A Coloring Algorithm for Disambiguating Graph and Map Drawings

Yifan Hu[1] and Lei Shi[2,⋆]

[1] Yahoo Labs, 111 W 40th St, New York, NY 10018, USA
yifanhu@yahoo.com
[2] SKLCS, Institute of Software, Chinese Academy of Sciences, China
shil@ios.ac.cn

Abstract. Drawings of non-planar graphs always result in edge crossings. When there are many edges crossing at small angles, it is often difficult to follow these edges, because of the multiple visual paths resulted from the crossings that slow down eye movements. In this paper we propose an algorithm that disambiguates the edges with automatic selection of distinctive colors. Our proposed algorithm computes a near optimal color assignment of a dual collision graph, using a novel branch-and-bound procedure applied to a space decomposition of the color gamut. We conduct a user study to establish the effectiveness and limitations of this approach in clarifying drawings of real world graphs and maps.

Keywords: graph drawing, maps, edge coloring, branch-and-bound algorithm.

1 Introduction

Graphs are widely used for depicting relational information among objects. Typically, graphs are visualized as node-link diagrams [1]. In such a representation, edges are shown as straight lines, polylines or splines. Graphs that appear in real world applications are usually non-planar. For such graphs, edge crossings in the layout are unavoidable. It is a commonly accepted principle that the number of edge crossings should be minimized whenever possible, this principle was confirmed by user evaluations which showed that human performance in path-following is negatively correlated to the number of edge crossings [18,21]. Later studies found that the effect of edge crossings varies with the crossing angle. In particular, the task response time decreases as the crossing angle increases, and the rate of decrease levels off when the angle is close to 90 degree [14,15]. This implies that it is important not only to minimize the number of edge crossings, but also to maximize the angle of the crossings. Consequently, generating drawings that give large crossing angles, or even right crossing angles, became an active area of research (e.g., [6]). Nevertheless, for general non-planar graphs, there is no known algorithm that can guarantee large crossing angles for straight line drawings. Therefore, techniques to mitigate the adverse visual effect of small angle crossings are important in practice.

In this paper we propose to use colors to help differentiate edges. Our starting point is an existing layout, and our working assumption is that the graph is to be displayed as

⋆ Supported by China National 973 project 2014CB340301 and NSFC grant 61379088.

C. Duncan and A. Symvonis (Eds.): GD 2014, LNCS 8871, pp. 89–100, 2014.

a static image on paper, or on screen. The motivation comes from users of the Graphviz [10] software. These users were generally happy with the layouts of their graphs, but were asking whether there was any visual instrument that can help them follow edges better. Examining their layouts, we realized that because edges were drawn using the same color (e.g., black), when there were a lot of edge crossings, it was difficult to visually follow these edges. Thus the feedback from our users, and our own observation, echo the findings by Huang *e*t al. [14,15]. When explaining why small crossing angles are detrimental to the task of following a path, they found, with the help of an eye tracking device, that *"when edges cross at small angles, crossings cause confusion, slowing down and triggering extra eye movements."* and that *"in many cases, it is crossings that cause confusion, making all the paths between two nodes, and branches along these paths, unforeseeable. Due to the geometric-path tendency, human eyes can easily slip into the edges that are close to the geometric path but not part of the target path."*.

Edge crossing is not the only hindrance to the visual clarity of a graph drawing. An additional problem is that when an edge from node u passes underneath the label of a node v and connects to a node w, it is impossible to tell visually whether there is one edge $u \leftrightarrow w$, or two edges $u \leftrightarrow v$ and $v \leftrightarrow w$, when all edges are of the same color (e.g., Fig. 3(b)). While these problems can be solved with user interactions by clicking on an edge of interest, or on a node to bring its neighbors closer (see, e.g., [17]), this involves an extra step for the user that may not be necessary if edges can be differentiated with a proper visual cue. Furthermore, there are situations where interaction is not possible, e.g., when looking at a static image of a graph on screen, or in print. These are the situations that are of particular interest in this paper.

We believe all the above mentioned problems of visually distinguishing and following edges can be greatly alleviated by choosing appropriate colors or line styles to differentiate edges. We first identify edge pairs that need to be differentiated (the *colliding edges*), and represent them as nodes of a dual collision graph. We then propose an algorithm to assign colors to the nodes of this collision graph, in a way that maximizes the color difference between nodes that share an edge. Thus our main contributions are:

- A novel branch-and-bound graph coloring algorithm that finds the globally optimal color embedding of each node with regard to its neighbors, and that works with both continuous color spaces and discrete color palettes.
- A user study that establishes the effectiveness/limitations of the coloring approach.

2 Related Work

Graph coloring is a classic problem in algorithmic graph theory. Traditionally the problem is studied in a combinatorial sense. For example, finding the smallest number of k colors on the vertices of a graph so that no two vertices sharing an edge have the same color. The difference between this and our work is that in $k-$colorability problem, a solution is valid as long as any pairs of vertices that share an edge have different colors, no consideration is given to maximizing the actual color differences. So in essence, the distance between colors is binary – either 0, or 1. For our problem we assume that even among distinctive colors, the differences are not equal, and are measured by color distances. In the special case when only k colors are allowed, our algorithm degenerates to find the optimal color assignment among all solutions of the $k-$colorability problem.

This last problem of optimal color assignment was also studied by Gansner *et* al. [9] and by Hu *et* al. [12], in the context of coloring virtual maps to maximize the color difference between neighboring regions. In these works, a set of k distinctive colors are assumed to be given, with k the number of countries in the map. Maps were then colored by an optimal permutation of the k colors. On the other hand, in this paper we assume that the color space can be either continuous or discrete, and we select among all colors in the color space to increase color differences. When applied to map coloring, our algorithm produces k distinctive colors as a side product.

Dillencourt *et* al. [7] studied the problem of coloring geometric graphs so that colors on nodes are as different as possible. The problem they studied is very related to ours, except that in their case the application is the coloring of geometric regions, while we are also interested in coloring edges of a graph. Dillencourt *et* al. used a force-directed gradient descent algorithm to find a *locally optimal* coloring of each node with regard to its neighbors. We propose a new algorithm based on a branch-and-bound process over an octree decomposition of the color space, that finds a *globally optimal* coloring for each node with regard to its neighbors. Furthermore, our approach is more flexible and works for discrete color palettes, in addition to continuous color spaces.

The angular resolution of a drawing is the sharpest angle formed by any two edges that meet at a common vertex.In addition to maximizing crossing angles (e.g., [6]), for the same reason of visual clarity, there have been researches to maximize the angular resolution of the drawing. Most recently, Lombardi Drawing of graphs was proposed [8,3], in which edges are drawn as arcs with perfect angular resolution. However, Purchase *et* al. [20] found that even though users prefer the Lombardi style drawings, straight-line drawings created by a spring-embedder gives better performance for path following and neighbor finding tasks. For straight-line drawings, while it is possible to adjust the layout to improve the angular resolution (e.g., [5,11]), the extent to which this can be done is limited. Although a previous study by Purchase *et* al. [19] did not find sufficient support for maximizing angular resolution, we do find that when two edges connected to the same node are almost on top to each other, it is difficult to tell whether these are two edges or one. For this reason we consider such edges as in collision too.

We note that a nice way to follow an edge, or to find the neighbors of a node, is to use interactive techniques such as “link sliding” and “bring & go” [17]. The algorithm we propose is primarily aimed at disambiguating a static drawing displayed on screen or on paper, it can nevertheless be used in conjunction with such interactive techniques.

Finally, we were made aware of the work of Jianu *et* al. [16] after the completion of this work. Jianu *et* al. [16] proposed a similar idea of using colors to differentiate edges. However there are multiple important differences between that work and ours. The construction of dual collision graph is different: Jianu *et* al. set the edge weights among all edges to be the inverse of either the intersection angle, or the edge distance if the edges do not intersect, which is not optimal since it is perfectly harmless to color edges that have no conflict with the same color. In fact, their method always results in a complete collision graph, making it more expensive for relatively large graphs. Furthermore, because of the complete collision graph, all edges of the original graph must have different colors. Therefore the drawings in [16], which are all of very small graphs, always contain a multitude of colors, which is unnecessary. Our collision graph

almost always contains disconnected components.This decomposes the coloring problem into smaller ones, and allows us to use the same (black) colors for many edges. Jianu et al. [16] solved the coloring problem using a force-directed algorithm, motivated by Dillencourt *et* al. [7]. We were kindly given the source code for [16] from one of the authors. Based on reading the code, we found that it applies force directed algorithm to nodes of the collision graph in the 2D subspace of the LAB color space (the AB subspace). It then sets a fixed L value of 75 (L is the lightness, between 0 to 100). This observation is consistent with the drawings in [16], where black background is used for all drawings due to the high lightness value (see also Fig. 5(d)). This makes the algorithm limited to a small subset of all possible colors. Finally, the force-directed algorithms of Dillencourt *et* al. [7] and Jianu et al. [16] can only be applied to continuous color space in 2D or 3D. Neither works for user specified color palettes, or 1D colors. Our algorithm works for both continuous or discrete color spaces. Overall, we believe that the idea of using colors for disambiguating edges are quite natural to think of. It is how to design appropriate algorithms to make the idea work effectively in practice that is crucial and that differentiates our work and [16]. Furthermore, we present a first user study to evaluate results of our algorithm with real users. The results suggest possible scenarios when our edge coloring approach is effective.

3 The Edge Coloring Problem and a Coloring Algorithm

Appropriate coloring can help greatly in differentiating edges that cross at a small angle. Fig. 1 (left) illustrates such a situation. With many crossing edges, it is difficult to follow the edge from node 19 (top-middle, blue) to node 16 (lower-right, blue). In comparison, in Fig. 1 (right), it is easier to see that 19 is connected to 16 by a blue edge. The objective of this section is to identify situations where ambiguities in following edges can occur, and propose an edge coloring algorithm to resolve such ambiguities.

3.1 Edge Collisions

Two edges are considered in collision if an ambiguity arises when they are drawn using the same color. The following are four conditions for edge collision:

- *C1: they cross at a small angle.*
- *C2: they are connected to the same node at a small angle.*
- *C3 (optional): they are connected to the same node at an angle close to 180 degree.*
- *C4: they do not cross or share a node, but are very close to each other and are almost parallel.*

We now explain the rationale for considering each of these four conditions as being in collision. C1 is considered a collision following the user studies described in Section 1 by Huang *et* al. [14,15]. When eyes try to follow an edge to its destination, small crossing angles between this edge and other edges create multiple paths along the direction of the eye movement, either taking eyes to the wrong path, or slowing down the eye movement. C2 creates a situation where one edge is almost on top of the other, making it difficult to visually follow one of these edges.

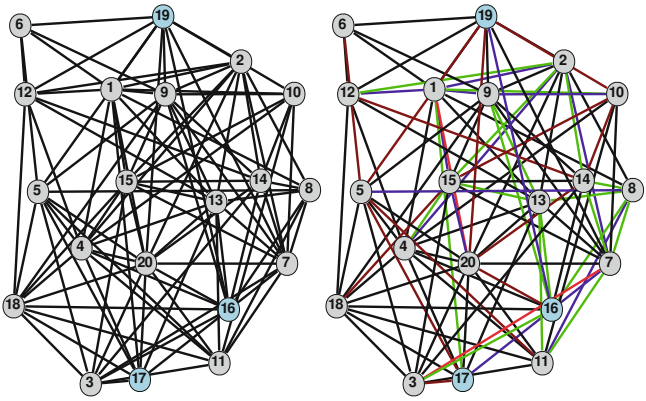


Fig. 1. Left: a graph with 20 nodes and 100 edges. It is difficult to follow some of the edges. For example, is node 19 (blue) connected to node 16 (blue)? Is node 19 connected to 17 (blue)? Right: the same graph, with the edges colored using our algorithm. Now it is easier to see that 19 and 16 are connected by a blue edge, but 19 and 17 are not connected.

C3 could create confusion as to whether the two edges connected at close to 180 degree are one edge, or two edges, when node labels are drawn. For example in Fig. 1 (left), it is difficult to tell whether nodes 19 and 17 are connected, or whether 19 is connected to 20 and 20 is connected to 17. When edges are properly colored (Fig. 1 (right)), it is clear that the latter is true. Note that if edges are allowed to be drawn on top of nodes, then an edge between 19 and 17 would be seen over the label of 20, thus this kind of confusion can be eliminated. Therefore we consider C3 as optional. But drawing edges over the label of nodes introduce extra clutter and make the node labels harder to read.

C4 causes a problem because when two edges are very close and almost parallel, it is difficult to differentiate between them. In addition, it can cause confusion when node labels are drawn. Fig. 3(a) shows two lines very close and almost parallel. While it is possible to differentiate between the two edges, when node labels are added (Fig. 3(b)), it is difficult to tell whether there are two edges ($1 \leftrightarrow 2$ and $3 \leftrightarrow 4$), or three edges ($1 \leftrightarrow 2$, $1 \leftrightarrow 4$ and $1 \leftrightarrow 3$), or whether there even exists an edge $3 \leftrightarrow 2$. This confusion can be avoided if suitable edge coloring is applied (Fig. 3(c)).

To resolve these collisions, we propose to color the edges so that any two edges in collision have as different colors as possible. We first construct a dual edge collision graph.

3.2 Constructing the Dual Collision Graph

Let the original graph be $G = \{V, E\}$. Denote by $N(v)$ the set of neighbors of a node v. The *dual collision graph* is $G_c = \{V_c, E_c\}$, where each node in V_c corresponds to an edge in the original graph. In other words, there is a one-to-one mapping $e : V_c \rightarrow E$. Two nodes of the collision graph i and j are connected if $e(i)$ and $e(j)$ collide in the original graph.

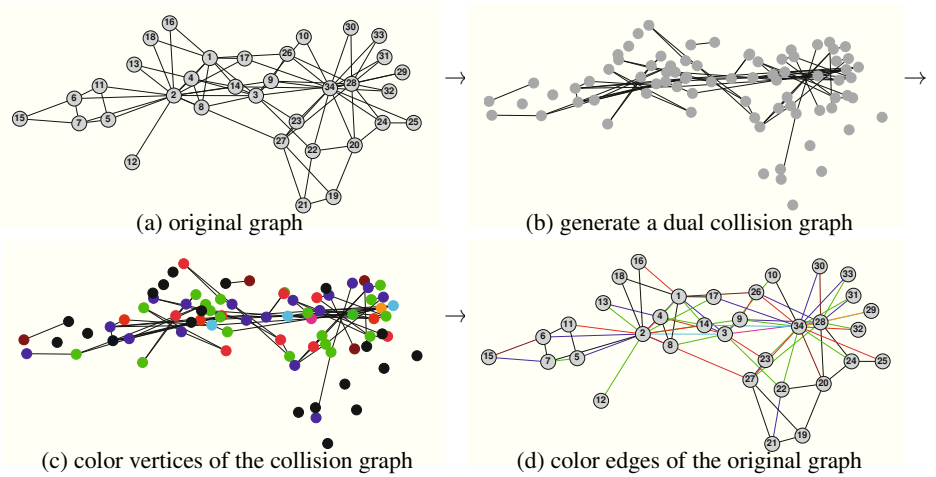


Fig. 2. The proposed pipeline for coloring the edges of the Zachary's Karate Club Graph: (a) the original graph; (b) the dual collision graph, with each node representing an edge of the original graph, and positioned at the center of that edge; (c) the collision graph, with nodes colored to maximize color differences along the edges; (d) the original graph, with edges colored using the node coloring in (c).

The problem of coloring the edges of G then becomes that of coloring nodes of the collision graph G_c. Let $\mathscr{C}$ be the color space, and $c(i) \in \mathscr{C}$ be the color of a node $i \in V_c$, we want to find a coloring scheme such that the color of each node in the collision graph is as different to its neighbors as possible. This task can be posed as a MaxMin optimization problem:

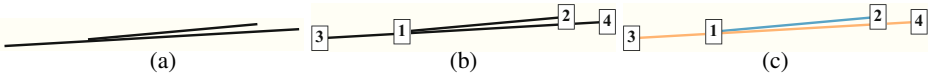


Fig. 3. An illustration of the rationale for collision condition C4. (a) Two edges that do not cross. (b) When nodes are shown, it is difficult to tell if there are two edges ($1 \leftrightarrow 2$ and $3 \leftrightarrow 4$), or three edges ($1 \leftrightarrow 2$, $1 \leftrightarrow 4$ and $1 \leftrightarrow 3$), or whether there even exists an edge $3 \leftrightarrow 2$. (c) After coloring each edges with a distinctive color, it is clear that there are two edges, $1 \leftrightarrow 2$ and $3 \leftrightarrow 4$

$$\arg\max_{c:V_c \to \mathscr{C}} \min_{\{i,j\} \in E_c} w_{ij} \|c(i) - c(j)\|, \tag{1}$$

where $w_{ij} > 0$ is a weight inversely proportional to how important it is to differentiate colors of nodes i and j, and $\|c(i) - c(j)\|$ is a measure of the difference between the colors assigned to the two nodes.

Note that (1) is stated rather generally: $\mathscr{C}$ could be a discrete, or continuous, color space. This is intentional since we are interested in both scenarios. All we assume is that $\mathscr{C}$ sits in a Euclidean space of dimension d.

Once we colored the collision graph, we can use the same coloring scheme for the edges of the original graph. The complete pipeline of our proposed approach is illustrated in Fig. 2. Notice that the collision graph in Fig. 2(b) is disconnected. We apply our algorithm on each component of the collision graph.

3.3 A Color Optimization Algorithm

Dillencourt *et* al. [7] proposed a force-directed algorithm in a Euclidean color space. They wanted *all* pairs of nodes to have distinctively different colors. Consequently their algorithm used a force model where repulsive forces exist among all pairs of nodes.

Because in our case edges can have the same color as long as they do not collide, there is no need to push all pairs of nodes of the collision graph apart in the color space. Therefore we can not use the algorithm of Dillencourt *et* al. [7] as is. Although it is possible to adapt their algorithm, we opt to propose an alternative algorithm. One reason is that we like to be able to use not only continuous color spaces, but also discrete color palettes.Another reason is due to the fact that even when deciding the optimal color for one node of the collision graph with regard to all its neighbors, this seemingly simple problem can have many local maxima.

We give an example to illustrate this point. For simplicity of illustration, within this example, we assume for that our color space is 2D, and that the color distance is the Euclidean distance. Suppose we want to find the best color embedding for a node u in the collision graph with six neighbors, and the six neighbors are currently embedded as shown in Fig. 4 (left). We want to place u as far away from the set of six points as possible. Fig. 4 (left) shows a color contour of the distance from the set of six points (the distance of a point to a set of point is defined as the minimum distance between this point and all the points in the set, assuming unit weighting factors). Color scale is given in the figure, with blue for low values and off-white for large. From the contour plot it is clear that there are seven or more local maxima. In 3D there could be even more local maxima. A force-directed algorithm such as [7], even with the random jumps and swaps, is likely to settle in one of the local maxima.

Instead we hope to find the global maximum. A naive way to find the global maximum position in the color space with regard to a set of points is to search exhaustively by imposing a fine grid over the color space, and calculating the distance from each mesh point to the set. However, given that the color space are typically of three dimensions, even at a resolution of 100 subdivisions along each dimension, we need 10^6 distance calculations. This is computationally too expensive, bear in mind that this computation needs to be performed for each and every node of the collision graph repeatedly until the overall embedding in the color space converges.

We propose a more efficient algorithm based on the octree data structure (quadtree for 2D) that does not require evaluations of the distance over all mesh points. Pseudo code for the algorithm is given in [13]. We give a high level description here. Using Fig. 4 (left) as an example, we want to find a point in the color space that is of maximal distance to a target set of points. Define the objective function value of a square to be

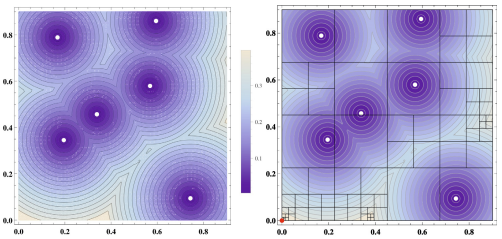


Fig. 4. Left: contour plot of the distance to a set of six (white) points in the space $[0,0.9] \times [0,0.9]$. There are seven or more local maxima. E.g., near $\{0,0.55\}$ and $\{0.4,0.7\}$. Right: an illustration of the quadtree structure generated during our algorithm for finding the global optimal embedding of a node that is farthest away from the set of six points. The final solution is $\{0,0\}$ (red point).

the distance from the center of the square to the target set. We start with a queue of one square covering the color space, and define the current optimal value as the maximal distance over all squares in the queue to the target set. Taking a square from the current queue, we subdivide it into four squares. If the distance of one of the four squares to the point set, plus the distance from the center of the square to a corner of the square, is less than the current optimal distance, this square is discarded. This is because no point in this square can have a larger distance to the target set than the current optimal distance. If the square is outside of the color space, it is also discarded. Otherwise the square is entered into the queue, and the optimal value updated. This continues until the half width of all squares in the queue is smaller than a preset threshold ε. The point that achieves the current optimal value is taken as the optimum. We know that the current optimal value should be within a value $\delta = d^{1/2}\varepsilon$ to the global optimal value, where δ is the half diagonal of the final square in d-dimensional space.

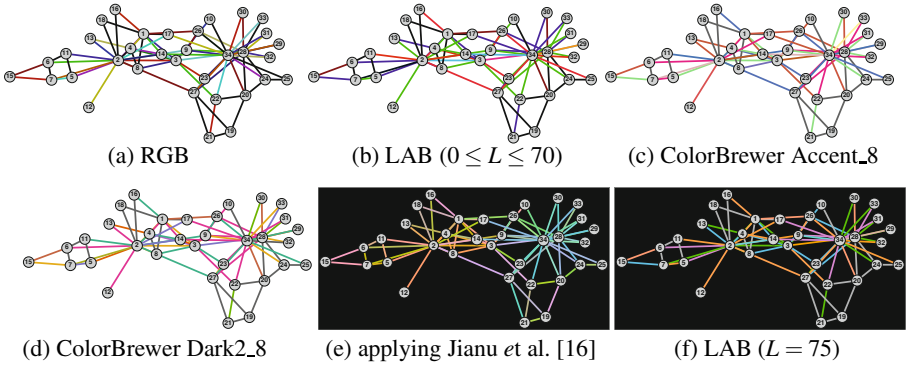
This algorithm is in essence a branch-and-bound algorithm operating on the octree (quadtree for 2D) decomposition of the color space. When applied to the problem in Fig. 4 (right), we can see that in the top-left quadrant, the quadtree branched twice and stopped, because the function values are relatively small in that quadrant. The top-right and bottom-right quadrants branched 3 and 4 times, respectively. The final optimal point is found in the bottom-left quadrant. Initially the algorithm homed in on two regions, one around $\{0.375,0\}$ and the other around $\{0,0\}$, eventually settled around the latter.

Of course this branch-and-bound algorithm only finds the global optimal embedding for one node. After applying the algorithm to every node of the collision graph once (one outer iteration), we repeated if the minimal color difference increases, or if it does not change, but the total sum of color difference across all nodes increases.

We name the algorithm CLARIFY (Edge Coloring for *CLARIFY*ing a Graph Layout) and formally state it in Algorithm 1 in the technical report [13].

4 Implementation and Results

CLARIFY works for both continuous color spaces (RGB and LAB), as well as discrete color space, including a fixed list of colors. Fig. 5 shows examples of applying CLAR-



(a) RGB (b) LAB ($0 \leq L \leq 70$) (c) ColorBrewer Accent_8

(d) ColorBrewer Dark2_8 (e) applying Jianu *et* al. [16] (f) LAB ($L = 75$)

Fig. 5. Applying CLARIFY on the Karate graph in RGB and LAB color spaces (a-b), and with two ColorBrewer palettes (c-d). For comparison we include the result of applying the algorithm of Jianu *et* al. [16], vs CLARIFY in LAB color space with fixed lightness of 75 (e-f).

IFY in the RGB color space, the LAB space with intensity $0 \leq L \leq 70$, and using two ColorBrewer [2] color palettes. In addition it shows how CLARIFY compared favorably with the result of [16]. CLARIFY can also be applied for coloring of virtual maps. Fig. 6 shows an author collaboration map (see [9]) colored using CLARIFY with two color palettes.

Detailed information on implementation, including how CLARIFY is made to work with both continuous and discrete color spaces, are given in the technical report [13]. CLARIFY is now available from Graphviz [10] as `edgepaint` (for edge coloring only; map coloring will be made available soon as part of `gvmap`).

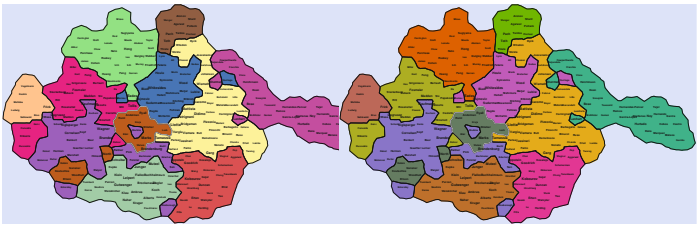


Fig. 6. CLARIFY on a virtual map with two ColorBrewer palettes: left: Accent_8, right: Dark2_8.

We now apply CLARIFY to graphs from real applications. Table 1 gives results on six of the graphs we tested, including running time and objective function (1) (color diff) achieved in LAB color space. These are from [4] or [10]. We intentionally avoided choosing mesh-like graphs – such graphs are easy to layout aesthetically. Their layouts

also tend to exhibit a low perceptual complexity, making it relatively easy to follow edges and paths. Compared with a non-mesh-like graph, a mesh-like graph is easier for our algorithm because there are typically fewer colliding edges. We ran the experiment on a Macbook Pro laptop with a 2.3 GHz Intel Core i7 processor.

Table 1. Statistics on the original and dual test graphs, CPU time (in second) and objective function (cdiff) for CLARIFY. The time in bracket is for constructing the dual collision graph.

graph	$\|V\|$	$\|E\|$	$\|E_c\|$	CPU	cdiff
ngk_4	50	100	54	0.6 (0.)	122.69
NotreDame_yeast	1458	1948	1685	1.3 (0.2)	67.9
GD00_c	638	1020	1847	1.7 (0.1)	64.32
Erdos971	429	1312	4427	2.1 (0.1)	59.3
Harvard500	500	2043	11972	2.3 (0.3)	35.0
extr1	5670	11405	34696	14.5 (7.9)	47.1

It can be seen from Table 1 that for graphs of up to a few thousand nodes and edges, CLARIFY runs quickly. The majority of the CPU time is spent on color assignment, while the construction of the collision graph takes relatively little time even with the naive dual graph construction algorithm. The Harvard500 graph gives a large $|E_c|$ (number of edges in the collision graph) in comparison to the number of edges, because it has a few almost complete subgraphs, which results in a lot of crossings at small angles.

5 User Study

We conducted a controlled experiment to study the effect of edge coloring on user's performance in fundamental graph-related tasks, such as visually following edges, finding neighbors and calculating the shortest path. Generally we compared two approaches, defined as two visualization types: the baseline graph drawing in black-white (B/W) and the improved graph drawing with edges colored by our algorithm (Color).

Experiment Design. We recruited 12 participants (8 male, 4 female) for this paper-and-pencil experiment. 10 of the participants were graduate students majoring computer science and the other 2 of them were department assistants with no technology background. Half of the participants had experiences on node-link graphs, one student was even an expert on graphs. The other half did not have previous knowledge with the node-link graph. The experiment followed a within-subject design with every participant doing all tasks with both visualization types. To eliminate the learning effect over the same task, we used two different layouts of the same graph data. We had a full factorial deign on the choice of two visualization types and two graph layouts. Each participant entered the same task four times in total. The experiment order was randomized across participants. Half of them completed the tasks first with the B/W approach and then with the Color approach. Another half adopted the opposite order. Further, in half of the time when participants were given the colored drawing, the algorithm is fixed to

use the LAB palette. In another half, the participants selected their favorite palette and completed tasks with the colored drawings generated by this palette.

Data and Task. Two layouts of the Zachary's Karate Club Graph were used. One was exactly the layout in Fig. 2. Another was rotated and re-labeled. Three types of graph-related tasks were designed:

T1 (Connectivity): Determine whether two nodes are connected by a direct edge;

T2 (Neighbor): Estimate the number of nodes a particular node connects directly;

T3 (Path): Estimate the minimum number of hops from a particular node to another, including the source and destination.

On each type, four tasks were selected on each graph layout with similar difficulty levels. To eliminate user's visual node querying time from their task completion time, we annotated the related nodes in each task on the corresponding graph layout before participants took the task.

Result. Results were analyzed separately on each task type. Detailed analysis and error bar charts are given in the technical report [13]. The major findings are that on connectivity tasks, the average task error of the Color group is less than 30% of the B/W group, and is statistically significant. Performance difference on neighbor/path tasks and color palettes were not statistically significant.

6 Conclusions

Edge crossings, particularly those at small crossing angles, are known to be detrimental to the visual understanding of graph drawings. This paper proposes an edge coloring algorithm for disambiguating edges that are in collision because of small crossing angles or partial overlaps. The algorithm, based on a branch-and-bound procedure applied to a space decomposition of the color gamut, generates color assignments that maximize color differences of the colliding edges, and works for both continuous color space and discrete color palettes. The algorithm can also be applied to generate coloring for disambiguating virtual maps. Our user study found that coloring edges in graph drawings helped user's performance in 1-hop graph connectivity task significantly. Consequently we have made the CLARIFY code available as part of Graphviz open source software.

The approach of coloring edges for disambiguating drawings has its limitations. Our working assumption is that the drawing is to be displayed as a static image on paper or screen. When an interactive environment is available, techniques such as "link sliding" and "bring & go" [17] could be more effective. In such a situation, the algorithms proposed here can be used as an additional visual aid to the interaction.

While the algorithm proposed here can run on relatively large graphs, our experience is that for graphs with a lot of edges, a static image is insufficient to allow the user to clearly see and follow each edge. Therefore our approach is best suited for small- to medium- sized graphs. Typical usage scenarios are illustrations of diagrams, such as computer or biological networks.

Finally, we note that sometimes edge colors are used to encode attributes on the edges. To apply our approach without interfering with the need to display such attributes, edges can be differentiated using dashed lines of different style and/or thickness, using CLARIFY through mapping different line styles to 1D or 2D spaces.

References

1. Battista, G.D., Eades, P., Tamassia, R., Tollis, I.G.: Algorithms for the Visualization of Graphs. Prentice-Hall (1999)
2. Brewer, C.: ColorBrewer - Color Advice for Maps, http://www.colorbrewer2.org
3. Chernobelskiy, R., Cunningham, K.I., Goodrich, M.T., Kobourov, S.G., Trott, L.: Force-directed lombardi-style graph drawing. In: Speckmann, B. (ed.) GD 2011. LNCS, vol. 7034, pp. 320–331. Springer, Heidelberg (2011)
4. Davis, T.A., Hu, Y.: University of Florida Sparse Matrix Collection. ACM Transaction on Mathematical Software 38, 1–18 (2011)
5. Di Battista, G., Vismara, L.: Angles of planar triangular graphs. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, STOC 1993, pp. 431–437. ACM, New York (1993)
6. Didimo, W., Eades, P., Liotta, G.: Drawing graphs with right angle crossings. Theor. Comput. Sci. 412(39), 5156–5166 (2011)
7. Dillencourt, M.B., Eppstein, D., Goodrich, M.T.: Choosing colors for geometric graphs via color space embeddings. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 294–305. Springer, Heidelberg (2007)
8. Duncan, C., Eppstein, D., Goodrich, M.T., Kobourov, S., Nöllenburg, M.: Lombardi drawings of graphs. J. Graph Algorithms and Applications 16, 85–108 (2012)
9. Gansner, E.R., Hu, Y., Kobourov, S.: Visualizing Graphs and Clusters as Maps. IEEE Computer Graphics and Applications 30, 54–66 (2010)
10. Gansner, E.R., North, S.: An open graph visualization system and its applications to software engineering. Software - Practice & Experience 30, 1203–1233 (2000)
11. Garg, A., Tamassia, R.: Planar drawings and angular resolution: Algorithms and bounds. In: van Leeuwen, J. (ed.) ESA 1994. LNCS, vol. 855, pp. 12–23. Springer, Heidelberg (1994)
12. Hu, Y., Kobourov, S., Veeramoni, S.: On maximum differential graph coloring. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 274–286. Springer, Heidelberg (2011)
13. Hu, Y., Shi, L. (2014), http://arxiv.org/abs/1409.0436
14. Huang, W.: Using eye tracking to investigate graph layout effects. In: 2007 6th International Asia-Pacific Symposium on Visualization, APVIS 2007, pp. 97–100 (2007)
15. Huang, W., Hong, S.-H., Eades, P.: Effects of crossing angles. In: Proceedings of IEEE Pacific Visualization Symposium, pp. 41–46. IEEE (2008)
16. Jianu, R., Rusu, A., Fabian, A.J., Laidlaw, D.H.: A coloring solution to the edge crossing problem. In: Proceedings of the 13th International Conference in Information Visualization (iV 2009), pp. 691–696. IEEE Computer Society (2009)
17. Moscovich, T., Chevalier, F., Henry, N., Pietriga, E., Fekete, J.: Topology-aware navigation in large networks. In: CHI 2009: Proceedings of the 27th International Conference on Human Factors in Computing Systems, pp. 2319–2328. ACM, New York (2009)
18. Purchase, H.C.: Which aesthetic has the greatest effect on human understanding? In: DiBattista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 248–261. Springer, Heidelberg (1997)
19. Purchase, H.C., Carrington, D., Allder, J.-A.: Experimenting with aesthetics-based graph layout. In: Anderson, M., Cheng, P., Haarslev, V. (eds.) Diagrams 2000. LNCS (LNAI), vol. 1889, pp. 498–501. Springer, Heidelberg (2000)
20. Purchase, H.C., Hamer, J., Nöllenburg, M., Kobourov, S.G.: On the usability of lombardi graph drawings. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 451–462. Springer, Heidelberg (2013)
21. Ware, C., Purchase, H., Colpoys, L., McGill, M.: Cognitive measurements of graph aesthetics. Information Visualization 1(2), 103–110 (2002)