Steerable, Progressive Multidimensional Scaling

Matt Williams[*] Tamara Munzner[†]

University of British Columbia

Abstract

Current implementations of Multidimensional Scaling (MDS), an approach that attempts to best represent data point similarity in a low-dimensional representation, are not suited for many of today's large-scale datasets. We propose an extension to the spring model approach that allows the user to interactively explore datasets that are far beyond the scale of previous implementations of MDS.

We present MDSteer, a steerable MDS computation engine and visualization tool that progressively computes an MDS layout and handles datasets of over one million points. Our technique employs hierarchical data structures and progressive layouts to allow the user to steer the computation of the algorithm to the interesting areas of the dataset. The algorithm iteratively alternates between a layout stage in which a sub-selection of points are added to the set of active points affected by the MDS iteration, and a binning stage which increases the depth of the bin hierarchy and organizes the currently unplaced points into separate spatial regions. This binning strategy allows the user to select onscreen regions of the layout to focus the MDS computation into the areas of the dataset that are assigned to the selected bins. We show both real and common synthetic benchmark datasets with dimensionalities ranging from 3 to 300 and cardinalities of over one million points.

CR Categories: I.5.3 [Pattern recognition]: Clustering—Algorithms E.1 [Data Structures]: Graphs and networks I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

Keywords: dimensionality reduction, multidimensional scaling

1 Introduction

Dimensionality reduction techniques allow a dataset of high-dimensional points to be explored by projection into low-dimensional spaces such as the 2D plane or 3D space. Multidimensional scaling, or MDS, has been one of the most popular approaches to reducing dimensionality since its introduction by Torgerson into the psychological literature fifty years ago [18] as a way to represent perceived similarities between a pair of stimuli. MDS is a technique where the ratio of differences between inter-point distances in the original high-dimensional space and in the projected low-dimensional space are minimized.

Dimensionality reduction techniques have been published in many fields: psychology [9, 18], cartography [7], machine learning [16, 17], and information visualization [1, 4, 11, 12, 13]. Error minimization requires many computationally expensive high-dimensional distance or matrix computations, and the challenge is reducing the cost and number of these calculations. Torgerson's early approach had a cost of $O(n^3)$, where n is the number of points in the dataset [18]. Methods with an $O(n^2)$ cost that can handle thousands of points have become common [1, 4, 9, 10]. Recently, a subquadratic algorithm was proposed by Morrison that could lay out thousands of points in minutes [11, 12, 13].

Despite the extensive previous work, there is a gap in the literature: no currently available algorithm or system allows interactive exploration of high-dimensional datasets with both a large number of dimensions and a large number of points. Although Morrison does handle hundreds of thousands of points in minutes [11, 12], that is only true when the number of dimensions is low. On our real-world dataset of 120,000 nodes and 294 dimensions, the HIVE system [15] took over 2 hours to compute the layout.

We present MDSteer, a steerable system that allows the user to progressively guide the MDS layout process so that exploration of huge datasets can begin immediately after startup. The user can interactively select local regions of interest, and then most of the available computational resources are spent on refining this selected area of interest. Users can immediately begin exploring datasets of over 1 million nodes. The overall dataset structure is apparent in a few seconds, providing an overview that helps users find potentially interesting areas in the projection. After interactive drill-down to a small local area, computational resources are steered to that location to quickly fill in that area. Again, even for datasets of over one million points, these small local areas can be fully populated within minutes. Our system handles datasets with dimensionality of several hundred and cardinality of over one million.

The ability to immediately explore and steer computational resources to regions of interest allows investigation of datasets an order of magnitude larger than previous work.

2 Previous Work

The term MDS has been used to refer to any general approach that attempts to reproduce similarity data in lower dimensions. The actual computational techniques vary from eigensolver approaches to iteratively refining spring models of the type used in force-directed graph layout [6, 8]. The basic spring model MDS approach [4] iteratively calculates a low-dimensional displacement vector for each point to minimize the difference between the low-dimensional and high-dimensional distance. Since every iteration requires each point to be compared with all other points in the dataset, the iteration complexity is $O(n^2)$. We chose to add steerability and progressive layout to the spring-model algorithm, using a publicly available software platform that handles high-cardinality datasets [15].

Our steerable technique builds on an algorithm proposed by Chalmers [4] that has a linear cost for each iteration. Instead of allowing forces on a point from every other point in the dataset, the position of a point was determined by interacting with a two small sets of points that each contained a constant number of items. Each point p maintained a list of V *neighborhood points* that persisted across spring-model iterations, and S *randomly sampled* points that were resampled at each iteration. At the beginning, the neighborhood was populated randomly from all points in the dataset. The neighborhood quality improved over iterations because a new random sample would force out the most distant neighborhood point, if it was closer. Although the per-iteration cost was linear in n, the total number of iterations required for this approach depended on the dataset size, so overall cost of this approach was $O(n^2)$. The

[*]e-mail: mwill@cs.ubc.ca
[†]e-mail: tmm@cs.ubc.ca

paper reports good results with $V = 10$ and $S = 5$, and the implementation of this algorithm distributed in the HIVE system [15] uses $V = 6$ and $S = 3$. We use the latter.

In 2002, Morrison improved on this result with an efficient 3-step approach: a initial base layout, interpolation, and final refinement layout [12]. The Chalmers [4] algorithm was used to lay out an initial $\sqrt{n}$ sample of points. That initial layout was followed by an interpolation stage that used the location of the sample points to find a good initial position for all remaining unplaced points. The final stage ran several MDS iterations on the entire dataset, refining the approximate initial placements into better final positions. We take this idea of breaking the work into discrete stages much further: we carry out up to $n/(\sqrt{n}/k)$ layout steps, where k is a tuneable parameter, adding a small number of new points at each step.

In Morrison's 2002 work the interpolation stage was the most expensive, with a $O(n * \sqrt{(n)})$ cost, since it compared each of the $n - \sqrt{(n)}$ unplaced points with the $\sqrt{(n)}$ placed points in order to find the initial layout location for those unplaced points. In 2003 Morrison [11, 13] improved the performance of computing starting spots for unplaced points by applying an efficient nearest-neighbor search technique at the interpolation stage. The hierarchical binning approach we developed to provide steerability also allows us to find a good starting location for unplaced points efficiently, so we present an alternate approach.

The work of Basalaj on incremental MDS [2, 3] is perhaps the most similar to our own. While they ignore local detail to focus on overall shape, we take the opposite approach, instead encouraging people to build up local detail in areas of interest. Basalaj has one of the very few systems that handles large datasets of over 100,000 nodes.

MDS is used in visualization systems both in the straightforward way of showing points in the plane [1] and in more indirect ways. For example, using MDS for a projection onto the plane, and then assigning an additional value for a third coordinate, results in a terrain view of the dataset [5, 19].

3 Steerable, Progressive MDS

The two techniques we use to introduce steerability into MDS are progressive layout of points and hierarchical binning. Steering allows computational power to be focused where it is needed to support exploration in parallel with continuing the layout process.

3.1 Algorithm

The MDSteer algorithm alternates layout with binning computation. At each layout step, we add $\sqrt{n}/k$ new points to the computation, find an initial position for each of these new points, and run MDS iterations on on the active set of points until the layout stops improving. Every k layout steps, we rebin all of the points. Algorithm 1 presents the details in pseudocode.

Algorithm 1 MDSteer algorithm

```
sampleSize = $\sqrt{n}/k$;
**while** allSelectedBins.hasUnplacedPoints **do**
    **for all** $b \in$ selectedBins **do**
        **for** [1, sampleSize / allSelectedBins.size] **do**
            $p = b$.getRandomPoint();
            activePointSet.add($p$);
            $p$.startLocation=$b$.nearestPlacedPoint.location;
        **end for**
    **end for**
    **while** stressIsShrinking() **do**
        **for** [1 , $\sqrt{sampleSize}$] **do**
            **for all** $p \in$ activePointSet **do**
                **for all** $q \in p$.comparisonSet **do**
                    doMDSAdjustment($p$,$q$);
                **end for**
            **end for**
            stressCalculate(activePointSet, placedPointSet);
        **end for**
    **end while**
    growBinHierarchyOneLevelDeeper();
    rebinAllPoints();
**end while**
```

3.2 Bins

We subdivide the low-dimensional plane into a hierarchical decomposition of rectangular screen-space regions that we call **bins**. Bins are drawn onscreen as wireframe boxes, which are highlighted when selected by the user. Selecting one or many of these regions causes the available computational resources to be focused on "filling in" those bins; that is, laying out the higher-dimensional points that are likely to be projected to that region of the lower-dimensional plane. The hierarchy of bins guides the MDS layout at several levels. First, we restrict the amount of work we do at each MDS iteration by allowing only the selected bins to have active points that move around. Second, the set of new points to activate is chosen only from selected bins. Third, bin membership is used to efficiently find starting positions for those new points that have just become active.

We start the computation with only a single bin which contains all the points. That initial bin is both the root of the bin hierarchy and the singleton leaf. At every rebinning pass, we increase the maximum depth of the bin hierarchy by one, subdividing each current leaf bin into two new child bins, so that the former leaf is now an interior node and the new children are now the leaves. The leaf subdivision is subject to a validity constraint that we explain below. After subdivision, the new bins subtend a smaller region of the plane.

Steering With Bins Bins serve as a mechanism for the user to select a subset of the data as the target of the available computational resources. Every point in the dataset is assigned to some bin. We categorize points into one of three states: unplaced, active, and placed. Unplaced points are not drawn, nor do they affect any MDS iteration, and all points are unplaced at the beginning of the computation. At each layout step we convert a set of s new points from unplaced to active, where $s = \sqrt{n}/k$, n is the total number of points in the dataset and k is a tuneable parameter. We draw our samples evenly from all active bins.

When we activate unplaced points, we need to find their initial locations in the plane before placements can be iteratively refined. We use the positions of the placed points as initial locations in the plane for the new points. Using these existing placements allows the new points to benefit from the previous computation. We can find the initial placement efficiently by using the binning to narrow down the possibilities: we check the distance from our target point to all placed or active points in the bin, and pick the closest high-dimensional neighbor in the bin.

The heart of the layout computation is the inner loop where multiple iterations of the spring-force MDS algorithm [4] are run, and only active points are the targets of this computation. Thus, they are the only points that visibly move around as their projection onto the plane changes. We check every $\sqrt{sampleSize}$ iterations to see if the layout is still improving, and terminate the layout step when progress is no longer being made. The inner loop termination criteria are discussed in more detail in Section 3.3.

When a bin is unselected, all the active points are placed; their positions are fixed and do not move around during successive MDS iterations. However, these placed points can affect movement of other active points during the MDS iterations because they are potential candidates for the random sample set. Both placed and active points are always visible to the user.

Increasing Bin Hierarchy Depth After k layout steps where a total of $\sqrt{n}$ new points have been laid out, we need to increase the bin hierarchy depth by one, then rebin all points. We first describe how to grow the hierarchy.

We do not store points at any interior node of the bin tree, only the leaves. Projected points may move outside the spatial boundary of their previous leaf bin during layout. We need to reassign these points so that they belong to the bins in which they lie before subdividing the current set of leaf bins to create the next layer of the bin hierarchy. We check the active set of points to see if any are out of bounds. For each such point that we find, we traverse upwards in the tree to find the first ancestor node that can spatially contain it. When we perform a top-down traversal of the tree in order to subdivide bins, then we also push these out-of-bounds points back down to the correct bin at the leaves.

We do not always subdivide a leaf bin. For subdivision to occur, a bin must have some minimum number of active points, and it must contain at least one unplaced point. We use the minimum threshold of 10, which we found empirically.

We alternate subdividing bins in the vertical and horizontal directions at each rebinning pass. To subdivide a bin, we find the placed or active points with minimum and maximum values in whichever direction we are currently using, and place the new dividing line halfway between these to create two new child bins. The minimum and maximum points we just found are each attached to the child bin in which they now fall, and we call these points the **representative points** and use them for the high-dimensional distance computations described below for rebinning points. These representative points can change from pass to pass, which gives rise to the irregular subdivision we see in Figure 5. The average number of items per bin decreases over the course of the progressive layout: after r rebinning passes, the average binsize is $(r * \sqrt{n})/2^r$.

When a selected bin is subdivided, both of its new child bins are selected. When the program starts, the single existing bin is selected. If the user never makes a steering choice by explicitly changing the selection state, then the computational resources are equally divided between all areas of the screen and our algorithm would simply do a progressive layout.

Rebinning Points Rebinning points is done immediately after subdividing a parent bin into two child bins, so for each point currently assigned to the parent bin the only choice to make is which of the two child bins to pick. The decision is easy for active and placed points because they already have projected 2D coordinates in the plane. We need only to check a single one of coordinates, whichever direction is the current active one, to find on which side of the dividing line the point currently lies. Assigning the unplaced points requires more computation. For each unplaced point, we compute the high-dimensional distances between it and the representative point for each child bin. We assign the point to the bin that has the closer representative. The cost of each rebinning step is linear: $2 * C * n$, where C is the high-dimensional distance computation cost and n is the total number of points in the dataset. This computation takes only a small fraction of the total time budget for a pass, but it is a cost that increases with the dimensionality of the dataset.

3.3 Termination Conditions

The inner loop of our system does the actual multi-dimensional scaling computations where we attempt to minimize the error, or **stress**. We use the popular Kruskal Stress-1 stress function [9]:

$$Stress = \frac{\sum_{i<j}(d_{ij} - p_{ij})^2}{\sum_{i<j} p_{ij}^2}$$

where d_{ij} is the Euclidean distance between two points in high-dimensional space, and p_{ij} is the distance in the low dimensional projection.

In MDSteer, the termination condition that deems the points to be well laid-out is based on stress measurements. We only measure the stress for the active point set, not all placed points. If the number of active points is greater than 100, then the measure is computed from a random sample of 100 of those active points.

If s objects are currently active, we measure the stress every $\sqrt{s}$ iterations of the spring model force calculations. When this value decreases, progress is being made. When this value fails to shrink for two consecutive calculations, we terminate the inner iteration loop.

Our current termination criterion is one of many possibilities. We could simply run the computation for a globally fixed number of iterations, or use the active point set size to set the number of iterations. The benefit of the change would be to eliminate the overhead of stress computation, but then we risk either undershooting the amount of work and ending up with more total error, or overshooting and adding overhead by doing many iterations that do not improve the layout. We could also use a value that is calculated as part of the MDS iterations, such as velocity [12].

4 Discussion

Steerability supports exploration of huge datasets. Often users do not need to see the placement of every point in order to carry out tasks of interest. Currently, people do not even attempt to carry out dimensionality reduction on huge datasets because the time it would take to lay out one million points is a huge barrier to exploration. By allowing users to immediately begin looking at the data, and to direct the computational resources to interactively-discovered areas that look promising, they may be able to answer their questions about the data long before all points have been placed. The ability to quickly see that a dataset is not promising would allow a user to abandon an unproductive direction, and immediately move on to check another that has the potential to be more informative. Without steerability, those judgements might require a turnaround time of hours or days rather than minutes. Another possible advantage is that the user spends the time while waiting for the system to finish layout engaged in productive exploration rather than waiting impatiently to start work.

The standard argument for computational steering is that human insight can help with many tasks where automatic algorithms are inadequate to fully solve the problem [14]. Visualization systems are deployed in those exact circumstances; where humans do need to have insight into the structure of a complex dataset.

We emphasize again that the critical contribution of our work is bringing steerability to MDS. Our algorithm does not complete a full layout using less time or memory than with previous approaches. In fact, our algorithm is based on, and takes time comparable to, the original Chalmers approach [4] that is notably slower than the more recent algorithms offered by Morrison [11, 12]. Rather, the benefit of our approach is that it enables the interactive investigation of datasets with both high cardinality and high dimensionality. We also do not claim to reduce the amount of memory required to do the computation.

We also distinguish steerability from visible change. Many MDS systems allow users to see the projected points move around the plane as the layout is refined, for instance the TreeComp system of

Amenta and Klingner [1]. Although users may both enjoy and benefit from seeing this real-time motion that shows them the progress of the algorithm, the only control they have is whether to stop or continue the system. The ability to control the allocation of computational resources through true steering provides far more power to users.

5 Results

MDSteer was implemented in Java and is based on the open-source HIVE spring-model MDS software infrastructure distributed by Ross and Chalmers [15]. We analyze both the running time and the layout error, known as stress, for our method. All performance figures are for a dual processor, 3.0GHz Xeon with hyperthreading, 4.0GB of main memory, and an nVidia Quadro4 980 XGL graphics card. We used the Windows XP Professional operating system and Java 1.4.2-b28 (HotSpot) with a 1.5GB heap.

We show two datasets of differing cardinality and dimensionality. We use the standard dimensionality 3 S-shaped synthetic benchmark sampled at different densities: our benchmarks are performed on cardinalities of 2,000; 5,000; 50,000; 200,000 and 1,000,000 points. Figure 5 shows the S at cardinality 200,000. The second dataset, with dimensionality 294 is a real dataset from our collaborators. In fact, the work presented here was motivated by the desire to explore this dataset and the lack of tools with which to do so. In this dataset, each point represents a modelled scenario of a possible future, where each dimension is a particular measure of environmental sustainability. For instance, dimensions include water quality, air quality as measured by carbion dioxide emissions, solid waste generated per capita, and so on. We show the 40,000 point cardinality version of this dataset in Figure 5, and benchmark comparisons for the 5000; 40,000; and 120,000 point versions.

In all cases, MDSteer provided a reasonable overview of what would become the final layout within a few seconds of startup, so that users could make informed decisions about how to steer the system and focus future computation time. The overview can always be generated quickly because only a small number of points need to be laid out. In fact, this overview is exactly what would also be generated by the Morrison algorithm, because our initial layout step corresponds to its first stage. For all subsequent layout steps, our system provides unprecedented power to users.

Timing As we discuss in Section 4, the time to place all points in the input dataset is not the right metric for judging a steerable system. The steering controls for MDSteer allow users to select which planar regions to fill in, so a measure of interest is the time required for it to fully populate one of those regions. Specifically, these timings show the result of steadily selecting a single new child bin soon after every subdivision. We show average times over three runs. Figure 3 shows the benefits of steerability, where we compare the time required to place the points that fall into a region with the time required to place all points in the whole dataset.

In a typical interactive exploration session the usage pattern would be much more dynamic and fluid, with frequent changes of selected bins. Users would often change selections before placement was complete in a region, and they might also select a larger number of bins than the single bin we use in this computation. The timing numbers here are intended to give an impression of how steerability opens up new possibilities for exploring huge datasets, and also to show that the overhead of progressive layout and rebinning is completely acceptable compared to the benefits of progressive exploration.

We compare times for MDSteer to complete a partial layout against running the whole dataset with the nonsteerable approach most similar to ours that is publicly distributed, namely Morrison's 2002 subquadratic algorithm [12] as implemented in the HIVE software distribution [15]. Also, we built MDSteer on top of HIVE, so the infrastructure time and memory costs match.

Unsurprisingly, the need for steerability is most apparent as either the dimensionality or the cardinality of the datasets increase. The most important aspect of these timing numbers is that we can load and immediately begin exploring these huge datasets that overwhelm conventional nonsteerable systems. The video accompanying this paper also communicates the look and feel of interactive exploration with steerable, progressive multidimensional scaling.

Stress Our steerable progressive algorithm calculates the final position of a point doing significantly fewer high-dimensional distance calculations than required for previous methods. We need to verify that we are still capturing the important aspects of the dataset structure in our projection; that is, verifying that the error between projected locations and the high-dimensional locations is sufficiently small. We can do so quantitatively by measuring the stress of the system, as defined in Section 3.3. Figure 4 compares the stress of MDSteer to the Morrison approach. Again, each result is the average value over three software runs. We can run the same test as for timing, where we instrument the software to report stress measurements. The comparison is somewhat unfair because completing a region requires layout of far fewer points than does completing the entire dataset. We thus also show the per-item stress graphs, which show that the stress is roughly comparable.

Visual Quality Finally, we compare the layout quality of MDSteer against the Morrison subquadratic algorithm, to ensure that the rough match of stress measurements translates into a rough match with subjective visual inspection.

Figure 1 shows that both methods are able to reproduce the S shape in the plane. Figure 2 shows that both methods also produce similar structures for the real environmental dataset.

MDSteer was able to produce a discernable two-dimensional dimensional S immediately with the 50,000 node dataset, and within a few seconds for the 1 million node S dataset. Figure 5 shows that the layout maintains the global shape of the projection while filling in the different regions of the place that were selected by the user.

6 Future Work

Currently when the system places all points in the selected bins, it falls idle until the user changes the selection. We would like to add an "auto-run" mode, so that the system would automatically change the selection to start work on a nearby area if there is no more work to do in the current bins. If left unattended, such a system would eventually place the entire dataset. Furthermore, in the current implementation, if during execution the user realizes that she wants to layout the whole dataset, the speed of the full layout will be comparable to the original Chalmers approach [4]. We would like our system to switch to a more efficient approach [11] if the user chooses to layout the full dataset.

We would like to implement Morrison's 2003 nearest neighbour finding technique [11] to improve the efficiency of our starting location algorithm. It would be interesting to explore further whether we could provide faster or more accurate layout by changing the selection criteria for the neighborhood and random sample sets used in the inner loop of MDS iteration, exploiting the known structure of our hierarchical bins instead of using purely random selection.

We are also intrigued by the challenge of creating a fully progressive algorithm. In the present implementation, rebinning is a global pass, which is acceptable because its cost is still overshadowed by the MDS iteration cost. However, this sort of global computation will eventually form a limit to scaling datasets of large cardinality or dimensionality, so progressive binning would be a good match with the current progressive layout. We would then have an approach limited only by system memory constraints, and that might scale far past our current million-node limit.

7 Conclusion

We present a system for steerable and progressive multidimensional scaling that allows users to interactively explore huge datasets. Steering allows computational power to be focused where it is needed to support exploration in parallel with continuing the layout process. We subdivide the low-dimensional plane where our high-dimensional points are projected into a hierarchical decomposition of rectangular screen-space regions. The user can interactively select regions of interest, and then most of the available computational resources are spent on refining this region. These small local areas can quickly be fully populated with the dataset points that project to the selected region of the plane. Our system handles datasets with dimensionalities of up to several hundred, and cardinalities of over one million.

8 Acknowledgements

We thank our collaborators at the University of British Columbia's Sustainable Development Research Initiative, Georgia Basin Futures Project, and Envision Sustainability Tools for the use of the environmental dataset. We also thank Luc Girardin of Macrofocus for the multiple-cardinality S dataset. We appreciate many productive discussions on dimensionality reduction with Katherine St. John, the technical writing contributions of Ciarán Llachlan Leavitt, and the video production help from James Slack and Kristian Hildebrand. This work was funded by the GEOIDE NCE.

References

[1] N. Amenta and J. Klingner. Visualizing sets of evolutionary trees. In *Proc. IEEE Symposium on Information Visualization*, pages 71–74, 2002.

[2] W. Basalaj. Incremental multidimensional scaling method for database visualization. In *Proc. Visual Data Exploration and Analysis VI, SPIE*, volume 3643, pages 149–158, 1999.

[3] W. Basalaj. Proximity visualization of abstract data. Technical Report 509, University of Cambridge Computer Laboratory, January 2001.

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

[5] G. S. Davidson, B. N. Wylie, and K. W. Boyack. Cluster stability and the use of noise in interpretation of clustering. In *Proc. IEEE Symposium on Information Visualization*, pages 23–30, 2001.

[6] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[7] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, 1995.

[8] Y. Koren, L. Carmel, and D. Harel. ACE: A fast multiscale eigenvector computation for drawing huge graphs. In *Proc. IEEE Symposium on Information Visualization*, pages 137–144, 2002.

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

[10] J. X. Li. Visualization of high dimensional data with relational perspective map. *Information Visualization*, 3(1):49–59, 2004.

[11] A. Morrison and M. Chalmers. Improving hybrid MDS with pivot-based searching. In *Proc. IEEE Symposium on Information Visualization*, pages 85–90, 2003.

[12] A. Morrison, G. Ross, and M. Chalmers. A hybrid layout algorithm for subquadratic multidimensional scaling. In *Proc. IEEE Symposium on Information Visualization*, pages 152–158, 2002.

[13] A. Morrison, G. Ross, and M. Chalmers. Fast multidimensional scaling through sampling, springs and interpolation. *Information Visualization*, 2(1):68–77, 2003.

[14] S.G. Parker and C.R. Johnson. SCIRun: A scientific programming environment for computational steering. In *Proc. Supercomputing*, 1995.

[15] G. Ross and M. Chalmers. A visual workspace for hybrid multidimensional scaling algorithms. In *Proc. IEEE Symposium on Information Visualization*, pages 91–96, 2003.

[16] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500), Dec 22 2000.

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

[18] W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17:401–419, 1952.

[19] J. A. Wise, J. 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 Symposium on Information Visualization*, pages 51–58. IEEE, 1995.

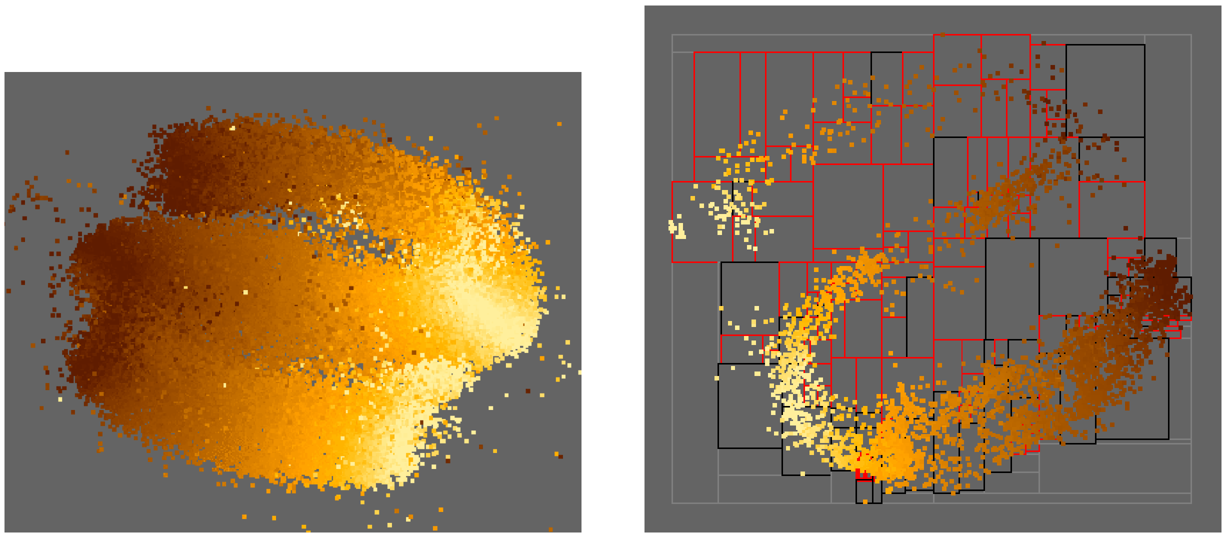

Figure 1: Visual Quality: S Dataset **Left**: We show the 50,000 point S-shaped benchmark dataset laid out with the Morrison [12] algorithm, taken after a full layout computation that takes 150 seconds. **Right**: We show a partially placed version of the same dataset after steering with MDSteer for roughly 20 seconds. We see the same large-scale structure, and the local region on the lower right where we have focused computational resources is completely filled in.

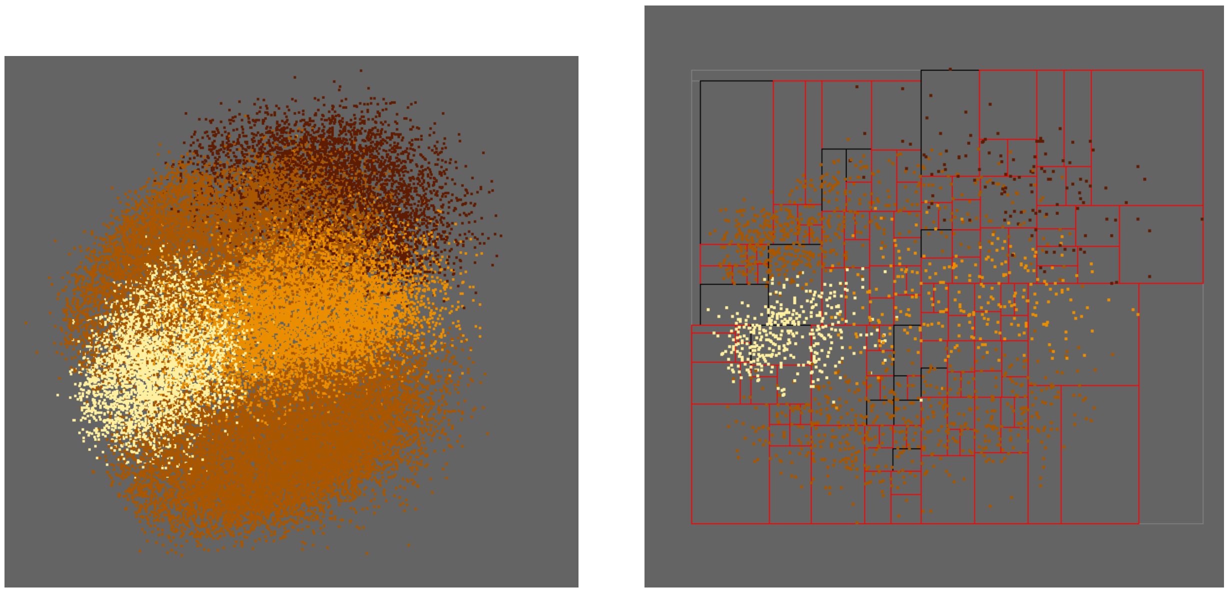

Figure 2: Visual Quality: Environmental Dataset. **Left**: We show the 40,000 point real environmental dataset laid out with the Morrison [12] algorithm, taken after a full layout computation that takes 16 minutes. **Right**: We show a partially placed version of the same environmental dataset after steering with MDSteer for roughly 2 minutes. Again, we see the same large-scale structure.

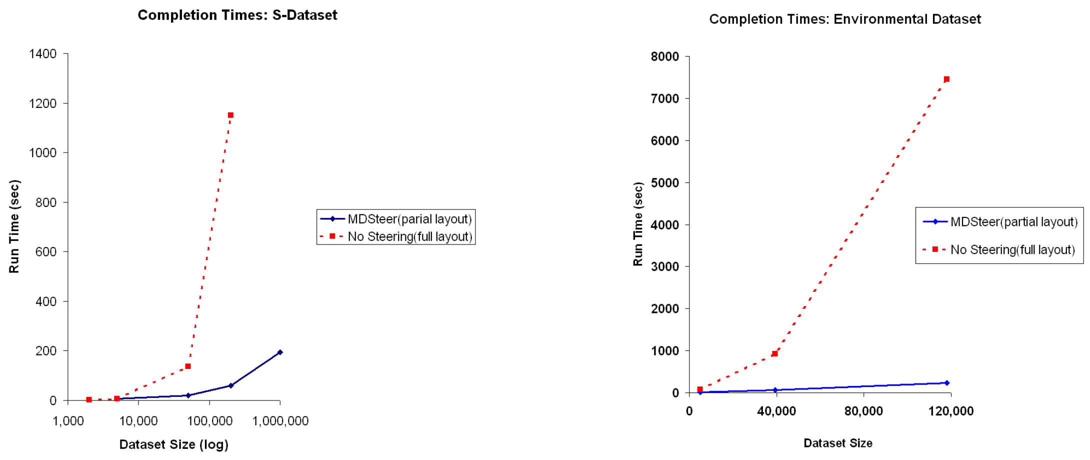

Figure 3: Time to complete the layout of a spatial subregion of the dataset for our steerable method compared to the full layout of non-steerable Morrison [12]method. The MDSteer timings are for a subset of the total points because we are steering to a spatial subregion. For example, MDSteer placed an average of 3900 points for the 1 Million point sized S-dataset. **Left**: We compare the times across a range of dataset sizes of a synthetic 3-dimensional S-shaped benchmark dataset. **Right**: We compare the times across a range of a 294-dimensional real dataset of environmental sustainability measures.

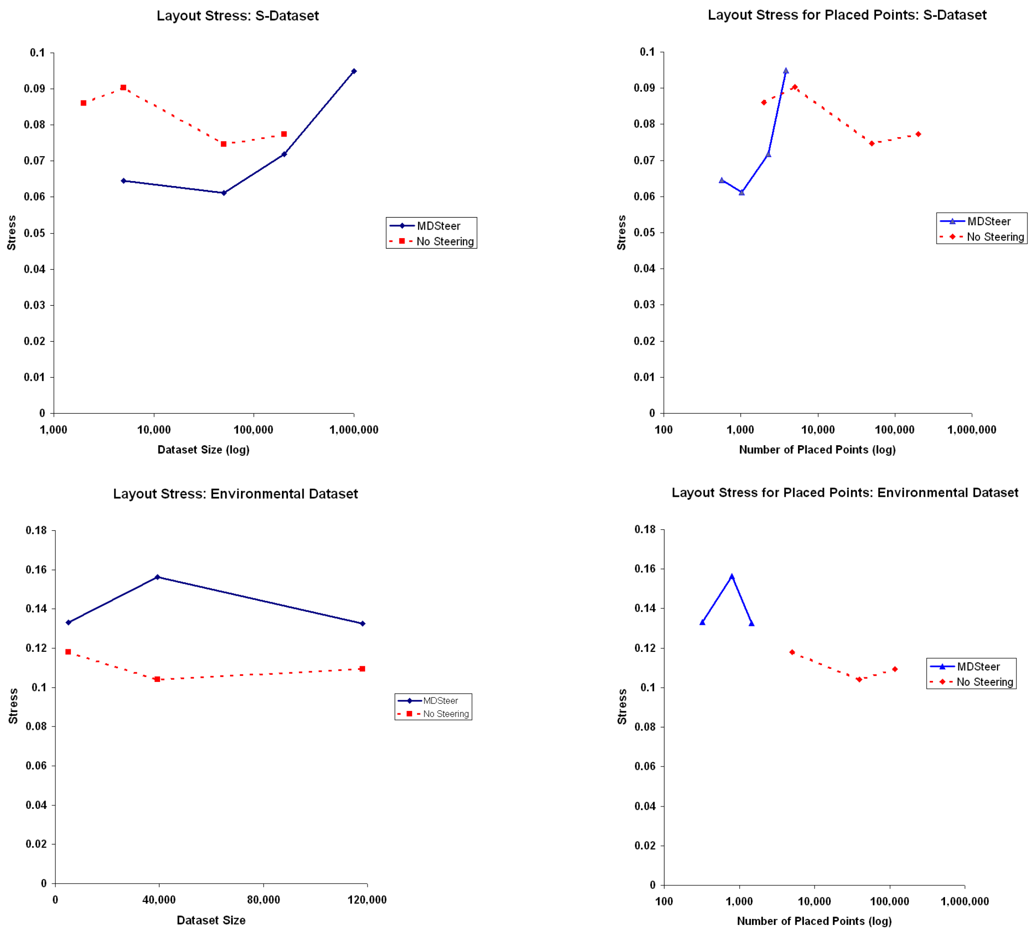

Figure 4: Layout Stress **Top Left**: We compare the stress across a range of dataset sizes of a synthetic 3-dimensional S-Shaped dataset. **Top Right**: We compare the stress across a range of a 294-dimensional real dataset of environmental sustainability measures. **Bottom Left, Bottom Right**: We plot the layout stress for the actual number of points placed, rather than the dataset size. These two quantities are disjoint with our scalable methods.

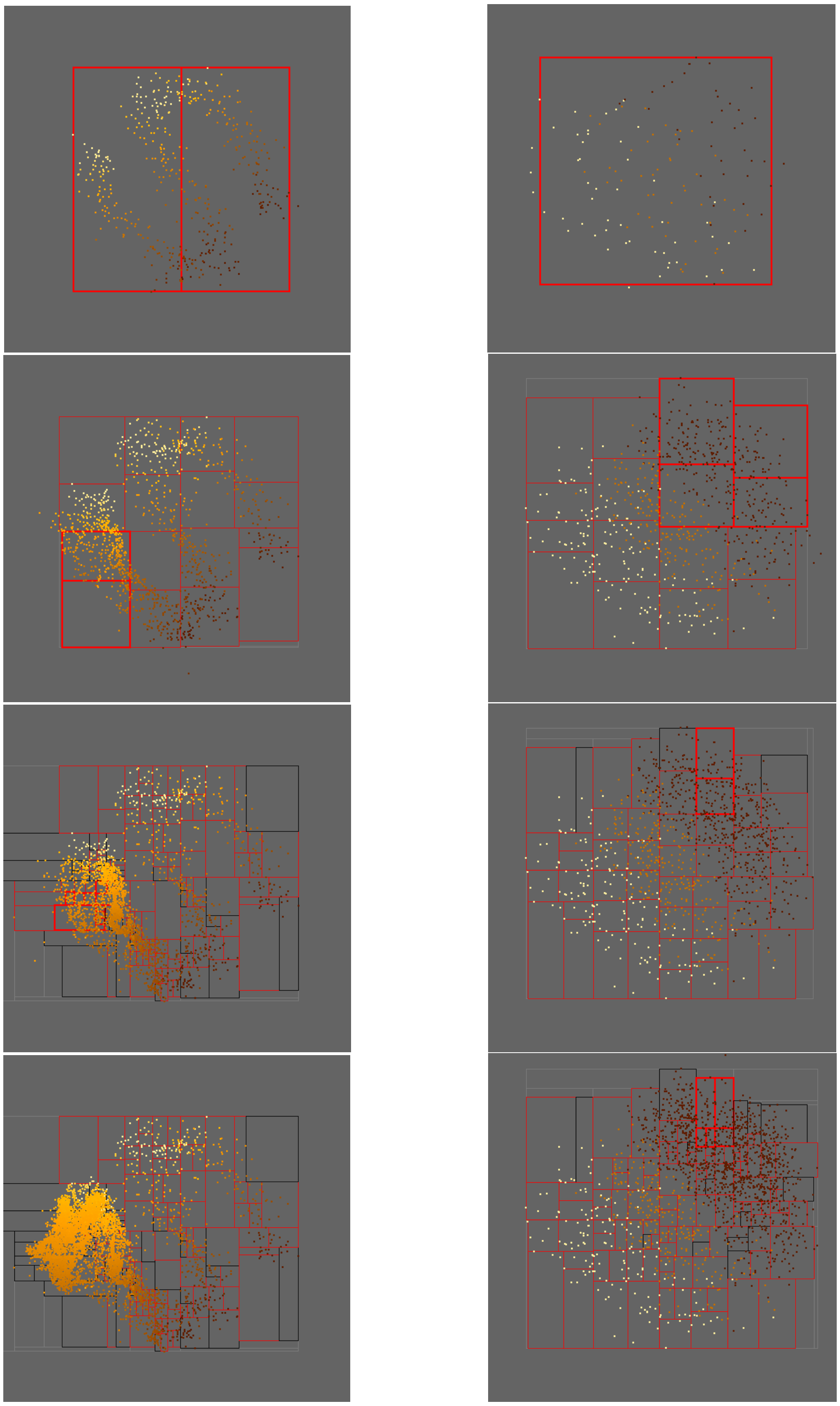
Figure 5: Steerable Progressive Layout. We show the progression of steered layouts. The onscreen regions outlined in bold red are user-selected bins. Black outlines identify bins that contain no more unplaced points. **Left Column**: S benchmark data with dimensionality 3 and cardinality 200,000. **Right Column**: Real environmental data with dimensionality 294 and cardinality 40,000.