Re-implementation of Visual Abstraction and Exploration of Multi-class Scatterplots

DAVID GROS[*], HAITONG LI[*], and JIAYU LIU[*], University of California, Davis

A re-implementation and exploration of "Visual Abstraction and Exploration of Multi-class Scatterplots" [4] is performed. We present a web application similar to the system provided by the paper for exploring multi-class scatterplots. Our system can re-sample the points of different classes to address the overdraw problem of conventional scatterplots while preserving the distribution features. The colors of different classes are automatically generated for better distinguishability. We have applied our system to a challenging dataset and the results are shown at the end of this paper.

CCS Concepts: • **Information visualization → interactive visualization**; *visual abstraction*.

Additional Key Words and Phrases: Scatterplot, Blue noise sampling, Poisson disc sampling, visualization systems

1 INTRODUCTION

This work re-implements work by Chen et al [4]. While for a complete description of motivation and prior work we will defer to that manuscript, here briefly summarize the motivation.

Chen et al. highlight the importance of scatter plots to explore 2d distributions, clusters, and outliers. However, when trying to plot a large number of points, scatter plots suffer from a issue of "overdraw". If one has two sets of points which are different classes and thus different colors, one class might draw over another, resulting in regions in which it is impossible to distinguish between points of different classes.

The authors give several prior work which they claim is inadequate. For example they cite [1, 2, 5, 8] which show density based shadings representing the distribution continuously over the domain. Chen et al point out that when dealing with multiclass data, density shading has issues with synthesizing new colors as the transparent shaded regions of the classes overlap. For example a red class and green class might produce varying shades of yellow in their overlap. As the number classes increases, this can make the classes hard to interpret.

The authors also cite [6, 7] as examples of techniques that adjusts points in order to avoid overlap. However, Chen et al. believe these are not ideal techniques as the it distorts the underlying data distribution.

The proposed sampling algorithm and visualization presentation solves these issues and neither synthesizes new colors or creates new points which do not exist in the dataset.

2 PROJECT GOALS

For this class project we set out to understand the techniques and intuitions of the paper, implement the proposed sampling algorithm, and demonstrate the visualization in a interactive web application. As noted in our proposal, we did not attempt to include all the interaction paradigms as they were tangential to the main algorithm, but we capture

[*]All authors contributed equally to this research.

the main interaction paradigms, such as zooming to reveal additional points and sidebar class-level histograms for exploring and filter each class. We were able to successfully develop an implementation that addresses the overdraw of a multi-class scatter plot.

3 IMPLEMENTATION

3.1 Density Estimation

The first step of the algorithm requires defining a continuous density function over the domain. This is done with Kernel Density Estimation (KDE). This calculates the likelihood a point would appear at a given position by calculating the density at that point normalized by the total density. The density is determined by placing a Gaussian distribution around each point in the dataset. Thus positions that are far from other points will have lower density. This can be expressed as

$$\hat{f}_i(x) = \sum_{j=1}^{m} K_h(x - x_i^j) \tag{1}$$

Where $\hat{f}_i$ is a density measure at point x for class i. K_h is the kernel function, in this case a Gaussian with bandwidth h specified by Silverman's Rule of Thumb [9].

3.2 Multi-class Poison Disk Sampling

The next part of the algorithm involves sampling using multi-class blue noise sampling. Chen et al. defer much of the explanation of this technique to [10], thus we found understanding that work to be critical. The reader is encouraged to also see [10] for a thorough explanation, but the relevant sections are summarized here with notes of the changes we found necessary.

A simple approach to sampling the sample random points uniformly. However, this leads to issues with not fully covering the space and may lead to overlaps. Blue noise instead will be better cover all of the space through a process of poison disk sampling. This is done through a "dart throwing" algorithm. One has a radius r which is the closest a point is permitted to be to another point. One starts with no points. Then one "throws a dart" in order to get a random point. One then checks to see if this point is with r distance of an existing point. If there are no other points in the r radius the point is added to the sampling. If there is a point already with r radius, then the sample is discarded and other is take. This guarantees that no points clumped together and encourages more uniform covering of the space.

The insight of [10] was to extend this multiple classes. One has thus maintains a $r_{i,j}$ which is the distance points of class i can be from class j. While [10] specifies r_i as user defined parameter and an algorithm is used to calculate each j distance, for Chen et al. the $r_{i,j}$ is a function of the density of class j at the given point. More specifically we define a ω parameter which is initialized as:

$$\omega = \frac{r}{\varphi * \sqrt{N_i}} \bar{f} \tag{2}$$

where r is defined as the point radius, φ is defined as the current zoom level, N_i is the number of points in class i, and $\bar{f}$ is the mean density of all dataclasses over all points. We note that this is different than the initialization used by Chen et al in that we divide by $\sqrt{N_i}$. We find that without this term the sampling removes too many points in sparse regions of the space (outliers).

A possible reason for the need of this difference is we calculate $\bar{f}$ over all discrete points in the dataset using the class level , while Chen et al. describe $\bar{f}$ as "mean density of all data classes over the domain". We assume they also accomplished this through calculating the density for all points, but is possible they use a technique to do it continuously or construct a different KDE which is shared between all classes. We find the paper does not have sufficient details to determine this, but that our initialization produces good results without manual adjustment.

We then use this to build the R-matrix of acceptable radiuses for the disk sampling. Following Chen et al. we set the diagonal elements of the matrix to $\omega/\hat{f}_i(x)$. While Chen et al. defer to [10] for explaining the initialization of the off diagonal elements, we find that the algorithm [10] requires fixed parameters for the desired radiuses, which are not available when dynamically calculating the values based off the density. Because the values calculated from the density estimation will almost always be unique between classes, the notion of "priority groups" used in [10] does not apply. Thus we are able to initialize the off diagonal elements according as

$$\left(\frac{\omega}{\hat{f}_i(x)} + \frac{\omega}{\hat{f}_j(x)}\right) * \frac{1}{2} \tag{3}$$

Or the averaging between the two estimated radiuses.

While Chen et al's implementation still allows for overdraw of the points by chance, we choose to place hard limits on the amount of overdraw between points. Thus $r_{i,j}$ is limited to be a minimum of 1.25 times the radius of the points for on diagonal values and 1.6 the radius of points for off-diagonal values.

During the course of the dart throwing algorithm Chen et al. describe that the least filled class is chosen. They again defer to [10] for the calculation of least filled. The formula in [7] again depends on fixed radius values which does not apply when dynamically choosing the radius based off density. We assume then the "desired fill rate" includes all points mean that the current fill rate is described as s_i/N_i where s_i is the number of points of a class already sampled. By always selecting the least filled class to sample and test, we ensure that our sampling resembles the class distribution of the underlying dataset and that certain classes do not "crowd out" the points from other classes.

3.3 Color optimization

According to [4], color is an important property when exploring multi-class data with scatterplots. Their system has the ability to automatically generate distinguishable colors for each class of data. The goal of color optimization is to make the difference between classes obvious to human eyes. So other than choosing colors that are generally distinguishable from each other, the relationship between classes should also be taken into account. We want the colors of the classes that are close to each other be as different as possible. To achieve this, the paper offers us an objective function.

$$E_{cost} = \sum_{m=1}^{M} \beta_m \sum_{i,j<n,i<j} \alpha_{m,i,j} |C_i - C_j| \tag{4}$$

A scatterplot is divided into M local regions. i and j are the indices of classes. $\alpha_{m,i,j}$ denotes the inter-class weight, while β_m denotes intra-class weight. C_i is the current color for class i. Further explanation about equation 4 can be found in [4]. Basically, E_{cost} measures how distinguishable the classes are, using a certain set of colors by taking the density of points of different classes in different local regions into account. The paper's approach is to maximize K_{cost} to get an optimal color set. And they add a soft constraint to ensure the colors are d away from each other in CIELAB color space to prevent unsatisfying result, where d is the Euler distance between colors.

Our approach invert the order of the soft constraint and the objective function since Chen et al. did not mention how they generate the colors to reach maximum K_{cost}, or the solver to solve the objective function. We first generate colors in CIELAB color space that are apart from each other by at least the distance d several times, then take the color set with maximum K_{cost} as our automatically generated color set. To generate colors, we again use a variation of

Poisson-disc sampling[3]. Poisson-disc sampling can generate evenly distributed samples in a given range of space. Here we manipulate its feature of having a hard constraint for minimum distance between samples to generate colors that are distance d from each other. The CIELAB color space has three channels: $L*$, $a*$ and $b*$. $L*$ denotes the perceptually perceived lightness and its a user specified parameter in [4]. In our approach, we fix the lightness value to 50, so the sampler only needs to sample color in a 2D space. What's more, the frontend color system allow us to define colors in RGB color space, which is smaller than CIELAB. So the generated color set might be beyond the space of RGB. We clamp the value converted from CIELAB in the range of $[0, 255]$ for displaying the colors in RGB. We calculate K_cost for a given color set based on our re-sampled scatter points to speed up the process. We evenly divide the scatterplot into 100 local regions to calculate α and β to get K_cost. The color set generation process is looped for 20 times in our approach, and the one with maximum K_cost is chosen as the result of automatic color selection.

3.4 Visualization Interface

The data presented on the interface is received through `jQuery.ajax()` calls that communicate with the data processing django-based backend. The plots and most of the interactive features are implemented with the functions in `d3.js`. Lasso selection and brushing on histograms rely on the imported `d3-lasso.js` and `d3-brush.js`.

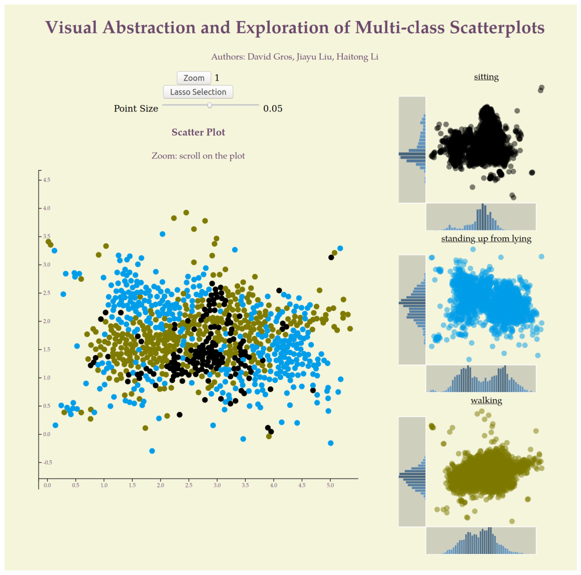

Fig. 1. The sampled scatter plot of the person activity dataset.

4 PAPER INADEQUACIES AND SOLUTIONS

5 RESULT

For testing our system, we applied our web application to the same dataset Chen et al. used in their paper. We choose to use the dataset because it contains the multi-class information that is so dense in certain regions to cause severe overdraw in scatterplots.

5.1 Visualization Interface

The interface of the visualization system we have implemented has two major parts: the main scatter plot and the data class list. The main plot displays data points from all classes, which can be identified with distinct colors. The toolbox for the main plot contains zoom, lasso selection and point size features. Initially, the plot shows all data points at zoom level 1. When zoom is enabled, the user can scroll up or down on the plot to zoom in or out the plot. As the plot is zoomed, more detailed data points will be added to the plot according to the zoom level, thus the user will be able to explore more data when necessary. The lasso selection allows the user to select a specific area in the plot to emphasize the data points for clearer observation. The point size slider makes it possible to adjust point size according to the user's preference. The current zoom level and point size are also displayed for reference.

The data class list contains small supplemental plots for each individual class. The data points are also displayed in their specific colors. The user can choose which classes to display by checking the checkboxes on each plot. For each class-specific plot, there are two histograms showing the values of point density for x and y axes. There is a shaded rectangle on each of the histogram that defines the range of points on each axis that will be on the main display. The user can adjust the range to display different sets of data points.

Compared to Fig.8 in [4], our implementation does not include the configuration panel because it may require complicated communications with the data processing server which are beyond our time frame and may be out of a general user's scope. For the toolbox of the main plot, we simplified the tools to the most useful ones for the potential users. The overall layout is also modified for better visual appearance.

6 LIMITATIONS

A large limitation of the approach is the slow performance sampling points. This is mitigated by pre-computing the sampling for the dataset and cacheing it on disk. Our implementation is for the most part done in pure python. Improvements could be made by converting the KDE to be approximate rather than exact, changing the R-matrix acceptance distance checks to use a data structure like a KD-tree subquadratic distance checks, and reimplementing the hot regions of the code in a more performant language.

As discussed in 3.2 outputs can be sensitive to choice of ω, and better approaches for setting this and determining the R-matrix values could be explored.

Additionally, depending on the use application, more interactions or animations could be added for the specific use case.

7 CONCLUSION

For this project we reimplemented the sampling algorithm of Chen et al. After implementing the algorithm, we find it might be unecessarily complicated. We suspect that a simpler approach based off uniform random sampling, with a cleaning pass to remove overlapping points would be equally as effective and faster. As mentioned in 3.2 we also found the author's discussion of the method to be missing key details that had to be assumed.

Still, we found the project instructive for understanding concepts like KDE, blue noise sampling, color optimization, and d3.

REFERENCES

[1] S. Bachthaler and D. Weiskopf. 2008. Continuous Scatterplots. IEEE Transactions on Visualization and Computer Graphics 14, 6 (Nov 2008), 1428–1435. https://doi.org/10.1109/TVCG.2008.119

[2] Sven Bachthaler and Daniel Weiskopf. 2009. Efficient and Adaptive Rendering of 2-D Continuous Scatterplots. In Proceedings of the 11th Eurographics / IEEE - VGTC Conference on Visualization (Berlin, Germany) (EuroVis'09). The Eurographs Association John Wiley Sons, Ltd., Chichester, GBR, 743–750. https://doi.org/10.1111/j.1467-8659.2009.01478.x

[3] Robert Bridson. 2007. Fast Poisson disk sampling in arbitrary dimensions. SIGGRAPH sketches 10 (2007), 1278780–1278807.

[4] Haidong Chen, Wei Chen, Honghui Mei, Zhiqi Liu, Kun Zhou, Weifeng Chen, Wentao Gu, and Kwan-Liu Ma. 2014. Visual abstraction and exploration of multi-class scatterplots. IEEE Transactions on Visualization and Computer Graphics 20, 12 (2014), 1683–1692.

[5] H. Hagh-Shenas, S. Kim, V. Interrante, and C. Healey. 2007. Weaving Versus Blending: a quantitative assessment of the information carrying capacities of two alternative methods for conveying multivariate data with color. IEEE Transactions on Visualization and Computer Graphics 13, 6 (Nov 2007), 1270–1277. https://doi.org/10.1109/TVCG.2007.70623

[6] Daniel Keim, Ming Hao, Umeshwar Dayal, Halldor Janetzko, and Peter Bak. 2010. Generalized Scatter Plots. Information Visualization 9 (12 2010), 301–311. https://doi.org/10.1057/ivs.2009.34

[7] Daniel A. Keim and Annemarie Herrmann. 1998. The Gridfit Algorithm: An Efficient and Effective Approach to Visualizing Large Amounts of Spatial Data. In Proceedings of the Conference on Visualization '98 (Research Triangle Park, North Carolina, USA) (VIS '98). IEEE Computer Society Press, Washington, DC, USA, 181–188.

[8] A. Mayorga and M. Gleicher. 2013. Splatterplots: Overcoming Overdraw in Scatter Plots. IEEE Transactions on Visualization and Computer Graphics 19, 9 (Sep. 2013), 1526–1538. https://doi.org/10.1109/TVCG.2013.65

[9] B. W. Silverman. 1986. Density Estimation for Statistics and Data Analysis. Chapman & Hall, London.

[10] Li-Yi Wei. 2010. Multi-Class Blue Noise Sampling. In ACM SIGGRAPH 2010 Papers (Los Angeles, California) (SIGGRAPH '10). Association for Computing Machinery, New York, NY, USA, Article 79, 8 pages. https://doi.org/10.1145/1833349.1778816