Choosing Colors for Geometric Graphs via Color Space Embeddings

Michael B. Dillencourt, David Eppstein, and Michael T. Goodrich

Dept. of Computer Science, Univ. of California, Irvine, CA 92697-3425 USA.
{dillenco,eppstein,goodrich}(at)ics.uci.edu.

Abstract. Graph drawing research traditionally focuses on producing geometric embeddings of graphs satisfying various aesthetic constraints. However additional work must still be done to draw a graph even after its geometric embedding is specified: assigning display colors to the graph's vertices. We study the additional aesthetic criterion of assigning distinct colors to vertices of a geometric graph so that the colors assigned to adjacent vertices are as different from one another as possible. We formulate this as a problem involving perceptual metrics in color space and we develop algorithms for solving this problem by embedding the graph in color space. We also present an application of this work to a distributed load-balancing visualization problem.
Keywords: graph drawing, graph coloring, color space, color perception

1 Introduction

Graphs are frequently visualized by embedding them in geometric spaces. That is, geometric representations are natural tools for visualizations; hence, we embed graphs in geometric spaces in order to display them. For instance, producing geometric embeddings of combinatorial graphs so as to satisfy various aesthetic constraints is a major component of *graph drawing* (e.g., see [5, 8, 9, 12]).

Once a graph has been embedded in a geometric space, such as $\mathbf{R}^2$, we refer to it as a *geometric graph*. That is, a geometric graph is a graph $G = (V, E)$ such that the vertices are geometric objects in $\mathbf{R}^d$ and the edges are geometric objects connecting pairs of vertices. Note that this definition is more general than the definition of "geometric graph" popularized by Alon and Erdös [1], in that they define a geometric graph to be a graph $G = (V, E)$ such that the vertices are distinct points in $\mathbf{R}^2$ and edges are straight line segments. For example, we allow a geometric graph to be a planar map, where the vertices are regions and the edges are defined by regions that share a common border. Intuitively, a geometric graph G is a graph that is "almost drawn," for to actually display G on some device we must assign colors to its vertices.

In this paper, we are interested in this final step in the drawing of a geometric graph, G. In particular, we are interested in methods for choosing colors for the vertices of a geometric graph so as to make distinctions between vertices as apparent as possible. We are also interested in the related *map coloring* problem,

where we color the faces of a map so as to make the distinctions between adjacent faces as strong as possible. The main challenge in performing a coloring is more than just choosing a good set of colors, however, for we also want to assign colors to vertices in a way that makes the colors assigned to adjacent vertices as different as possible. That is, we are interested in a bi-criterion color assignment problem, where all the colors are different from one another and adjacent colors are really different.

1.1 Previous Related Work

Graph coloring is a classic problem in algorithmic graph theory (e.g., see [3]). Given a graph G, the traditional version of this problem is to color the vertices of G with as few colors as possible so that adjacent vertices always have different colors. The traditional graph coloring problem is posed as a "coloring" problem purely for abstraction's sake, however: no paint or pixels are involved. Even so, there has been some prior work on algorithms for coloring geometric graphs (in the traditional sense). For example, there has been some prior research on coloring quadtrees [2] and intersection graphs [6]. In addition, there has been a host of prior work on the traditional version of graph coloring for purely combinatoric graphs (e.g., see [3]).

Also of interest is work that has been published in the information visualization literature on methods for choosing colors effectively for data presentation. Healey [7] presents a heuristic for choosing a well-separated set of colors for visualizing segmentation data in images. Likewise, Levkowitz and Herman [11], Robertson [16], and Ware [22] discuss various ways for effectively building color maps that correspond to data values in an image or data visualization (e.g., a bar chart histogram). Rheingans and Tebbs [15] describe an interactive approach that constructs a color scale by tracing a path through color space. Brewer [4] describes several guidelines for choosing colors for data visualization, focusing primarily on ways of representing linear numerical scales. There are also several good books on the subject of color use for data visualization (e.g., see [19–21]). In spite of this wealth of previous work on color selection for data visualization, we are unfamiliar with any prior work that uses adjacency information to select dissimilar colors for visualization purposes.

1.2 Our Results

In this paper, we investigate the following problem for geometric graph coloring:

- *Maximizing minimum color difference.* Given a geometric graph G and a color space $\mathcal{C}$, assign visibly distinct colors from $\mathcal{C}$ to the vertices of G so to maximize the minimum color difference across the endpoints of edges in G.

We investigate this problem in terms of embeddings of G in the human-perceptible subset of the color space $\mathcal{C}$. This embedding of G in $\mathcal{C}$ is purely to find good colors to assign to the vertices of G, however. The actual coordinates for G's vertices and equations for G's edges will still use G's geometric

embedding in $\mathbf{R}^d$ (e.g., as produced by an existing graph-drawing algorithm). Nevertheless, the placement of vertices and edges in our embedding of G in $\mathcal{C}$ implies a "goodness" score on the degree to which adjacent vertices are well-separated and non-adjacent vertices are fairly-separated (which corresponds to a similar degree of separation for the vertex colors when we display G using its original geometry). We design a force-directed algorithm to produce such embeddings.

By planar duality, our algorithms are also applicable to the *map coloring* problem, where we are given a planar map and asked to color the regions with distinct colors so that the color difference between bordering regions is as large as possible. We give an application of this map coloring problem to an interesting data visualization problem for load-balancing distributed numerical algorithms.

2 Color Space

Since we wish to assign colors to the vertices of a geometric graph so that colors assigned to adjacent vertices look as different as possible, it is useful to have a precise, mathematical notion of color and color difference.

Pure colors can be defined in terms of wavelengths of light, with the visible spectrum of colors going roughly from 400 nm (violet) to 800 nm (red). Humans perceive color, however, as a combination of intensity signals from three types of cone cells in our eyes:

- **S cone cells**: These cells respond to short wavelengths and typically have their peak transmission around 440 nm (violet). (For historical reasons, these cells are often referred to as "blue" cone cells.)
- **M cone cells**: These cells respond to medium wavelengths and typically have their peak transmission around 550 nm (yellow-green). (For historical reasons, these cells are often referred to as "green" cone cells.)
- **L cone cells**: These cells respond to long wavelengths and typically have their peak transmission around 570 nm (yellow). (For historical reasons, these cells are often referred to as "red" cone cells.)

This physiology forms the basis of all color displays, from old-fashioned color TVs to modern-day color LCD panels and plasma displays, for these displays create what we perceive as colors by an additive combination of three color intensities, such as red, green, and blue (RGB, as exemplified by the sRGB space [18] used by many digital cameras and color displays). This physiology also forms the basis of most color printing, as well, where printers create what we perceive as colors by the subtractive combination of three color intensities, such as cyan, magenta, and yellow (CMY). Thus, colors can be viewed as belonging to a three-dimensional *color space*. Moreover, RGB, which defines a three-dimensional cube of color values, is the color space that corresponds to the way most modern devices display colors.

Ironically, even though the RGB space is the most popular for display devices, humans are very poor at interpreting the perceived color that results from the

addition of intensity values in red, green, and blue. Moreover, perceived color differences do not define a uniform metric in RGB space. Our brains instead use the following notions:

- **Hue**: the actual color, e.g., "blue," "yellow," "orange," etc., as defined by a radial value around a color wheel.
- **Saturation**: the vividness or dullness of the color.
- **Luminosity**: the lightness or darkness of the color.

Thus, human perceived color defines a three-dimensional color space, called HSL, which corresponds to two cylindrical cones joined at their base, as shown in Figure 1. The two apexes of these cones correspond to opposite corners in RGB space. As with RGB, however, perceived color differences do not define a uniform metric in the HSL color space. Moreover, the geometry of the HSL space makes choosing colors inside its double-cone of visible colors more challenging.

CIE L*a*b* (or "Lab," for short) is an absolute color space that defines each color uniquely as a combination of Luminosity (L), a value, a*, which is a signed number that indicates the degree of magenta (positive) or green (negative) in a color, and a value, b*, which is a signed number that indicates the degree of yellow (positive) or blue (negative) in a color. Geometrically, Lab is a slightly distorted version of the HSL double-cone, with color points addressed using Cartesian coordinates. Thus, defining the subset of visible colors in Lab space is admittedly more challenging. Offsetting this drawback, however, is the fact that empirical evidence supports the claim that Euclidean distance in this color space corresponds to perceptual color difference [17]. There is a related, CIE L*u*v* color space, which also is designed to provide a uniform color-difference metric, but the Lab color space seems to be more uniform. Thus, the Lab color space is the more popular of the two.

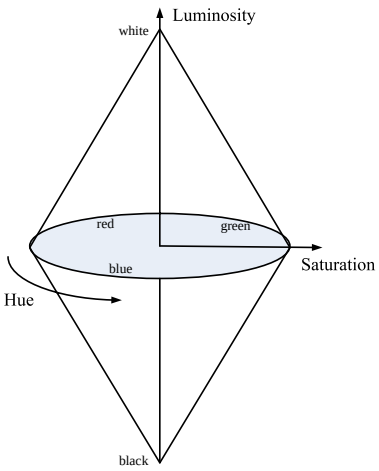

Fig. 1. The Hue-Saturation-Luminosity (HSL) color space.

There is a tradeoff between the two most popular color spaces, then. RGB corresponds better to display hardware and it defines a simple cube geometry for the space of visible colors. But perceived color difference is not a uniform metric in RGB. Lab space, on the other hand, has a more complex geometry and requires a translation to RGB for display purposes, but it supports a uniform color-difference metric. In this paper, therefore, we explore color choosing algorithms for both of these spaces.

3 Application

A specific application motivating this research is a problem in distributed programming. The Navigational Programming (NavP) methodology [13] for converting a sequential program into a parallel distributed program using migrating threads consists of the following three steps:

1. **Data Distribution:** The data used by the program is distributed over the network. The guiding heuristic principle is minimizing communication cost while balancing the load on each processing element (PE).
2. **Computation Distribution:** Navigational commands ("hop" statements) are inserted into the sequential code. The output of this step is a *distributed sequential program*, a single thread that "follows" the data through the network.
3. **Pipelining:** The single migrating thread produced in step 2 is broken into multiple threads, which are then formed into a pipeline by adding appropriate synchronization commands.

The methodology incorporates a feedback loop: information obtained in step 3 can be used to improve the data distribution in a subsequent application of the three steps.

The data distribution step is based on constructing a Navigational Trace Graph (NTG), which relates communication costs to data placement, and then applying a graph partitioning heuristic. In the NTG, the vertices are the data elements, and edge weights between vertices reflect the cost of placing the corresponding data elements on different machines [14]. Among the factors influencing the edge weights between two data items are (1) whether one of the data items is used directly in the computation of the other; (2) whether they are physically allocated close together in the sequential program; (3) whether they are referenced in temporally close statements in the code. The first factor is a direct source of communication overhead if the elements are on different machines, while the second and the third attempt to capture locality information that is implicit in the sequential code and may significantly affect performance. Additional factors affect the partitioning, such as balancing the computational load and amount of data on each PE, and constraints that certain data elements must be on different PE's (to allow parallelism in subsequent steps).

The interaction of these constraints can be complex, so it is important to be able to effectively visualize the resulting data partitions. One ingredient of a

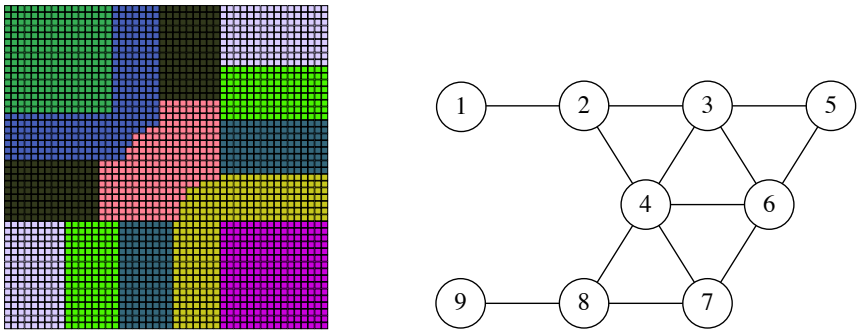

Fig. 2. The Navigational Programming application. (a) an example partition of an array of data into nine regions for a NavP application, colored by a preliminary version of our algorithm; and (b) the corresponding graph of adjacent regions.

good visualization tool is effective use of color. In this particular visualization context, the individual sets in the partitions are not necessarily connected. Hence our scheme for assigning colors to partitions should follow two basic principles:

1. The colors assigned to regions in the partition should be highly dissimilar, to make it easy to see the boundaries between regions.
2. All colors used should be somewhat dissimilar from each other, so that it is apparent which disconnected regions belong to the same set in the partition.

An example of a partition produced by this system is shown in Figure 2. The underlying sequential code from which this partition, adapted from Lee and Kedem [10], can be conceptualized as a sequence of scans over a square matrix, where the scans alternate between row-major and column-major order and also alternate directions. In each scan, the value of each element $A[i, j]$ is computed as a function of its neighbors and also the neighbors of its transposed element $A[j, i]$. As can be seen, the partition is somewhat irregular. Because of the strong data affinity between each element and its transpose, the sets are symmetric about the diagonal of the matrix, and some of them are not connected. Thus, because of the irregularity of the partitioning and the potential complexity of the partitions, it is useful to have an effective assignment of colors assigned to the regions in order to visualize the regions and their interactions.

4 What is a Good Coloring?

Formally, our problem can be stated as follows. We are given as input an undirected graph G, the vertices of which have been partitioned into *regions* r_i. We would like to display this region structure, overlayed on a conventional drawing of the graph, by assigning distinct colors to vertices in different regions. Our task is to choose a color for each region, satisfying the following constraints:

– Each region must have a different color, and the colors assigned to regions must be visually distinct.
– If two regions r_i and r_j are adjacent in G (that is, if some vertex in r_i and some vertex in r_j are adjacent), then it is especially important that regions r_i and r_j be given dissimilar colors. We desire that the colors of such adjacent regions be as dissimilar as possible, subject to the first constraint that all region colors be visually distinct.

To solve this problem, we construct a *region graph* R (as in Figure 2(b)). We form one vertex in R per region r_i, with two regions r_i and r_j connected by an edge in R if and only if they are adjacent regions. We view the problem of assigning colors to the regions as one of embedding R geometrically, into a three-dimensional space representing the gamut of colors available on the display device. Ideally, distances in this space should represent the visual dissimilarity of a pair of colors. As mentioned above, color spaces such as Lab have been designed so that this dissimilarity can be approximated by a Euclidean distance in that space.

Thus, we have a geometric graph embedding problem: assign color coordinates in a color space $\mathcal{C}$ to each vertex of the region graph R, according to the dissimilarity criteria identified above. However, unlike the embedding problems coming from graph drawing problems, we want to place vertices so that edges are long rather than short.

In order to formalize the problem, we define a *coloring* to be any mapping χ from the vertices of R to the color space of interest. Let $d_{i,j}$ denote the distance between $\chi(r_i)$ and $\chi_(r_j)$, as measured by an appropriate distance function corresponding to visual dissimilarity. Let D be the dimension of the color space; in most instances we will have $D = 3$. Let n be the number of vertices in the region graph. For any region r_i, let N_i denote the set of adjacent regions in R. Finally, let Δ denote the diameter of the color space into which we are embedding our region graph. We define a quality measure $q(\chi)$ by the following equation:

$$q(\chi) = \sum_{r_i} \left(\sum_{r_j \in R} \frac{1}{d_{i,j}^{D+1}} + \frac{n^{1+1/D}}{\Delta^D} \sum_{r_j \in N_i} \frac{1}{d_{i,j}|N_i|} \right).$$

One of our goals in defining a function of this form is that, by making the quality a sum of relatively simple terms, we may find its gradient easily, simplifying the application of standard numerical optimization techniques. There are two terms per region in this sum, both normalized to be of roughly equal significance.

The first term is the one having the form $\sum_{r_j \in R} d_{i,j}^{-(D+1)}$. We expect, in a good embedding of the region graph, that the regions will be roughly uniformly distributed around the region graph. The exponent $D+1$ in this term is chosen with this assumption in mind: for infinitely many uniformly spaced regions with a spacing of δ, $\sum d_i, j^{-(D+1)}$ will converge to $\Theta(\delta^{-(D+1)})$, being influenced most strongly by the regions nearest r_i. On the other hand, a similar sum with an exponent of D or less would diverge, and thus lowering the exponent in this term would cause our quality measure to be dominated more by global than local

concerns. For n vertices in a D-dimensional region of diameter Δ, we expect spacing $\delta = \Theta(\Delta n^{-1/D})$, and thus we expect

$$\sum_{r_j \in R} \frac{1}{d_{i,j}^{D+1}} = \Theta(\delta^{-(D+1)}) = \Theta(\frac{n^{1+1/D}}{\Delta^{D+1}}).$$

The second term is the one having the form $\sum_{r_j \in N_i} 1/(d_{i,j}|N_i|)$. We hope, especially in the case of relatively sparse region graphs, to have $d_{i,j}$ roughly proportional to Δ for all edges between adjacent regions r_i and r_j. If these edges are all sufficiently long, the normalization by $|N_i|$ will leave this term roughly proportional to $1/\Delta$. The low exponent on the distance is acceptable as we wish this part of the quality measure to act long-range, causing adjacent regions to be placed far apart. The normalization factor

$$\frac{n^{1+1/D}}{\Delta^D}$$

prior to the second term in our definition of q is chosen to make the two terms of the sum roughly proportional.

5 Finding a Good Coloring

The problem of finding a good coloring can be approached with a standard gradient descent or hill climbing heuristic: choose initial vertex locations in color space, and then gradually move the locations in a direction that causes the most local improvement in our quality measure. Thus, we need to calculate the gradient of our quality measure. In order to do so, it is simplest for our color space to form a normed vector space, preferably Euclidean. Lab color is optimal for this task, as it has been designed in such a way that Euclidean distances in Lab color closely approximate visual dissimilarity. However, the same approach could also be applied directly to RGB-based color spaces, such as sRGB, with some degradation in the goodness of fit between our quality measure and the visual dissimilarity of the resulting colors.

As our quality measure is linear, we may compute the gradient separately for the term of it applying to each region r_i. The gradient at r_i is a vector-valued quantity, formed by summing for each r_j a vector directed away from r_j. if r_i and r_j are not adjacent, this vector has length $(D+1)/d_{i,j}^{D+2}$. If r_i and r_j are adjacent, we add another vector in the same direction with length

$$\frac{n^{1+1/D}}{|N_i|\Delta^D} d_{i,j}^{-2}.$$

We note that gradient descent with these vectors will cause the locations of regions in color space to spread apart rapidly. But we do not allow this to continue unconstrained, as we must confine the colors of each region to the gamut of displayable colors on the intended output device. We considered several options for this confinement:

– We could add an additional term in the quality measure penalizing colors outside the allowable gamut. However, we do not wish to penalize colors near the boundaries of the gamut, because those boundaries provide saturated colors that are easy to visually distinguish. Nor do we wish to allow colors to drift very far beyond the gamut. So the penalty term would have to have a very steep derivative, making the numerical optimization more difficult.
– We considered a form of truncation, in which any color outside the gamut is clipped to the nearest color within the gamut. Like the penalty term, this method would affect only the colors that reach the boundaries of the allowable gamut, and the numerical optimization procedure would have difficulty propagating the effects of this clipping to the interior of the gamut. More significantly, this truncation could end up distorting the points near boundaries of the color space that are not perpendicular to the center of the gamut: effectively, the outward repulsive forces of the gradient descent together with the truncation would cause these points to be pushed along the boundary away from the center.
– We experimented with a procedure that, after each step of gradient descent, rescales the entire color space, so that all vertices again lies within the gamut of allowable colors. This seems to work acceptably well for symmetric color spaces such as the sRGB gamut. However, when we tried it with Lab colors, for which the color space is more stretched out in some directions than others, we found that this method tended to accentuated this stretching, causing the gamut to be compressed in the other directions. In particular, this led to significant desaturation of the resulting Lab colors.
– We finally settled on the following procedure: after each step of gradient descent, rescale (rather than truncating) the out-of-gamut points, while leaving the other points in place. In our experiments this method performed better than the other ones above, allowing the gradient descent to improve the color placement without distorting the gamut.

Our implementation chooses initial vertex locations at random within the color gamut. Then in each iteration it attempts to move the locations of the vertices in color space, one vertex at a time by three types of moves: random jumps, swaps with other vertices, and moving by a fixed step size in the gradient direction. For each of these three move types our algorithm accepts the move only when it improves the overall quality of the coloring. If an iteration fails to find any quality improvement, we reduce the step size, and terminate the algorithm when this step size falls below a preset threshhold.

6 Results of Our Implementation

As a proof of concept, we implemented our algorithm both for the sRGB color space and Lab color, and compared the results with those from an algorithm that chose sRGB colors independently at random for all regions. As the color spaces we use form convex polyhedra with eight vertices, our algorithm will tend to choose colors at those vertices for graphs with eight or fewer regions.

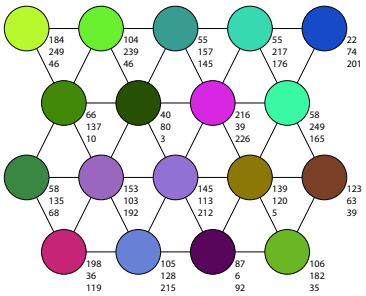

Fig. 3. Results of implementation: random assignment of colors to vertices.

For this region, we chose for our experiments a region graph in the form of an eighteen-vertex triangulation. In each case, once our implementation was working correctly, we generated only one coloring in order to avoid biasing our results by human choices; however the random nature of the starting points for our algorithms means that these precise results would not be repeatable, and more systematic usability testing would be needed to verify our results.

In the first implementation (Figure 3), we chose random colors independently for all vertices, uniformly among the 2^{24} possible sRGB values. As expected, this did not work very well. The random assignment did not prevent several very similar colors from being chosen, often for adjacent regions. We did not feel that the additional complexity of similarly generating Lab colors uniformly at random was likely to lead to significant improvement over this result.

In the second implementation (Figure 4) we applied our gradient descent optimization algorithm directly to the sRGB color space, using the quality measure we defined earlier via the Euclidean distance in this space despite the fact that this distance is known to fit human vision poorly. The algorithm chose a diverse selection of well saturated colors, and all pairs of adjacent regions have easily distinguishable colors. However, there are several nonadjacent colors that are difficult to distinguish: two yellows (254,254,0 and 255,255,145), two cyans (0,255,246 and 140,255,255), three blues (1,129,255, 0,0,245, and 130,0,255), two reds (255,112,99 and 255,0,1), and two pinks (255,4,255 and 255,171,255). We believe these faults are due to the poor match between Euclidean distance in sRGB color space and human visual dissimilarity.

Finally, we applied our gradient descent algorithm for coordinates in the Lab color space (Figure 5). The gamut of representable colors in Lab space is significantly larger than that for sRGB, so in order to ensure that our algorithm generated colors that could be displayed, we restricted all colors to a gamut formed geometrically as the convex hull in Lab space of the eight colors black, white, red, green, blue, cyan, magenta, and yellow forming the most extreme values of the sRGB color space. When our gradient descent algorithm caused vertices to be assigned colors outside this convex hull, as described above, we returned them to the gamut by a rescaling operation centered at the neutral gray color with Lab coordinates $50, 0, 0$. As with the sRGB output, the result

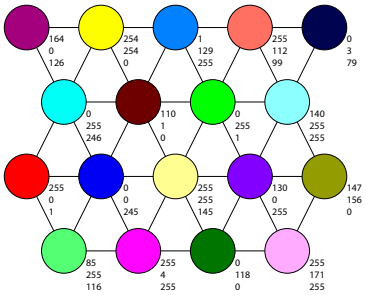

Fig. 4. Results of implementation: gradient descent in sRGB color space.

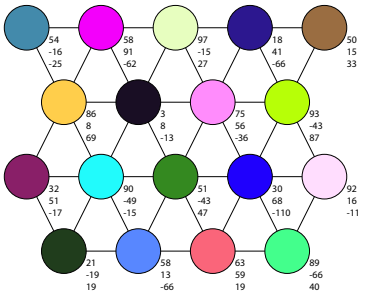

Fig. 5. Results of implementation: gradient descent in Lab color space.

of this algorithm was a collection of diverse well saturated colors, with all pairs of adjacent regions having easily distinguishable colors. Compared to the sRGB results, there were fewer sets of difficult to distinguish nonadjacent colors: primarily the three pinks (58,91,-62, 75,56,-36, and 92,16,-11), the lightest of which is easy to distinguish from the darkest but the other two pairs of which are less easy to distinguish. It is also somewhat difficult to distinguish the dark green (21,-19,19) from the black (3,8,-13). On the whole, it seems that using Lab color has led to a better selection of colors, and equally good assignment of the chosen colors to the vertices of the region graph.

7 Conclusion

We have given what we believe is the first color assignment algorithm that uses adjacency information in the input geometric graph so as to choose colors that are very different for adjacent vertices. For possible future work, one could consider a weighted version of the problem, where edges of the input geometric graph are weighted (e.g., by length) and we wish to assign colors so that the colors assigned to vertices of low-weight edges are more dissimilar than those on high-weight edges. In addition, another interesting adaptation would be to perform our color assignment algorithm for color spaces corresponding to color-blind people (of which there are six types that collectively make up roughly 8% of the male population).

References

1. N. Alon and P. Erdős. Disjoint edges in geometric graphs. *Discrete Comput. Geom.*, 4:287–290, 1989.
2. M. W. Bern, D. Eppstein, and B. Hutchings. Algorithms for coloring quadtrees. *Algorithmica*, 32(1):87–94, 2002.
3. J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Macmillan, London, 1976.
4. C. A. Brewer. Color use guidelines for data representation. In *Proc. Section on Statistical Graphics, American Statistical Association*, pages 55–60, 1999.
5. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.
6. D. Eppstein. Testing bipartiteness of geometric intersection graphs. In *Proc. 15th Symp. Discrete Algorithms*, pages 853–861. ACM and SIAM, 2004.
7. C. G. Healey. Choosing effective colours for data visualization. In R. Yagel and G. M. Nielson, editors, *IEEE Visualization '96*, pages 263–270, 1996.
8. M. Jünger and P. Mutzel. *Graph Drawing Software*. Springer, 2003.
9. M. Kaufmann and D. Wagner. *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.
10. P. Lee and Z. M. Kedem. Automatic data and computation decomposition on distributed memory parallel computers. *ACM Trans. Programming Languages and Systems*, 24(1):1–50, 2002.
11. H. Levkowitz and G. T. Herman. Color scales for image data. *IEEE Computer Graphics and Applications*, 12(1):72–80, 1992.
12. T. Nishizeki. *Planar Graph Drawing*, volume 12 of *LNSC*. World Scientific, 2004.
13. L. Pan, M. K. Lai, K. Noguchi, J. J. Huseynov, L. F. Bic, and M. B. Dillencourt. Distributed parallel computing using Navigational Programming. *Int. J. Parallel Programming*, 32(1):1–37, 2004.
14. L. Pan, J. Xue, M. K. Lai, L. Bic, M. B. Dillencourt, and L. Bic. Toward automatic data distributions for migrating computations. Submitted for publication, 2006.
15. P. Rheingans and B. Tebbs. A tool for dynamic explorations of color mappings. *Computer Graphics*, 24(2):145–146, 1990.
16. P. K. Robertson. Visualizing color gamuts: A user interface for the effective use of perceptual color spaces in data displays. *IEEE Computer Graphics and Applications*, 8(5):50–64, 1988.
17. Y. Rubner, J. Puzicha, C. Tomasi, and J. M. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. *Computer Vision and Image Understanding*, 84:25–43, 2001.
18. M. Stokes, M. Anderson, S. Chandrasekar, and R. Motta. A Standard Default Color Space for the Internet – sRGB. Available online at http://www.w3.org/pub/WWW/Graphics/Color/sRGB.html, 1996.
19. M. Stone. *A Field Guide to Digital Color*. Morgan Kaufmann, 2 edition, 2003.
20. E. R. Tufte. *The Visual Display of Quantative Information*. Graphics Press, Cheshire, Connecticut, 1983.
21. E. R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, Connecticut, 1990.
22. C. Ware. Color sequences for univariate maps: Theory, experiments, and principles. *IEEE Computer Graphics and Applications*, 8(5):41–49, 1988.