A Coloring Algorithm for Disambiguating Graph and Map Drawings

Yifan Hu, Lei Shi, and Qingsong Liu

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 give examples demonstrating this approach in real world graphs and maps, as well as a user study to establish its effectiveness and limitations.

Index Terms—Graph drawing, virtual maps, edge coloring, branch-and-bound algorithm, global optimization

1 Introduction

Graphs are widely used to depict 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 [27], [30]. 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 [20], [21]. 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., [7]). Nevertheless, for general non-planar graphs, there is no known algorithm that can guarantee large crossing angles for straight line drawings [7]. 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 we assume that the graph is to be displayed as a static image on paper, or on screen. The motivation comes from users of the Graphviz [14] software. These users were generally happy with the layouts of their graphs, but asked whether there was any visual instrument that could help them follow edges better. Examining their layouts, we realized that because edges were drawn using the same color (e.g., black), it was difficult to visually follow these edges when there were a lot of edge crossings. The feedback from our users, and our own observation, echo the findings by Huang et al. [20], [21]. 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. We denote by the term *label* the drawing of a node, including the text label. 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 visually tell 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. 3b). 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., [23]). However, doing so 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 problems of visually distinguishing and following edges mentioned above can be greatly alleviated by choosing appropriate colors or line styles to

- *Y. Hu is with the Yahoo Research, Sunnyvale, CA 94089. E-mail: yifanhu@yahoo.com.*
- *L. Shi and Q. Liu are with the SKLCS, Institute of Software, Chinese Academy of Sciences and UCAS, Beijing 100049, China. E-mail: {shil, liuqs}@ios.ac.cn.*

Manuscript received 17 July 2017; revised 16 Jan. 2018; accepted 21 Jan. 2018. Date of publication 25 Jan. 2018; date of current version 31 Dec. 2018. (Corresponding author: Lei Shi.)
Recommended for acceptance by B. Lee.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2018.2798631

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 that maximizes the color difference between nodes that share an edge. Our main contributions are:

- An approach for establishing a dual graph among colliding edges/regions, and coloring the nodes of the dual graph to disambugate graph/map drawings.
- A novel branch-and-bound graph coloring algorithm that finds the globally optimal color embedding of each node with regard to its neighbors. The algorithm works with both continuous color spaces and user-defined color palettes, and can be applied to graphs with straight or curved edges, and to maps.
- A user study that establishes the effectiveness of the coloring approach, as well as its limitations.

2 Related Work

2.1 Color Assignment

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 the 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. 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. They are measured by distances in the color space. 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. [12] and Hu et al. [18], in the context of coloring virtual maps to maximize the color differences between neighboring regions. In these works, a set of exactly k distinctive colors are assumed to be given, with k being the number of countries in the map. The map was 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 maximize color differences. When applied to map coloring, our algorithm produces k distinctive colors as a side product.

Dillencourt et al. [8] 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, whereas 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 respect 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 respect to its neighbors. Furthermore, our approach is more flexible and works for discrete color palettes, in addition to continuous color spaces.

2.2 Crossing Angles

Given the finding by Huang et al. [20], [21] that edge crossings at close to 90 degree hamper human performance less than those at smaller angles, there are active researches in the so called Right-Angle Crossing (RAC) drawings of graphs. In such a drawing, edges cross at right angles (e.g., [7]). This is a practice employed in hand- and algorithm-drawn metro maps as well (e.g., [31]). However, it was shown [7] that a straight-line RAC drawing can have at most $4n - 10$ edges, with n the number of vertices. As far as we are aware, even that is only a necessary, but not sufficient, condition. Therefore, techniques to help alleviate the effect of small angle crossings when RAC or larger angle drawings are not feasible are important in practice.

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., [7]), there have been research efforts in maximizing the angular resolution in order to improve visual clarity. Most recently, Lombardi Drawings of graphs was proposed [3], [9], in which edges are drawn as arcs with perfect angular resolution. However, Purchase et al. [29] 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., [6], [15]), the extent to which this can be done is limited. Although a previous study by Purchase et al. [28] did not find sufficient support for maximizing angular resolution, we 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 to be in collision as well.

2.3 Edge Bundling and Edge Coloring

Edge bundling is another useful tool for decluttering drawings of complex graphs [4], [11], [13], [16], [17]. However when edges are bundled, it is no longer possible to follow an individual edge to its exact destination. Pupyrev et al. [26] proposed to separate edges belonging to the same bundle by a small gap. While this makes it possible in theory to follow individual edges, in practice the edges in each bundle are drawn very close to each other. We believe whether fully bundled, or separated by a small amount, bundled or routed edges can benefit from using colors to differentiate among them (see Fig. 8). Peltonen and Lin [25] devised an alternative approach to coloring edges in the context of edge bundling: they made the color distance among bundled edges proportional to the sum of euclidean distances of the end points. They solved a multidimensional scaling (MDS) problem to embed each bundled edge in a lower dimensional color space. The MDS process minimizes the sum of the square of differences between a pair of edges in the color space, and their distance in the layout space. The minimization underweights edge pairs that have similar starting and ending points. The resulting embedding colors edge bundles starting from and ending in the same regions with similar colors. This is contrary to our use case where such edges will be assigned distinctive colors, and makes it difficult to follow a particular edge. The paper highlighted other differences from our original conference paper [19]. In addition, the

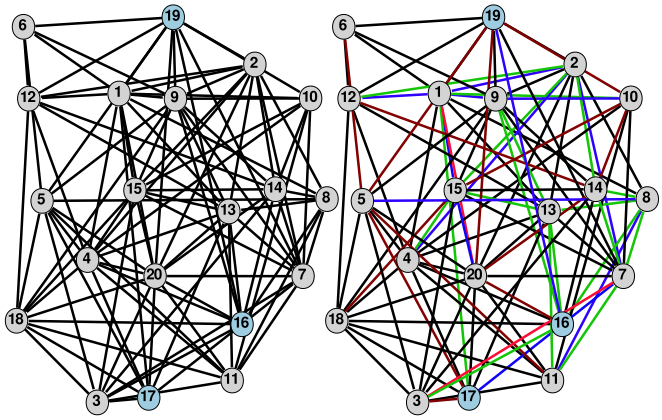


Fig. 1. Left: A graph with 20 nodes and 100 edges. It is difficult to follow some of the edges. For example, is node 19 (blue) connected to node 16 (blue)? Is node 19 connected to 17 (blue)? Right: The same graph, with the edges colored using our algorithm. Now it is easier to see that 19 and 16 are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

optimization problem solved in [25] is unconstrained and without consideration of the shape of the color space. An affine transformation is applied afterwards to find the final embedding in the color space. On the other hand, we optimize directly in the color space, thus potentially have more freedom in choosing the right colors. Furthermore, unlike our algorithm, the algorithm proposed in [25] does not work with discrete palettes.

As far as we know, Jianu et al. [22] were the first to propose the idea of using colors to differentiate edges. However, we believe our work is substantially different and better. Jianu et al. considered all pairs of edges, and 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. This is sub-optimal in the resulting coloring since it is perfectly harmless to color edges that have no conflict with the same color. In our work, we propose a sparse dual collision graph, constructed based on four collision conditions. Using our terminology, the approach of Jianu et al. always results in a complete collision graph, making it inefficient other than for very small graphs. Furthermore, because they consider all edge pairs, every edge of the original graph ends up with a unique color. Therefore the drawings in [22], which are all of very small graphs, always contain a multitude of colors, which is unnecessary. Our collision graph almost always contains disconnected components (e.g., Fig. 4). This decomposes the coloring problem into smaller ones, and allows us to use the same (black) colors for many edges. Jianu et al. [22] solved the coloring problem using a force-directed algorithm, similar to Dillencourt et al. [8]. We were kindly given the source code for [22] 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 [22], where black background is used for all drawings due to the high lightness value (see also Fig. 6d). This makes the algorithm limited to a small subset of all possible colors. Furthermore, the force-directed algorithms of Dillencourt et al. [8] and Jianu et al. [22] 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. What differentiates our work from [22] is the design of an appropriate algorithm that makes the idea work efficiently and effectively in practice, and for continuous and discrete color spaces. Finally, we present the first user study which evaluates the results of our algorithm. The results suggested possible scenarios when edge coloring is effective, and demonstrated that our proposed algorithm is more effective than that of Jianu et al.

This paper was originally published as a conference paper [19]; the journal version adds substantial new materials including a new user study, and expands the algorithm and results sections.

3 The Edge Coloring Problem and a Coloring Algorithm

Appropriate coloring can be of great help in differentiating edges that cross at a small angle. Fig. 1 (left) illustrates such a situation. It is difficult to follow the edge from the blue node 19 to the blue node 16. In comparison, in Fig. 1 (right), it is easier to see that they are connected via 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.

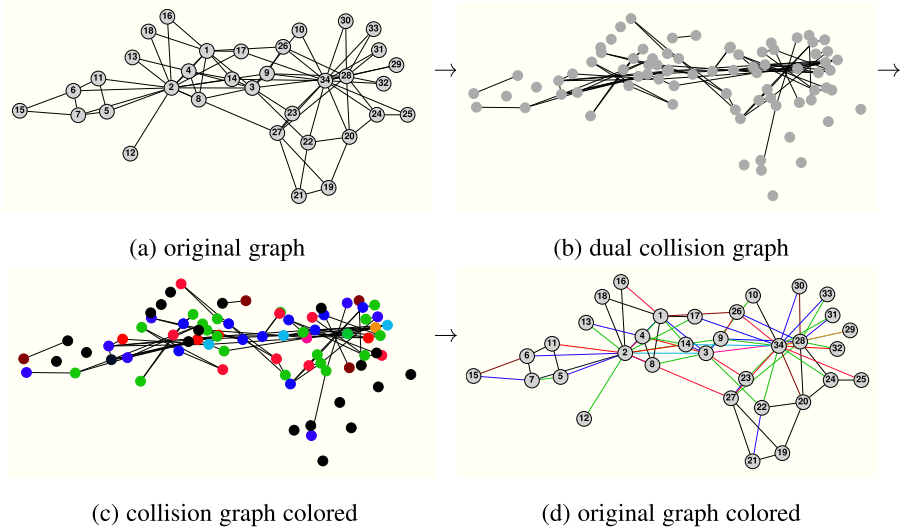
3.1 Edge Collisions

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. [20], [21]. 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.

C3 could create confusion as to whether the two edges connected at close to 180 degree behind a node label are one edge, or two edges. 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. However drawing



(a) original graph (b) dual collision graph

(c) collision graph colored (d) original graph colored

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 dual collision graph, with nodes colored to maximize color differences along the edges (see Fig. 4 for a clearer force-directed layout of this graph); (d) the original graph, with edges colored using the node coloring in (c).

edges over the label of nodes introduces 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. 3a shows two lines very close and almost parallel. While it is possible to differentiate between the two edges, when node labels are added (Fig. 3b), 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. 3c).

To resolve these collisions, we propose to color the edges so that any two edges in collision have colors that differ as much 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.

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 seek to maximize the *minimum color difference* between all pairs of neighboring nodes in the collision graph

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

where $w_{ij} \geq 1$ 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 we do not seek to maximize the *average color difference*, because that could lead to a situation where the average is optimized at the expense of some neighboring nodes being colored with similar colors. However, if the layout is in such a way that most of the edges collide with each other, then minimizing the average color differences may be more appropriate since at least on average, the color difference would be large. In practice, we have not found this to be necessary. Typically, the collision graphs are sparse and disconnected, e.g., Fig. 4.

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. 2b (displayed more clearly using a force-directed layout in Fig. 4) is disconnected. We apply our algorithm on each component of the collision graph.

3.3 A Color Optimization Algorithm

Dillencourt et al. [8] 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.

Since 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

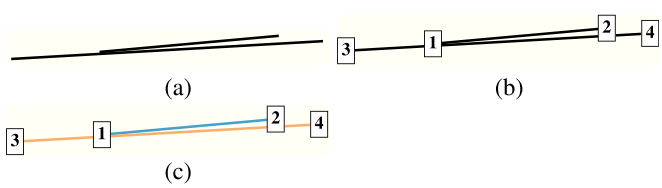


Fig. 3. An illustration of the rationale for collision condition C4. (a) Two edges that do not cross. (b) When nodes are shown, it is difficult to tell if there are two edges ($1 \leftrightarrow 2$ and $3 \leftrightarrow 4$), or three edges ($1 \leftrightarrow 2$, $1 \leftrightarrow 4$ and $1 \leftrightarrow 3$), 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 $3 \leftrightarrow 4$.

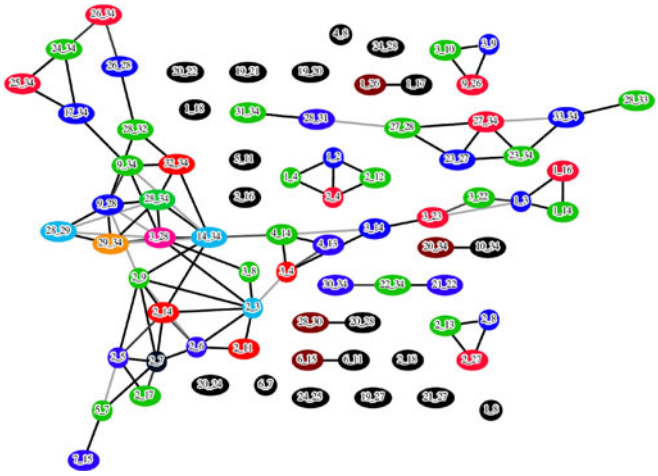


Fig. 4. The collision graph in Fig. 2c, with a force-directed layout. A node labeled "i_j" represents edge $i \leftrightarrow j$ in the original graph. Nodes are colored using Algorithm 1, so that each node is colored as differently from its neighbors as possible. To disambiguate edges we color them in gray scale.

can not use the algorithm of Dillencourt et al. [8] as is. Although it is possible to adapt their algorithm, we opt to propose an alternative algorithm for two reasons. One is that we would like to be able to use not only continuous color spaces, but also discrete color palettes (Section 4.1). The other 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. These two reasons mean that a force-directed algorithm, which operates in a continuous space and is known to be prone to get stuck in local minimum, is not ideal.

Here, we give an example to illustrate the second reason. For simplicity of illustration, within this example, we assume 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. 5 (left). We want to place u as far away from the set of six points as possible. Fig. 5 (left) shows a color contour of the distance from the set of six points. 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 [8], 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 convergence.

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. Using

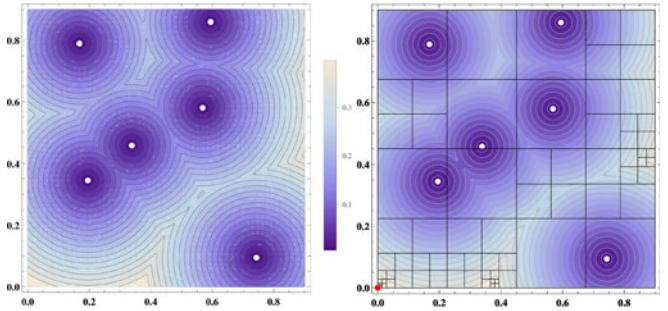


Fig. 5. Left: Contour plot of the distance to a set of six (white) points in the space $[0, 0.9] \times [0, 0.9]$. There are seven or more local maxima. E.g., near $\{0, 0.55\}$, $\{0.35, 0.9\}$ and $\{0.4, 0.7\}$. Right: An illustration of the quadtree structure generated during our algorithm for finding the global optimal embedding of a node that is farthest away from the set of six points. The final solution is $\{0, 0\}$ (shown as the red point).

Fig. 5 (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. We define the objective function value of a square to be 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 $d^{1/2}\epsilon$ to the global optimal value, where $d^{1/2}\epsilon$ 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. 5 (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 repeat if the minimal color difference increases, or if it does not change, but the total sum of color difference across all nodes increases.

We have named the algorithm CLARIFY (Edge Coloring for *CLARIFY*ing a Graph Layout) and formally state it in Algorithm 1. The following are the notations used in the presentation of the algorithm: for a point x and a finite point set C in the euclidean color space $\mathscr{C}$, we define the point-set distance as $\mathrm{dist}(x, C) = \min_{y \in C} w_{i(x),i(y)} \|x - y\|_2$, where $i(x)$

is edge index that corresponds to the point x. We denote the center of a square or cube s as $\mathrm{center}(s)$, its children (by dividing a square into 4 or a cube into 8) as $\mathrm{children}(s)$, and its *half width* as $\delta(s)$. We define the distance between s and a set of point C as that between the center of s and C, that is

$$\mathrm{dist}(s, C) = \mathrm{dist}\left(\mathrm{center}\left(s\right), C\right).$$

The CLARIFY algorithm utilizes the global optimization algorithm for embedding one node, given in Algorithm 2 as EmbedOneNode.

Algorithm 1. CLARIFY$(G, \mathscr{C}, \epsilon)$

input: graph $G = \{V, E\}$, color space $\mathscr{C}$, threshold ϵ
compute a dual collision graph $G_c = \{E_c, V_c\}$ of G
randomly choose $c(i)$ in $\mathscr{C}$ for all $i \in V_c$
set: $\mathrm{mindist} = 0$, $\mathrm{sumdist} = 0$
repeat
  set: $\mathrm{mindist}_{\mathrm{old}} \leftarrow \mathrm{mindist}$, $\mathrm{sumdist}_{\mathrm{old}} = \mathrm{sumdist}$
    $\mathrm{mindist} = \infty$, $\mathrm{sumdist} = 0$
  for $i \in V_c$
    define $c(N(i)) := \{c(j) | j \in N(i)\}$
    define $w_{max} = \max_{j \in N(i)} w_{ij}$
    $c(i) = \mathrm{EmbedOneNode}(c(N(i)), w_{max}, \epsilon)$
    $\mathrm{mindist} = \min\{\mathrm{mindist},\ \mathrm{dist}(c(i), c(N(i)))\}$
    $\mathrm{sumdist}$ += $\mathrm{dist}(c(i), c(N(i)))$
until($\mathrm{mindist} < \mathrm{mindist}_{\mathrm{old}}$ ||
    ($\mathrm{mindist} = \mathrm{mindist}_{\mathrm{old}}$ && $\mathrm{sumdist} \leq \mathrm{sumdist}_{\mathrm{old}}$))
return: $c(e(i)) = c(i),\ i \in V_c$

Algorithm 2. EmbedOneNode(C, w_{max}, ϵ)

input: a set of points $C \in \mathscr{C}$, max weight w_{max}, a threshold ϵ
set: s a square/cube covering the color space $\mathscr{C}$
set: a first-in-first-out queue $Q = \{s\}$
set: $c^* = \mathrm{center}(s)$ and $\mathrm{dist}^* = \mathrm{dist}(s, C)$
for $s \in Q$
  if $\delta(s) < \epsilon$ break
  $Q := Q - \{s\}$
  for $t \in \mathrm{children}(s)$
    if $t \cap \mathscr{C} = \emptyset \mid \mathrm{dist}(t, C) + w_{max} d^{1/2} \delta(t) < \mathrm{dist}^*$
      continue
    if $\mathrm{dist}(t, C) > \mathrm{dist}^*$
      $c^* = \mathrm{center}(t)$, $\mathrm{dist}^* = \mathrm{dist}(t, C)$
    $Q := Q \cup \{t\}$
return: c^*

4 Implementation and Results

We now give details on the implementation of CLARIFY, and results of using the algorithm on real world graphs.

4.1 Color Space

CLARIFY works for both continuous color spaces (as long as it is a metric space), as well as discrete ones.

The RGB Color Space. An often used color model is RGB. This model defines color by a combination of three color intensities, red, green, and blue. Thus colors in the RGB model can be considered as residing in a three-dimensional cube.

RGB color model is widely used for the representing and displaying images in electronic systems, such as LCD/LED display. However, distance between two colors in the RGB space is not an accurate measure of perceived difference by human eyes. For that purpose, the LAB color model is consider better [10].

The LAB Color Space. The LAB color space (a rectangular box $[0, 100] \times [-128, 128] \times [-128, 128]$) includes all perceivable colors, and more. We only care about the LAB color gamut—the part of the LAB space that corresponds to the RGB space. It has a complex shape. Applying CLARIFY requires checking whether a cube is outside of the LAB gamut, which is considerably more complicated than checking whether a point is outside of the gamut.

Instead, since CLARIFY works just as well on a *discrete* set of colors, we modify CLARIFY slightly as follows. We first sample the LAB gamut: we subdivide L, A and B at one unit increment, and check whether the resulting points are inside the LAB gamut by converting the point to RGB space, and back to the LAB space. If the double-conversion ends at the same point (within a threshold of 0.02 in euclidean distance), the point is considered inside the LAB gamut. This resulted in 826,816 points (12.4 percent of the LAB space). Note that we only have to find this sample set once and store as a file. We then construct an octree over this point set. The CLARIFY algorithm works with this octree, without worrying about staying inside the LAB gamut. This sampling technique also makes it very easy to control the lightness of the color—if we need to display the drawing in a dark background and thus light colors are desired, we can simply filter out points with a low L value in the sample. Fig. 6b shows the result of apply CLARIFY in the LAB space with $0 \leq L \leq 70$.

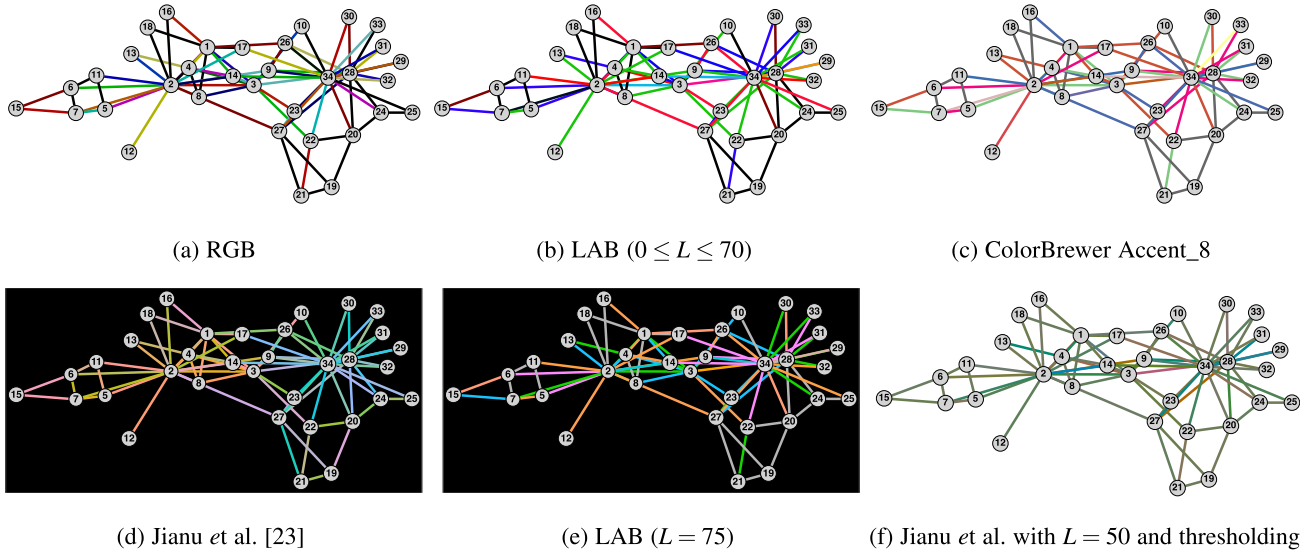
In terms of CPU time, we found that working in LAB space with the sampling technique gives very similar CPU time to working in the RGB space. Speed can be further improved if we take a coarser sample.

User-Define Color Palettes. Any user defined color palette can be handled in a similar way to the LAB gamut—we convert the color palette consists of k colors to the LAB space, then interpolate these k colors to get K sample points. We do so by subdividing the path linking these k points in the LAB space into $K - 1$ segments of equal distance. The path can be constructed along a natural ordering of the palette, or along a shortest path/tour by solving a Traveling Salesman Problem in 3D. An octree is then constructed using the K sample points and CLARIFY is applied over the octree. Fig. 6 gives some examples of using two a ColorBrewer [2] color palette, with $K = 10^4$.

4.2 Complexity of the CLARIFY Algorithm

The CLARIFY algorithm consists of two main steps: finding the dual collision graph, and computing a color assignment.

The collision graph is calculated by checking whether edge pairs are in collision. Conditions C2 and C3 can be checked by looping through each node of the original graph, and testing if a pair of edges starting from the node nearly overlap, or run in almost opposite directions. This check can be done after sorting the angles, hence on a node with l neighbors, assuming that the edges are not entirely on top of each other, the cost should be around $l \log(l)$, so the cost of checking over all nodes is $|E| \log |E|$ (the pathological case of all edges on top of each other would give a complete collision graph thus a complexity of $|E|^2$).



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

(d) Jianu *et* al. [23] (e) LAB ($L = 75$) (f) Jianu *et* al. with $L = 50$ and thresholding

Fig. 6. Applying CLARIFY on the Karate graph in RGB and LAB color spaces (a-b), and with a ColorBrewer palettes (c). For comparison we include the result of applying the algorithm of Jianu et al. [22], versus CLARIFY in LAB color space with fixed lightness of 75 (d-e). Finally we modified the Jianu et al. [22] code to use a white background and dropped pairs of edges with closeness metric less than 0.1 (f).

Condition C1 can be checked using the Bentley-Ottmann algorithm [24] with a complexity of $O((|E| + k)\log|E|)$, where k is the number of edge crossings. If k is $|E|^2$ or more, a naive algorithm which checks all $|E|^2/2$ edges should be used. We are not aware of a good algorithm for checking C4, one possibility is to replace each edge with a rectangle in the shape of a thicker edge, then apply the Bentley-Ottmann algorithm, which should give us the same complexity as checking C1.

The second step of CLARIFY, that of assigning colors, applies the EmbedOneNode algorithm repeatedly over all nodes. EmbedOneNode is a branch-and-bound algorithm over an octree data structure. Its complexity is dependent on the number of local maxima, and how close they are to the global maximum (in terms of the objective function value). If the local maxima have much smaller function values compared with the global maximum, as in the case of Fig. 5, then branches of the octree/quadtree corresponding to the local maxima will terminate at an early stage, and the complexity of the algorithm is around $|\log(\epsilon)|$, otherwise the complexity is around $L * |\log(\epsilon)|$ where L is the average number of local maxima and ϵ the half width of the smallest cube in the octree structure. Overall the worst case complexity is $O(|E||\log(\epsilon)|L)$ per iteration over all nodes. L is a value hard to quantify, we believe it is related to the average degree of the collision graph.

Taking both the collision graph formation and the optimization into account, the CLARIFY algorithm has an average case complexity of $O((|E| + k)\log|E| + t|E||\log(\epsilon)|L)$, with k the number of edge crossing, L the average number of local maxima, and t the number of iterations of Algorithm 1. The worst case complexity is $O(|E|^2 + t|E||\log(\epsilon)|L)$, in the pathological situation where all edges are on top of each other.

In practice we found that the optimization step dominates the computation time even when we use the naive algorithm for computing the collision graph (see Table 2). Therefore for the rest of the paper we use the naive algorithm for the first step of forming the collision graph, which makes computation of C4 much simpler.

4.3 Choice of Parameters

For all our experiments on graphs, we take $w_{ij} = 1$ in Equation (1). For maps, we use non-unit weights as described in Section 4.4. To check collision conditions, we need to define what is a "small angle" and what is "close to 180 degree." Based on visual observations by the authors, for the rest of this paper we set these to be 15 degree and 165 degree. We define two lines being "very close" if the smallest distance between two points on the lines is less than 1 percent of the larger of the length of the lines. We consider two lines as "almost parallel" if they form an angle that is less than one degree. The parameter ϵ controls the accuracy with which we find the global optimal embedding for one node. Table 1 shows the effect of this parameter on the color difference achieved, as well as on CPU time. Clearly the CPU time increases almost linearly with $|\log(\epsilon)|$, as predicted by the complexity analysis. The color difference is also in-line with expectation: from ϵ to $\epsilon/10$, it changes roughly proportionally to $d^{1/2}\epsilon$ or less, where $d = 3$ is the dimension of the color space. This fits our analysis in Section 3.3. Our own visual observation convinced us that $\epsilon = 10^{-2}$ gives very similar coloring to $\epsilon = 10^{-3}$, hence we set $\epsilon = 10^{-2}$ by default.

4.4 Examples

We now apply CLARIFY to graphs from real applications (additional examples are at http://yifanhu.net/EdgeColoring).

TABLE 1
Effect of ϵ on the Color Difference and CPU Time When Applying CLARIFY (Ten Random Starts) in RGB Color Space (Max Possible cdiff $= 1.732$) to the Graph in Fig. 1

ϵ	cdiff	CPU
10^{-1}	0.866	0.02
10^{-2}	0.974	0.05
10^{-3}	0.988	0.09
10^{-4}	0.990	0.23
10^{-5}	0.990	0.43

The CPU time is that for CLARIFY, minus the time for constructing the collision graph (0.04 seconds). The latter is independent of ϵ.

TABLE 2
Statistics on the Original and Dual Collision Graphs, CPU Time (in Second) and Objective Function (cdiff) for CLARIFY (One Random Start)

graph	$\|V\|$	$\|E\|$	$\|E_c\|$	CPU	cdiff
ngk_4	50	100	54	0.6 (0.)	122.69
NotreDame_yeast	1,458	1,948	1,685	1.3 (0.2)	67.9
GD00_c	638	1,020	1,847	1.7 (0.1)	64.32
Erdos971	429	1,312	4,427	2.1 (0.1)	59.3
Harvard500	500	2,043	11,972	2.3 (0.3)	35.0
extr1	5,670	11,405	34,696	14.5 (7.9)	47.1

The time in bracket is for constructing the dual collision graph.

Table 2 gives results on six of the graphs we tested, including running time and objective function (1) (color diff) achieved in LAB color space. These come either from the University of Florida Sparse Matrix Collection [5], or from the test graphs distributed with Graphviz [14], and originate from different application areas. 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.

It can be seen from Table 2 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 collision 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.

Fig. 7 shows the ngk_4 graph before and after the coloring. It is difficult to tell, from Fig. 7a, whether nodes 45 and 15 (blue) are connected. From Fig. 7b we can tell that they are indeed connected by a red edge.

So far we have been applying CLARIFY to straight-line drawings of graphs. The algorithm can also be used for drawings where edges are splines. This could be the result of an edge bundling, or an edge routing. Fig. 8 shows the result of applying our algorithm to a graph from a user of our software, this is one of the examples that motivates our work. As we can see, from the original drawing, it is difficult to differentiate some of the splines. For example, is node 16 connected to node 60, or to node 19 (both below node 16)? With colored splines, we can see that node 16 is connected to node 60 by a red spline.

Finally, we applied CLARIFY to color virtual maps where countries could be fragmented. Because of the fragmentation, we have to use as many color as there are countries. Fig. 9 (top) shows colored versions of an author collaboration map (see [12]) using two color palettes. Here each node is an author who published in the International Symposium of Graph Drawing between 1994 to 2004. Authors are connected by edges if they co-authored a paper. This gives a collaboration graph. Nodes are then clustered to form countries. Up to now, for coloring the edges of node-link graphs, we assume that it is equally important to differentiate all colliding edge pairs, thus set the w_{ij} in (1) to 1. For coloring virtual maps, it is more important to color adjacent countries with more distinct colors, at the same time, we also want to differentiate all countries. Thus we set w_{ij} to be the inverse of the length of the shortest path that connect countries i and j in the collision graph of the map. From Fig. 9 we can see that CLARIFY works well in using the specified palettes, keeping neighboring countries colored with very distinct colors. Unlike the coloring algorithm in [12], we also maintain good color distinction among non-neighboring countries. Additional examples are given at http://yifanhu.net/EDGE_COLORING, where readers can find graphs and maps colored using curated color palettes. These palettes add themes (e.g., pastel) to the drawings,

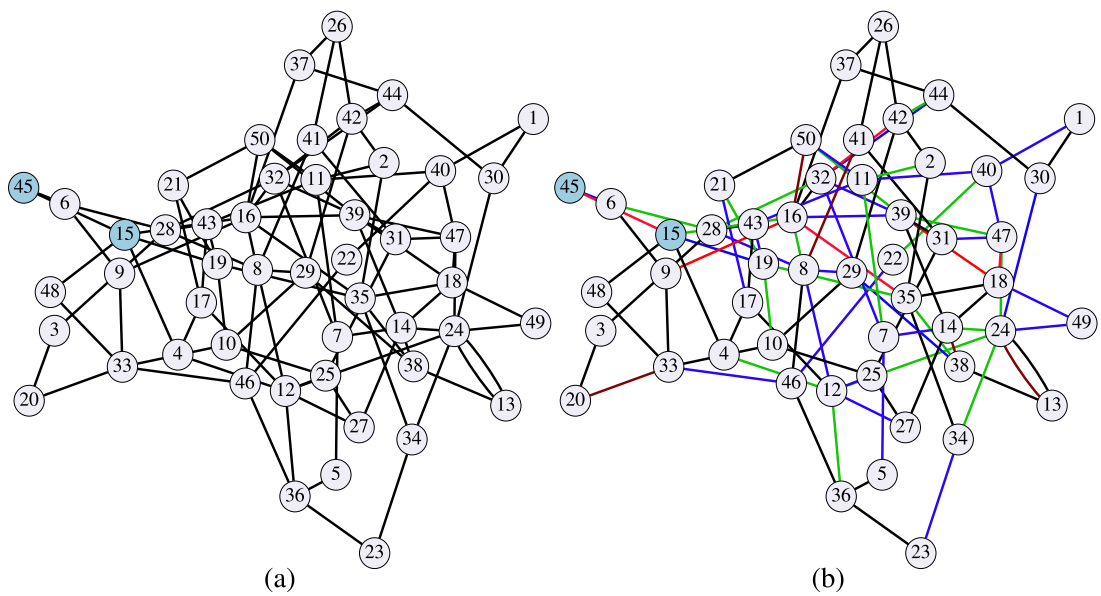


Fig. 7. Edge coloring on ngk_4 graph: (a) The original graph. Are nodes 45 and 15 (blue) connected? (b) the colored drawing. We can tell that 45 and 15 are indeed connected by a red edge.

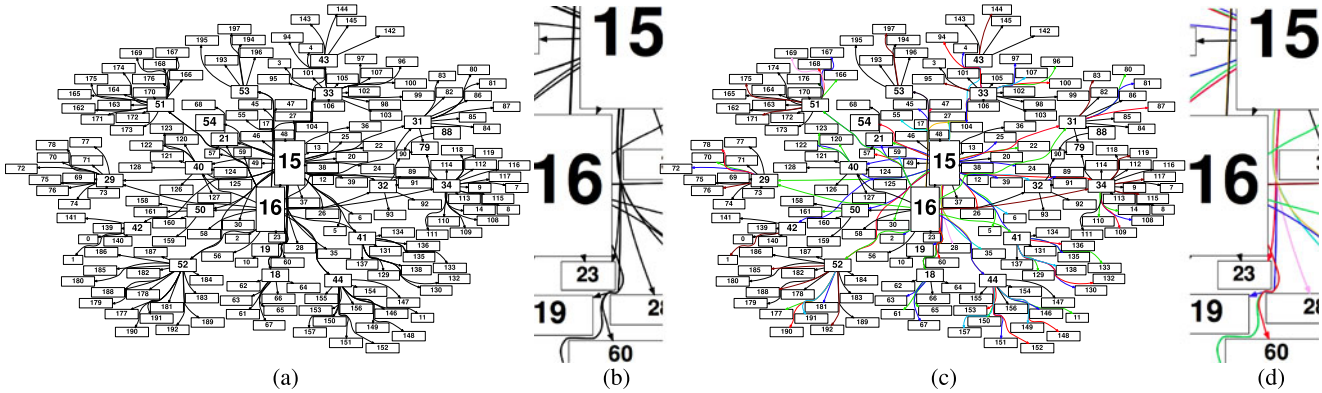


Fig. 8. (a) A graph with spline edges. Some of the splines are hard to differentiate. (b) In the zoomed-in view, is node 16 connected to node 60, or to node 19 (both below node 16)? (c) Splines are colored using the CLARIFY algorithm. Now colliding edges are easier to differentiate. (d) In the zoomed-in colored view, node 16 is seen to be connected to node 60 by a red spline, but not to 19. The latter is connected by a blue spline to node 15 above.

and makes them more aesthetically pleasing instead of looking random and jumbled, yet the edges/countries are distinct enough where it matters. We note that CLARIFY is the only method that can utilize color palettes effectively. While one could also generate colored maps by randomly select colors in a specific color palette, the end results are often unsatisfactory. For example, some neighboring countries are assigned very similar colors (Fig. 9 (bottom)).

4.5 Comparison with Jianu et al. [22]

We evaluated our algorithm against that of [22] (hereafter called JRFL), using the code kindly supplied by the authors. Fig. 6d gives the result of applying JRFL on the Zachary graph. Following [22], we use a black background, because the code sets lightness to 75. It is seen that near nodes 34 and 28, it is difficult to differentiate edges. E.g., it is not clear whether node 34 is connected to 27 or not, due to the colors of edges 34–27 and 34–23 being very similar. For a like-for-like comparison, Fig. 6e shows the result of CLARIFY with fixed lightness of 75. Despite of the restricted lightness, it does not suffer from the ambiguity seen in Fig. 6e. Finally, to understand whether JRFL is handicapped by the black background and by the fact that it considers all the edge pairs, we modified the JRFL code to use a lightness of 50, and set the weights for the pairs of edges with a closeness metric ≤ 0.1 to 0. It is seen that changing the background color and using a threshold do not resolve the issues we observed in Fig. 6d. We also compared with JRFL on other graphs, and found CLARIFY better both in terms of ability to disambiguate drawings (by visual inspection), and in speed. On most graphs, CLARIFY is about 10 times faster. An additional advantage of CLARIFY is that it is the only known algorithm that can utilize color palettes effectively.

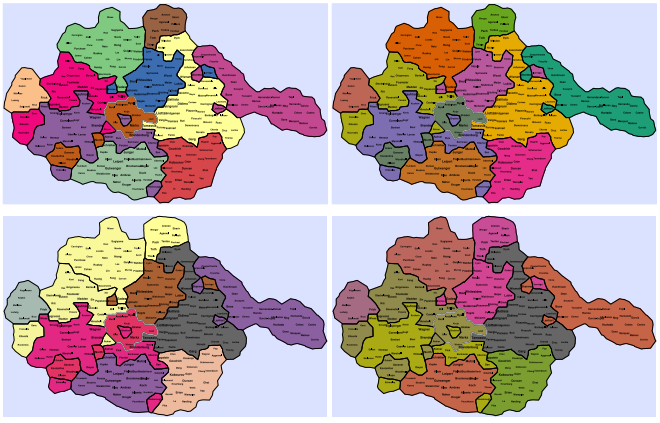


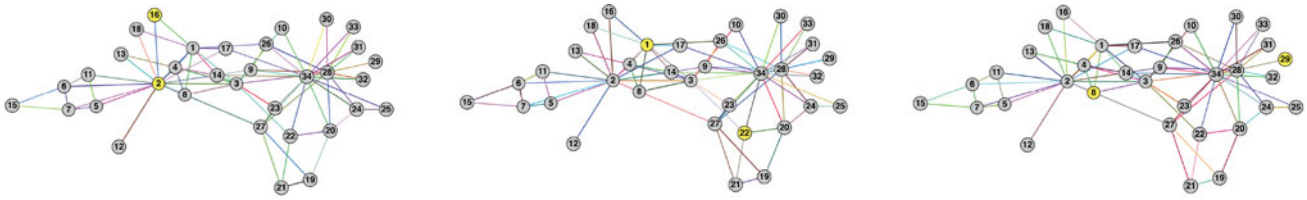
Fig. 9. Top: Applying CLARIFY on a collaboration map with two ColorBrewer palettes: Accent_8 (left) and Dark2_8 (right). Bottom: Randomly selecting colors from the palettes.

5 User Study

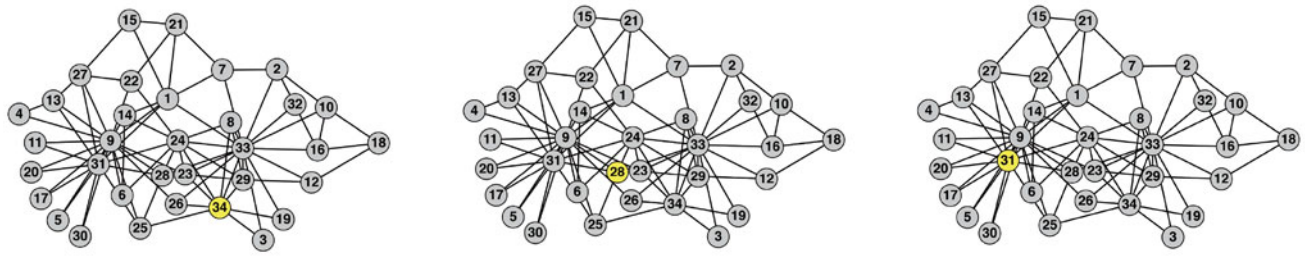
We conducted two sets of controlled user experiments to study the performance of CLARIFY edge coloring algorithm on representative graph analysis tasks, such as visually following edges, finding neighbors and identifying paths. In the first experiment, we compared three methods: the baseline method that draws each edge in Black-only (B, Fig. 10b), the edge coloring method applying our CLARIFY algorithm to determine the color of each edge (C, Fig. 10c), and the rAndom coloring (A, Fig. 10a) that uses one randomly-picked color in the entire color space for each edge. This comparison is to validate the effectiveness of using colors to draw graphs, with respect to the ordinary black-only drawing. The second experiment compared the CLARIFY algorithm with the latest JRFL edge coloring algorithm [22], to examine whether CLARIFY advances the state-of-the-art. Both experiments apply the same study design, data sets, graph analysis tasks, and study procedure.

5.1 User Study Design

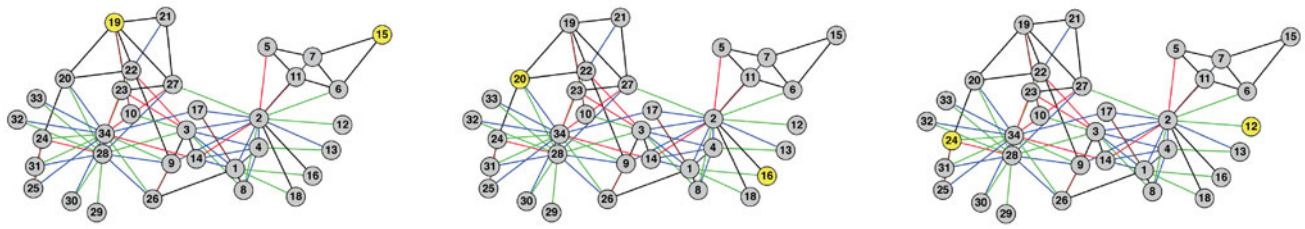
To fully utilize each subject, we applied a within-subject design that every subject entered the experiment with each of the edge coloring methods in comparison. In such design, user's performance can be distorted by the learning effect if the same graph layout is used for all the edge coloring methods. Therefore, we used three different layouts of the same graph data (Figs. 10a, 10b, and 10c). The subjects were arranged by a full factorial design to study the effect of all the combinations of edge coloring method and layout factor. The detailed design in the first experiment is shown in Fig. 11a, where subjects were grouped into 6 classes, entering different coloring-layout combinations. Within each group, we arranged 6 subjects to test every possible experiment order, so as to counter-balance the practice and fatigue effect (Fig. 11b). In total, we recruited 36 subjects for the official test of the first experiment. In the second experiment,



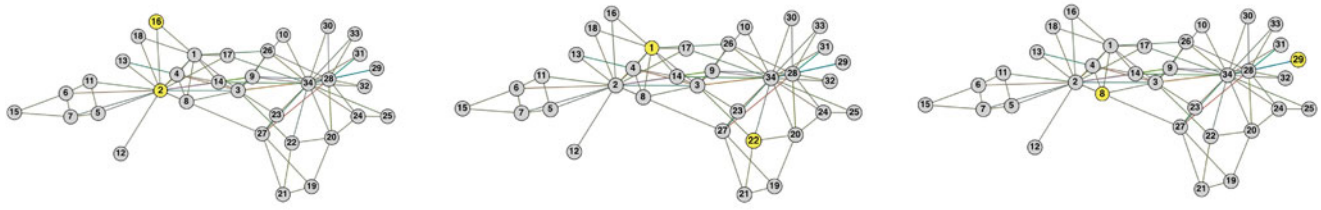
(a) Random coloring on layout 1 (A1), connectivity tasks. Left: easy case; middle: medium case; right: hard case.



(b) Black-only coloring on layout 2 (B2), degree tasks. Left: easy case; middle: medium case; right: hard case.



(c) CLARIFY coloring on layout 3 (C3), path tasks. Left: easy case; middle: medium case; right: hard case.



(d) JRFL coloring on layout 1 (J1), connectivity tasks. Left: easy case; middle: medium case; right: hard case.

Fig. 10. Four colored graph examples tested in the user study, under certain [coloring method] by [layout] by [analysis task] combinations. Not all combinations are shown due to the space limit.

we recruited another set of 24 subjects as there are only four different coloring-layout-order combinations, i.e., two coloring methods (CLARIFY and JRFL) and two layouts (layout 1, 2). All subjects were graduate students majoring in computer science, which accorded well with the potential user base of graph drawing tools. Note that we do not consider the between-subjects difference.

5.2 Data and Tasks

All tests were done using the Zachary's Karate Club social graph. This is a well known benchmark graph with 34 nodes and 78 edges. Three layouts were generated with different node numberings to eliminate the learning effect (Figs. 10a, 10b, and 10c). On each combination of the edge coloring and layout, three graph analysis tasks (T1∼T3) were designed for users to complete (Fig. 11c).

T1 (1-hop Connectivity): Determine whether two nodes are connected by an edge directly;

T2 (Degree): Estimate the number of nodes that a particular node connects to directly;

T3 (Path): Write down the shortest path between two nodes (including these two).

Examples of the colored graphs used for the three tasks are given in Figs. 10a, 10b, 10c, and 10d respectively. Each task was repeated three times for each subject, representing

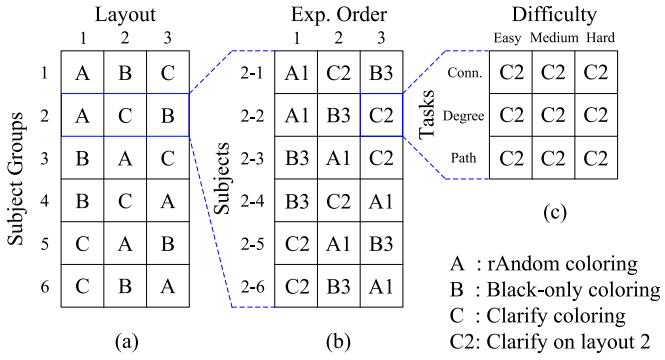


Fig. 11. The full factorial design to counter-balance negative effects such as learning, practice and fatigue (the case of the first experiment).

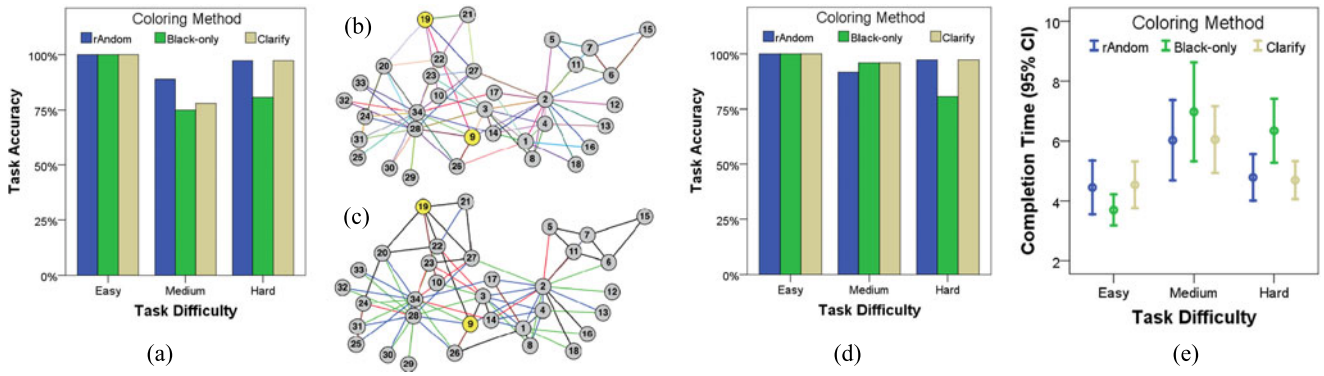


Fig. 12. The accuracy and completion time of connectivity tasks, reported by their difficulty levels (first experiment).

easy, medium and hard tasks by the different edges/nodes/paths selected.

After completing all the tasks for each coloring method, subjects also responded to two subjective questions (Q1∼Q2). Answers were selected from a 1∼7 Likert scale.

Q1 (Usability): How much does this coloring method help you in completing the tasks and finding the correct answers?

Q2 (User Experience): How much do you like the experience with this coloring method?

5.3 Study Procedure

The study procedure is composed of two sessions: a training session and a test session. The training session was introduced to make sure each subject understood all tasks well and became familiar with the graph drawing and coloring methods. It included two sample tasks from each task type on a much simpler graph. The organizer checked the answer of each training task and explained any ambiguity on the task immediately. The test session is the formal user study, during which we recorded the subject's answer and the completion time in each task. The task completion time was measured after the subject had read the question, so that the reading skill variation was excluded.

Note that the study procedure and task designs had been carefully refined based on the result and feedback from the pilot study before this test. First, we found that the task accuracy and completion time had a coupling effect with the task difficulty, which may also influence the comparison result on coloring methods. Therefore, we classified each task into three difficulty levels according to the edge/node/path chosen in the question (Fig. 10, from left to right). User's results were compared within the same level to eliminate the factor of task difficulty. Second, it was noticed during the experiment, most of the subjects spent a good amount of time locating the nodes mentioned on each question, which can be a disturbing factor to the result. Our solution was to annotate the related nodes in yellow to remind the subjects of their focuses (Fig. 10). Third, our initial design on the path task was to ask subjects to count the length of the shortest path. However, in the pilot study, subjects can either get wrong due to a misunderstanding of the concept of length for the shortest path (false negative) or get right by counting the length of a non-path (false positive). We solved this by asking subjects to write down the shortest path, for which the correctness of being a path and the shortest path were checked after the experiment.

All experiment results were analyzed separately on each task. Significant level was set at 0.05 throughout the analysis.

5.4 Result on Comparing Black-Only and Colored Drawings

In the first experiment, three coloring methods are compared: the black-only, the CLARIFY, and the random coloring.

Connectivity Task. The user's accuracy in judging the 1-hop connectivity is summarized in Fig. 12a. On easy tasks, it is as expected that all methods receive a 100 percent accuracy because there is no conflicting edge or overlapping to disturb the connectivity, such as the case in the left part of Fig. 10a. On hard cases, both the random coloring and CLARIFY reach a high accuracy of 97.3 percent, while the black-only coloring only receives an accuracy of 80.6 percent. The result on the medium difficulty level has been a surprise: the CLARIFY algorithm (77.9 percent) only gains a tiny advantage over the black-only coloring (74.9 percent), and both are worse than the random coloring (89.2 percent). In a finer grained analysis, we find the root cause for this exception. There is one tricky case that the CLARIFY algorithm uses the color of black for the target edge, as shown in Fig. 12c, the same with the black-only coloring on this edge, while the random coloring chooses a much brighter nontrivial color for this edge (Fig. 12b). The user's accuracy in this case is 83.3 percent for the random coloring, 41.7 percent for the CLARIFY algorithm and 33.3 percent for the black-only coloring, with a big deviation from the average case. After we remove this tricky case (about 11 percent of all cases), the resulting task accuracy plot becomes Fig. 12d, in which all three methods receive similar accuracy under easy and medium difficulty levels.

We conducted the binary logistic regression to capture the boolean value of the task accuracy. It is shown that on hard tasks, the contribution of the coloring method to the task accuracy variation is statistically significant ($p = .027$). Compared to the random coloring and the CLARIFY algorithm, the black-only coloring decreases the likelihood (odds) of correctly answering each task to 9.3 percents of the random/CLARIFY methods (95 percent CI = [1, 87.7], $p = .038$), controlling for the difference on layouts. The goodness of fit of this logistic regression model is 0.339 (Nagelkerke R Square). On the completion time, Fig. 12e shows the average time in 95 percent Confidence Interval (CI) error bars. The black-only method costs the user more time on medium and hard tasks, but requires less time on

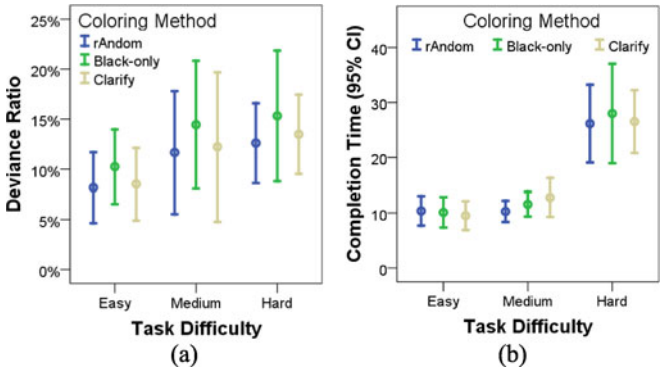


Fig. 13. The deviance ratio and completion time of degree tasks, reported by their difficulty levels (first experiment).

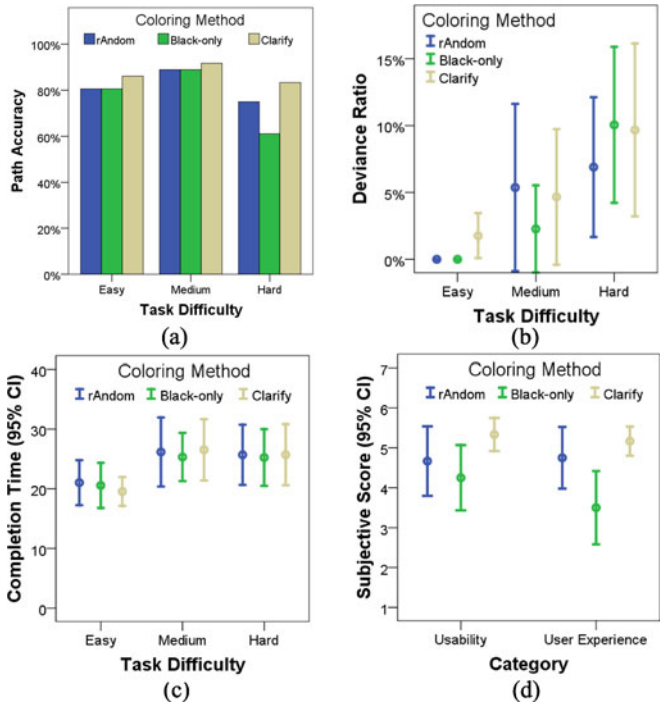


Fig. 14. (a)∼(c) The accuracy, deviance ratio and completion time of path tasks, reported by their difficulty levels; (d) Subjective ratings on the usability and user experience of three coloring methods (first experiment).

easy tasks. From the in-depth pilot study, this result can be explained by the user feedback that the black-only drawing spends less of the user's cognitive effort in completing very easy tasks. The follow-up analysis of variance (ANOVA) test reveals a significant difference in the completion time of coloring methods on hard tasks, by an unequal variance F-test, $F(2, 67) = 3.841, p = .026$. Because of the non-compliance to the homogeneity of variances ($p < .05$ in the Levene test), we have applied the Welch ANOVA here. By the Games-Howell post hoc test, there are significant differences between the black-only method and the CLARIFY algorithm ($p = .025$), and between the black-only method and the random coloring ($p = .05$).

Degree Task. The estimated degree by users is translated into the measure of deviance ratio, which is the absolute degree deviance divided by the correct degree. Fig. 13a summarizes the deviance ratio of degree tasks. It is clear that on all difficulty levels, the black-only method suffers from a higher error in degree tasks, though the difference is not significant in the ANOVA test. On the completion time, as shown in Fig. 13b, the differences on all difficulty levels are quite small.

Path Task. The raw input on the path task is checked in two phases: 1) whether the answer is a true path in the graph; 2) whether the true path is the shortest one. Again, we use the deviance ratio as the measure of error for a path, which is the absolute path length deviance divided by the length of the shortest path. Fig. 14a reports the mean accuracy in identifying a true path. On all difficulty levels, the CLARIFY algorithm achieves a better accuracy than the black-only method and the random coloring. However, the significant difference is only observed on hard tasks: compared to the CLARIFY algorithm, the black-only method decreases the likelihood (odds) of correctly identifying a path to 4.5 percents of the CLARIFY algorithm (95 percent CI = [0.4, 45.5], $p = .009$), controlling for the difference on layouts. The goodness of fit of the logistic regression model is 0.69 (Nagelkerke R Square).

On the identified true paths, we summarize their deviance ratios to the shortest path in Fig. 14b. Both graphical and statistical analyses do not find coherent difference among coloring methods. On the completion time, as shown in Fig. 14c, the performance of three coloring methods is also close to each other.

Subjective Questions. The user's subjective scores on the usability and user experience are summarized in Fig. 14d. It is shown that the CLARIFY algorithm is better rated than the random coloring and the black-only coloring. We then apply the Kruskal-Wallis test to analyze their differences, which does not require a normality assumption of the observed data. Results indicate that there is a large difference among coloring methods on the usability ($\chi^2(2) = 16.2, p = .067$), though not significant. The mean rank is 70.4 for CLARIFY, 53.6 for the random coloring, and 42.5 for the black-only coloring (the rank value has a range of 1 to 108 from 108 feedbacks on three coloring methods). On the user experience, there is a significant difference among coloring methods ($\chi^2(2) = 35.4, p = .003$). The mean rank is 71.6 for CLARIFY, 62.9 for the random coloring, and 32 for the black-only coloring. Follow-up Mann-Whitney tests are conducted to evaluate the pairwise difference among coloring methods. It is shown that, between CLARIFY and the black-only coloring, the subjective ratings are significantly different on both the usability ($U = 310.5, p = .019$) and the user experience ($U = 157.5, p = .001$). Between the random coloring and the black-only coloring, the subjective ratings are significantly different only on the user experience ($U = 292.5, p = .018$). Between CLARIFY and the random coloring, the subjective ratings are not significantly different, though the CLARIFY algorithm receives better average ratings.

5.5 Result on Comparing Coloring Methods

In the second experiment, two coloring methods are compared: the CLARIFY coloring and the JRFL coloring.

Connectivity Task. The user's accuracy in judging the 1-hop connectivity is summarized in Fig. 15a. On the easy task, same with the first experiment, both JRFL and CLARIFY coloring methods receive a 100 percent accuracy. On medium and hard tasks, CLARIFY (100, 96 percent) is better than JRFL (96, 87 percent), though the difference is not significant. Note

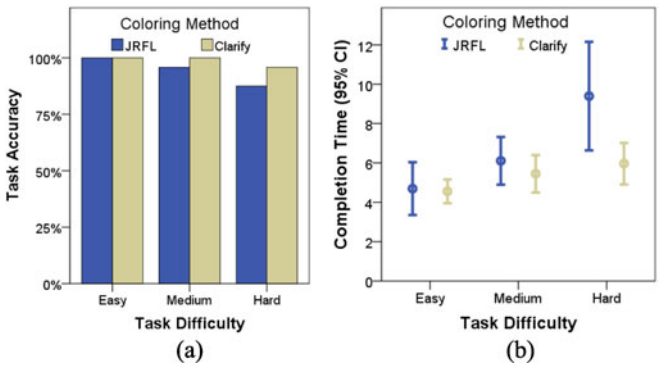


Fig. 15. The accuracy and completion time of connectivity tasks, reported by their difficulty levels (second experiment).

that the tricky case in the first experiment is avoided by not using layout 3. We conducted the binary logistic regression to capture the boolean value of the task accuracy. On all three difficulty levels, the contribution of the coloring method to the task accuracy variation is not significant. The largest effect happens in the hard task: compared to the JRFL coloring, CLARIFY increases the likelihood (odds) of correctly answering the task to 3.66 times (95 percent CI = [0.32, 41.7]), controlling for the difference on layouts. The goodness of fit of this logistic regression model is 0.317 (Nagelkerke R Square).

On the completion time, Fig. 15b shows the average time in 95 percent Confidence Interval error bars. JRFL costs the user more time on all difficulty levels. The follow-up analysis of variance (ANOVA) test reveals a significant difference on the hard task, by an unequal variance F-test, $F(1, 30) = 5.789, p = .023$. Because of the non-compliance to the homogeneity of variances ($p < .05$ in the Levene test), we applied the Welch ANOVA here.

Degree Task. Fig. 16a summarizes the deviance ratio of degree tasks. It can be seen that on all difficulty levels, JRFL suffers from a higher error in degree tasks. The differences are not significant in the ANOVA test for all levels, but on easy task, the difference is very close to significance ($p = .071$). On completion time (Fig. 16b), the effect is the same with the connectivity task that JRFL costs users more time on all difficulty levels. On the hard task, the difference is significant by the Welch ANOVA, $F(1, 30) = 4.393, p = .045$.

Path Task. Fig. 17a reports the mean accuracy in identifying a true path. On all difficulty levels, CLARIFY and JRFL have a similar path accuracy, with differences smaller than 10 percent. By binary logistic regression, the contribution

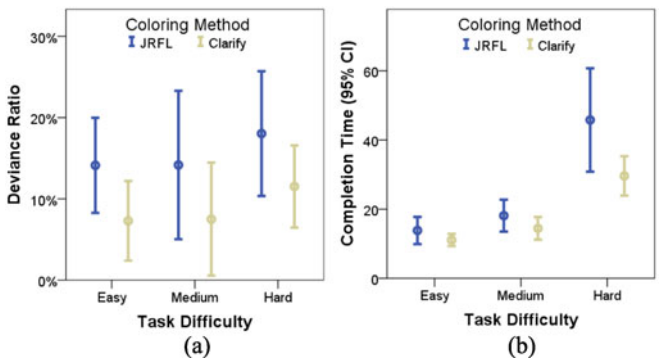


Fig. 16. The deviance ratio and completion time of degree tasks, reported by their difficulty levels (second experiment).

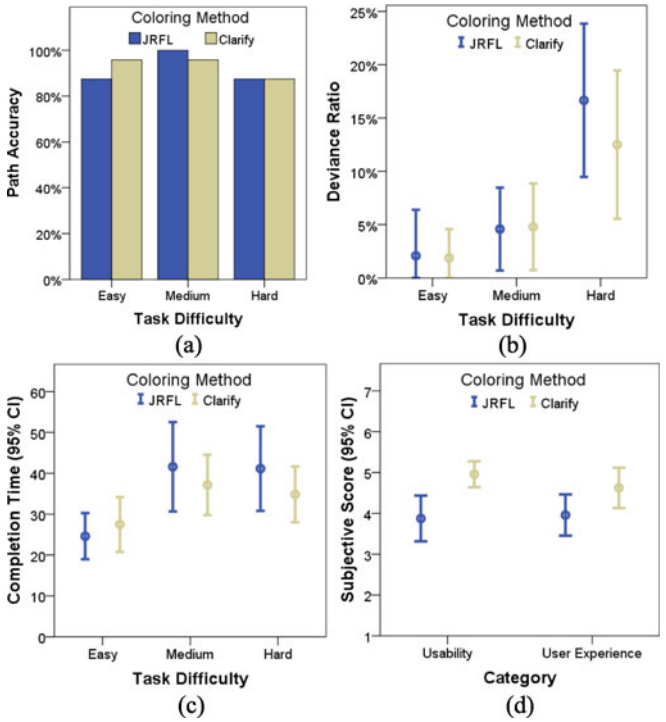


Fig. 17. (a)∼(c) The accuracy, deviance ratio and completion time of path tasks, reported by their difficulty levels; (d) Subjective ratings on the usability and user experience of two coloring methods (second experiment).

of coloring method to the path accuracy is not significant on all levels.

On the identified true paths, we summarize their deviance ratios to the shortest path in Fig. 17b. On all difficulty levels, the ANOVA test on the path deviance ratio does not reveal significant difference between CLARIFY and JRFL.

On the completion time, as shown in Fig. 17c, JRFL lets users spend more time than CLARIFY on medium and hard tasks, and slightly less time on the easy task, though all differences are not significant by the ANOVA test.

Subjective Questions. The user's subjective scores on the usability and user experience are summarized in Fig. 17d. It is shown that CLARIFY is better rated than JRFL on both scores. We then apply the Mann-Whitney test to analyze their differences, which does not require a normality assumption of the observed data. Results indicate that there is a significant difference between CLARIFY and JRFL on the usability score ($U = 143.0, p = .002$). The mean rank is 30.5 for CLARIFY and 18.5 for JRFL (the rank value has a range of 1 to 48). On the user experience, the difference is also significant ($U = 188.5, p = .031$). The mean rank is 28.7 for CLARIFY and 20.3 for JRFL.

5.6 Summary and Implication

First, our user study results demonstrate that the edge coloring technique (the random and CLARIFY algorithms) can improve user's performance in all the three representative graph analysis tasks studied here, and they receive significantly better subjective ratings from the user. Exceptions only happen on a few easy/medium tasks without conflicting edges (e.g., the connectivity task in the left part of Fig. 10a). The superiority of edge coloring is especially notable on hard tasks, where we obtain significance on the task

accuracy of connectivity and path tasks. These results greatly encourage the use of edge coloring techniques.

Second, among different coloring algorithms, CLARIFY enjoys a clear advantage over the previous JRFL algorithm, on all three graph analysis tasks. Significant differences are observed on the completion time of connectivity and degree tasks, as well as the user's subjective ratings. The advantage of CLARIFY can be attributed to the larger color discrimination among conflicting edges, in both intensity and color hue. More details can be found in Section 4.5.

Third, between the random coloring and our CLARIFY algorithm, we should claim that in a lot of cases the difference is small. However, on path-related hard tasks, as well as the user's subjective ratings, CLARIFY performs much better. The algorithm minimizes the number of colors to distinguish conflicting edges, so that user's cognitive efforts on hard tasks are reduced. This is more important on global tasks (e.g., the path task) than local tasks (e.g., the connectivity and degree task); and on hard tasks than easy tasks. Furthermore, CLARIFY is the only algorithm that can effectively take advantage of a color palette for specific visual theme. While random coloring could also use a color palette, as shown in Fig. 9, this often gives unsatisfactory results.

Last but not the least, the user study result suggests a few improvements to the CLARIFY algorithm design. Most notably, the brighter non-black color should be used for all edges with connectivity conflict, so that users can pay more attention to disambiguating these edges.

6 Discussions

The approach of coloring edges for disambiguating drawings has its limitations. Our working assumption is that the drawing is displayed as a static image on paper, or on screen. In cases where an interactive environment is available, interactive techniques such as "link sliding" and "bring & go" [23] could be more effective. In such a situation, the algorithms proposed here can be used as an additional visual aid to the interactive techniques.

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.

During our user study, we found that using black as the default color for non-conflicting edges may not be the most appropriate option in some cases. Some users fail to understand that a black edge passing under a node label is one edge, not two edges. For example, the edge between nodes 19 and 9 in Fig. 12c may be better visualized using a brighter color as the default for such edges. Our user study also shows that for the purpose of distinguishing edges, random coloring also works. However as shown in Section 4.4, random coloring does not work when applied to color palettes. The color space for a color palette is discrete and much smaller, thus it is likely that random coloring will fail to find the optimal color combination.

There are situations in which it may not be appropriate to use colors to differentiate edges. First, there is a perceptual cost of introducing color to the visualization of graphs. As our user study shows, for easier tasks, colored versions take users more time to complete. Second, colors may be reserved to encode other information. The proposed method can work with any style spaces. For example, for disambiguating the edges in Fig. 4, we avoided using colors for edges in order to accurately display colors of the nodes. For that drawing we used CLARIFY with grayscale, so that edges are in black or gray. In general, with CLARIFY, edges can be differentiated using dashed lines or textures of different style. This can be achieved by mapping different line styles to a region or a set of discrete points in the 2D/3D space.

7 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. The algorithm works for both continuous color space and discrete color palettes, and can also be applied to generate coloring for disambiguating virtual maps. Our user study found that coloring edges in graph drawings helped users' performance in major graph analysis tasks, and sometimes the improvement is significant. Consequently, we have made the CLARIFY code available as the `edge-paint` function in the open source Graphviz software.

Acknowledgments

Lei Shi was supported by China National 973 Project 2014CB340301, NSFC Grants 61379088, 61772504, and the Key Research Program of Frontier Sciences, CAS (Grant No. QYZDY-SSW-JSC041).

References

[1] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Algorithms for the Visualization of Graphs*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1999.

[2] C. Brewer, ColorBrewer. (2017). [Online]. Available: http://www.colorbrewer2.org

[3] R. Chernobelskiy, K. I. Cunningham, M. T. Goodrich, S. Kobourov, and L. Trott, "Force-directed Lombardi-style graph drawing," in *Proc. 19th Int. Symp. Graph Drawing*, 2011, pp. 320–331.

[4] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li, "Geometry-based edge clustering for graph visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 14, no. 6, pp. 1277–1284, Nov./Dec. 2008.

[5] T. A. Davis and Y. Hu, "University of Florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, pp. 1–18, 2011. [Online]. Available: http://www.cise.ufl.edu/research/sparse/matrices/

[6] G. Di Battista and L. Vismara, "Angles of planar triangular graphs," in *Proc. 25th Annu. ACM Symp. Theory Comput.*, 1993, pp. 431–437.

[7] W. Didimo, P. Eades, and G. Liotta, "Drawing graphs with right angle crossings," *Theoretical Comput. Sci.*, vol. 412, no. 39, pp. 5156–5166, 2011.

[8] M. B. Dillencourt, D. Eppstein, and M. T. Goodrich, "Choosing colors for geometric graphs via color space embeddings," in *Proc. 14th Int. Symp. Graph Drawing*, 2006, pp. 294–305.

[9] C. Duncan, D. Eppstein, M. T. Goodrich, S. Kobourov, and M. Nöllenburg, "Lombardi drawings of graphs," *J. Graph Algorithms Appl.*, vol. 16, pp. 85–108, 2012.

[10] B. Fraser, C. Murphy, and F. Bunting, *Real World Color Management*, 2nd ed. Berkeley, CA, USA: Peachpit Press, 2004.

[11] E. Gansner, Y. Hu, S. North, and C. Scheidegger, "Multilevel agglomerative edge bundling for visualizing large graphs," in *Proc. IEEE Pacific Vis. Symp.*, 2011, pp. 187–194.
[12] E. R. Gansner, Y. Hu, and S. Kobourov, "Visualizing graphs and clusters as maps," *IEEE Comput. Graph. Appl.*, vol. 30, no. 6, pp. 54–66, Nov./Dec. 2010.
[13] E. R. Gansner and Y. Koren, "Improved circular layouts," in *Proc. 14th Int. Symp. Graph Drawing*, 2006, pp. 386–398.
[14] E. R. Gansner and S. North, "An open graph visualization system and its applications to software engineering," *Softw. Practice Experience*, vol. 30, pp. 1203–1233, 2000.
[15] A. Garg and R. Tamassia, "Planar drawings and angular resolution: Algorithms and bounds," in *Proc. 2nd Eur. Symp. Algorithms*, 1994, pp. 12–23.
[16] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, pp. 741–748, Sep. 2006.
[17] D. Holten and J. J. van Wijk, "Force-directed edge bundling for graph visualization," *Comput. Graph. Forum*, vol. 28, pp. 983–990, 2009.
[18] Y. Hu, S. Kobourov, and S. Veeramoni, "On maximum differential graph coloring," in *Proc. 18th Int. Symp. Graph Drawing*, 2010, pp. 274–286.
[19] Y. Hu and L. Shi, "A coloring algorithm for disambiguating graph and map drawings," in *Proc. 21th Int. Symp. Graph Drawing*, 2014, pp. 89–100.
[20] W. Huang, "Using eye tracking to investigate graph layout effects," in *Proc. 6th Int. Asia-Pacific Symp. Vis.*, 2007, pp. 97–100.
[21] W. Huang, S.-H. Hong, and P. Eades, "Effects of crossing angles," in *Proc. IEEE Pacific Vis. Symp.*, 2008, pp. 41–46.
[22] R. Jianu, A. Rusu, A. J. Fabian, and D. H. Laidlaw, "A coloring solution to the edge crossing problem," in *Proc. 13th Int. Conf. Inf. Vis.*, 2009, pp. 691–696.
[23] T. Moscovich, F. Chevalier, N. Henry, E. Pietriga, and J. Fekete, "Topology-aware navigation in large networks," in *Proc. 27th Int. Conf. Human Factors Comput. Syst.*, 2009, pp. 2319–2328.
[24] J. O'Rourke, *Computational Geometry in C*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 1998.
[25] J. Peltonen and Z. Lin, "Peacock bundles: Bundle coloring for graphs with globality-locality trade-off," in *Proc. 24th Int. Symp. Graph Drawing Netw. Vis.*, 2016, pp. 52–64.
[26] S. Pupyrev, L. Nachmanson, S. Bereg, and A. Holroyd, "Edge routing with ordered bundles," in *Proc. 19th Int. Symp. Graph Drawing*, 2011, pp. 136–147.
[27] H. C. Purchase, "Which aesthetic has the greatest effect on human understanding?" in *Proc. 5th Int. Symp. Graph Drawing*, 1997, pp. 248–261.
[28] H. C. Purchase, D. A. Carrington, and J.-A. Allder, "Experimenting with aesthetics-based graph layout," in *Proc. 1st Int. Conf. Theory Appl. Diagrams*, 2000, pp. 498–501.
[29] H. C. Purchase, J. Hamer, M. Nöllenburg, and S. G. Kobourov, "On the usability of Lombardi graph drawings," in *Proc. 20th Int. Symp. Graph Drawing*, 2012, pp. 451–462.
[30] C. Ware, H. Purchase, L. Colpoys, and M. McGill, "Cognitive measurements of graph aesthetics," *Inf. Vis.*, vol. 1, no. 2, pp. 103–110, Jun. 2002.
[31] A. Wolff, "Drawing subway maps: A survey," *Informatik-Forschung und Entwicklung*, vol. 22, no. 1, pp. 23–44, 2007.

Yifan Hu received the BS and MS degrees in applied mathematics from Shanghai Jiao-Tong University and the PhD degree in optimization from Loughborough University, United Kingdom. He is currently a senior principal research scientist at Yahoo Research, having previously worked at AT&T Labs. He is a contributor to the Graphviz graph drawing system. His research interests include data mining and information visualization.

Lei Shi received the BS, MS, and PhD degrees from Tsinghua University, in 2003, 2006, and 2008, respectively. He is a professor in SKLCS, Institute of Software, Chinese Academy of Sciences. His research interests include visual analytics and data mining. He has published more than 70 papers in refereed conferences and journals. He is the recipient of the IBM Research Accomplishment Award on "Visual Analytics" and the VAST Challenge Award twice in 2010 and 2012.

Qingsong Liu received the BS degree from the International School of Software, Wuhan University, in 2009. He is working toward the graduate degree in SKLCS, Institute of Software, Chinese Academy of Sciences. His research interests include information visualization and visual analytics.