springer LNCS 5024, pp. 199–208. 2

[Tor52] TORGERSON W.: Multidimensional scaling: I. theory and method. *Psychometrika 17* (1952), 401–419. 3