Supporting Interactive Graph Exploration Using Edge Plucking

Nelson Wong and Sheelagh Carpendale

Department of Computer Science, University of Calgary, Calgary, Canada

ABSTRACT

Excessive edge density in graphs can cause serious readability issues, which in turn can make the graphs difficult to understand or even misleading. Recently, we introduced the idea of providing tools that offer interactive edge bending as a method by which edge congestion can be disambiguated. We extend this direction, presenting a new tool, Edge Plucking, which offers new interactive methods to clarify node-edge relationships. Edge Plucking expands the number of situations in which interactive graph exploration tools can be used to address edge congestion.

Keywords: Graph visualization, interaction, edge congestion, Information Visualization, navigation

1. INTRODUCTION

An increasing number of tasks require people to explore, navigate, and search extremely complex data sets. If these data sets can be described as entities and relationships between these entities then they can be visualized as graphs. There are many real world examples of this type of data such as electrical and telecommunication networks, web structures, airline routes, software entity relationship diagrams, and road maps. One problem is that graphs of these real world data sets have many relationships between nodes, ultimately leading to visualizations that are hard to understand because of the amount of visual congestion caused by the edges. Most previous research has approached this problem as a node layout problem endeavouring to improve readability by changing node positions to create better layouts. A new direction[1,2] introduced the concept of interactive solutions and the idea of addressing edge congestion by temporarily curving edges to alleviate congestion in a given location. This promising new direction thus far has only one method, EdgeLens,[1] which can be used to interactively address several but not all edge congestion situations. We present Edge Plucking, a new approach that extends these ideas, offering interactive solutions to an increasing number of edge congestion situations.

Edge Plucking is similar to EdgeLens in its basic concept in that it temporarily curves edges to clarify edge congestion and graph readability issues; however, the basic interaction technique of Edge Plucking can be thought of as the opposite to that of the EdgeLens. EdgeLens pushes edges away from a lens centre. In contrast, the interaction concept behind Edge Plucking is to use a natural plucking action to mitigate edge congestion. That is, one selects an edge(s) and pulls them out of the way. This plucking action is used in everyday life. For example, as in Figure 1, when one wants to peek through a set of closed Venetian blinds, one may run a finger down the blinds and then peek through. Afterwards, when one releases the blinds, they will return to their original shape. Edge Plucking simulates this plucking action for use in graph exploration. Also, unlike the EdgeLens, Edge Plucking does not have a lens centre; instead, it operates directly with the cursor. To pluck edges, one selects edges with the cursor and then, once edges have been selected, the moving cursor will pull the selected edges in the direction of the moving cursor.

Next, we discuss edge congestion readability issues and then from this perspective, we describe the related research. In Section 4, we explain the concept of Edge Plucking and outline our design goals. Its development is described in Section 5. Section 6 shows how Edge Plucking can address graph readability issues. In Section 7 we present a case study of using Edge Plucking and EdgeLens separately in a social network. In Section 8 we compare and contrast Edge Plucking with EdgeLens and conclude in Section 9.

Further author information: (Send correspondence to N.W.)
N.W.: E-mail: yw@cs.ucalgary.ca
S.C.: E-mail: sheelagh@cs.ucalgary.ca

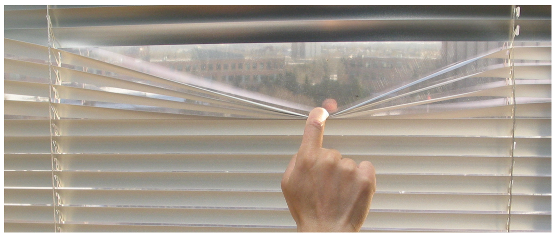


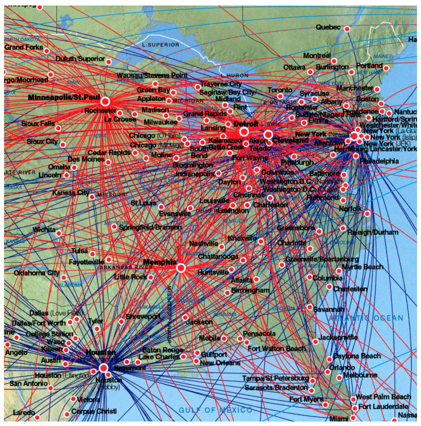
Figure 1. Running a finger down a set of closed Venetian blinds

2. EDGE CONGESTION READABILITY ISSUES

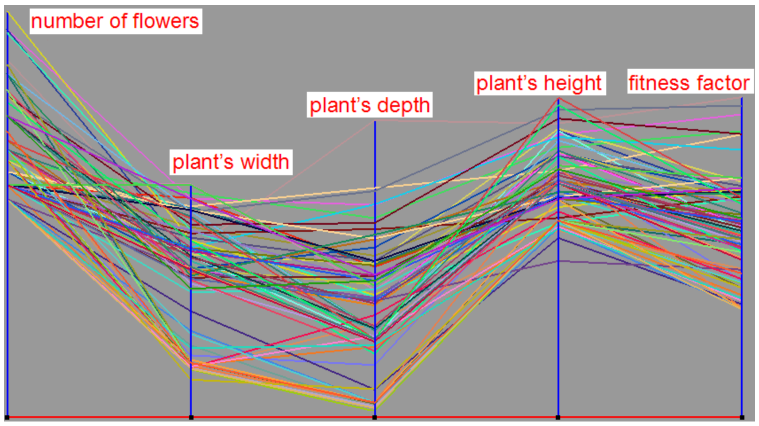
In taking another look at the edge congestion problem we start by creating an integrated list of edge congestion issues. In this we draw from the concepts raised in graph drawing aesthetics such as minimizing edge crossings, minimizing edge bends, maximizing minimum angles between edges, and aiming for continuity, symmetry, and orthogonality[3] as well as the ambiguities noted when edges overlap each other and other nodes.[1] Creating a more complete list of edge congestion issues invites design considerations to develop tools and methods can alleviate any of these points.

Occlusion caused by congested edges: When a large number of edges overcrowd a region of a graph, these edges may obscure nodes and labels as well as information in the background. To illustrate this point we show a portion of a real world graph of the airline routes flown by NorthWest Airlines (November 2001 in-flight magazine) (Figure 2(a)). As seen in this image, the background can be covered by edges and the labels can become difficult to read.

Ambiguous edges: Edges indicate relationships between nodes. An ambiguous edge, or set of edges, is one that fails to sufficiently clarify these relationships. While ambiguous edges are more common with edge congestion, they can occur in simple graphs. For instance, Figure 3 shows three different possible node and edge connections in a three-node graph. Graph (a) is an ambiguous graph, which can be interpreted as different graphs, e. g. Graph (b) and (c). Graph (b) only has one edge connecting both end nodes and the middle node is not attached to any edges. In Graph (c), the end nodes are connected and one of them is connecting to the centre node. Even a simple three-node graph can be ambiguous, creating difficulties when identifying relationships between nodes. Figure 2(b) shows a common multi-dimensional data visualization technique, Parallel Coordinates[4] which has many examples of ambiguous edges.



(a) Airline routes from NorthWest Airlines, "World Traveler", November, 2001



(b) Parallel Coordinates of plant generations

Figure 2. Examples of edge congestions

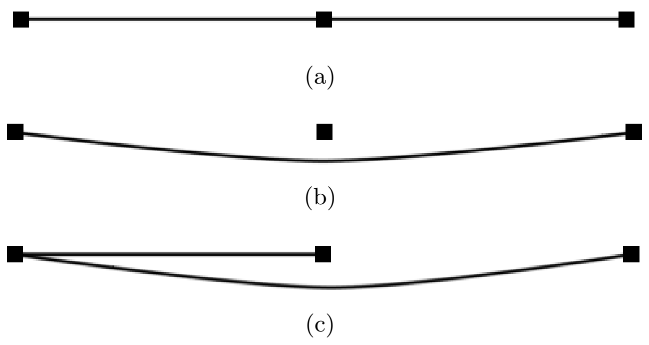

(a)
(b)
(c)

Figure 3. Different node-edge relationships in a three-node graph

Low angular resolution: When the angular resolution—the angle between adjacent edges incident on a given node—is low it can be difficult to distinguish between nearby edges. The graph in Figure 4 is an example that shows edges with low angular resolution that are confusing. Node A has an edge connected to Node B and another edge connected to Node C, but it is difficult to see these connections in the graph.

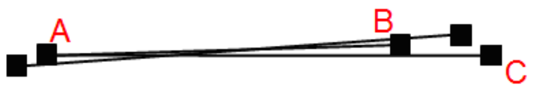


Figure 4. Low angular resolution

Long edges are difficult to follow: Figure 5 shows an example where the confusion is aggravated by long edges. Here it is difficult to see which nodes are actually connected to Node A. When following a long edge that is close to several other long edges, it is easy to lose track of the edge of interest.

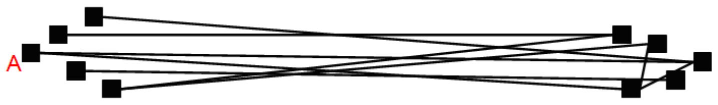


Figure 5. Long edges

Edge crossings and bends: Both edge crossings and bends can affect the readability of graphs[5] because they may be confused with nodes. Also inflection points make edges hard to follow. When a separate edge crosses a given edge at an inflection points it can be difficult, if not impossible, to assess which sections of the edges belong together. For instance, the graph in Figure 6 is difficult to read because a crossover occurs at the inflection point. It is difficult to tell whether Node A is connected to Node B or D, and whether Node C is connected to Node B or D.

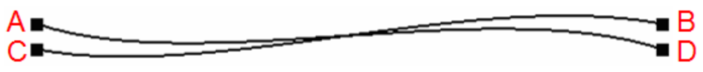


Figure 6. Inflection point

While each of these issues can give rise to ambiguity in themselves, they tend to exacerbate each other. The higher the edge density the more likely it is that several of these issues will be present.

3. RELATED WORK

Initially, the edge congestion problem was most frequently approached from the perspective of creating better graph layouts by rearranging the nodes. However, it is difficult to obtain optimal algorithms.[6–8] Many graphs are inherently non-planar, making it impossible to create layouts without edge crossings. Also, when using graphs to represent information that is spatially explicit, such as road maps where nodes represent cities, layout algorithms are not applicable because node adjustments would result in losing important geographical information.

Graph layout can also be created[9] or adjusted interactively.[10] Allowing people to change the arrangements or appearances of nodes and edges interactively is another way to address the edge congestion problem. Examples of such techniques include various types of magnification and filtering.

For magnification, different methods for enlarging objects of interest to reveal more detail have been explored. In a full zoom the whole representation is enlarged. As a smaller area of the representation fills the available display space, contextual information is lost. To maintain context, zooming has been extended to insets, overview & detail presentations, and fisheyes. Insets and overview & detail presentations show two views; one at original scale and one enlarged. The organization of the two views may vary and connections may be drawn between them,[11, 12] however context is juxtaposed rather then integrated. With fisheye views the area of interest is enlarged in place and context adjusted or compressed in order to provide space.[13] However, magnification does not really help the edge congestion problem. If a confusing and cluttered layout exists magnification may make it larger but it likely does not clarify the situation.

With filtering, edge congestion can be reduced by removing less important edges, thus revealing only the important relationships in the graph.[14] To be effective this technique requires a good method to distinguish "important" from "unimportant" edges. Filtering also interferes with context. When "unimportant" edges are filtered, the relationships between the "important" and "unimportant" edges are lost. Partial filtering has also been used to remove the central portions of edges interactively. This is the technique used in SeeNet,[15] SeeNet3D,[16] and the visualization of MBone.[17] This filtering method leaves an indication that showing the directions of edges, but precise relationships are harder to determine because the originally connected nodes are now visually disconnected.

The most closely related previous approaches to our work are two interactive techniques that provide methods for temporarily moving edges out of the way. These are BubbleLens[2] and EdgeLens.[1] They both work on the metaphor of a force field that pushes out from the lens centre, curving edges to create a clear region. This has been shown to be quite effect for local dis-ambiguation.[1] However, to use EdgeLens follow long edges one must resort to wiggling the lens over an edge to cause the edge to vibrate. This is quite effective while the motion persists but becomes more difficult when the long edges also have low angular resolution and of course is no longer effective once the motion stops. However, since graph exploration tasks are frequently sequential, it is not always possible maintain this motion (see Section 7 for further explanation). Edge Plucking offers a new interaction metaphor that specifically addresses these issues, further enabling interactive graph exploration.

4. DESIGNING A NEW INTERACTIVE TOOL

We start by developing a set of design criteria, keeping in mind the issues of long edges and edges with low angular resolution which have as of yet not been adequately addressed with an interactive tool.

- **Reduce edge congestion:**
  - **Address low angular resolution:** by temporarily increasing angular resolution to improve graph readability.
  - **Make long edges easy to follow:** by using visual interaction to make it easier to distinguish one particular edge from others adjacent to it. Motion can aid graph reading. For instance, one can easily see edges of interest when they vibrate.[18]
  - **Alleviate occlusion caused by edges:** by allowing people to temporarily see what has been occluded by densely located edges without fundamentally reorganizing the graph.
  - **Disambiguate overlapping edges.** by temporarily curve edges to varying degrees to clarify overlapping edges.

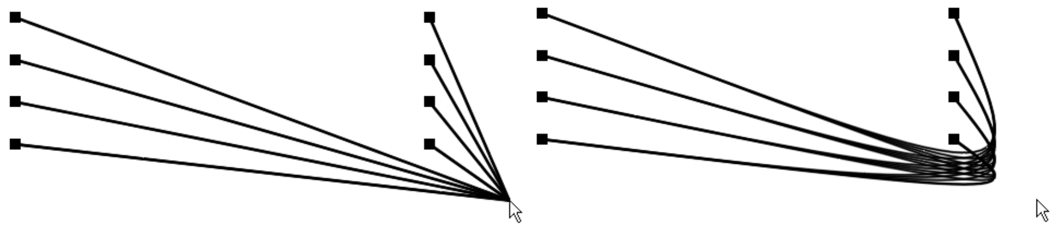
- **Minimize disturbance:** The basic assumption is that the graph contains useful information and that some of this information may be represented in how the graph has been laid out, for instance, the spatial location of the nodes. Not disturbing such a graph layout helps to preserves one's mental map.[19] Two design goals emerge from this criterion.
  - **Preserve node properties.** All node properties such as position, size, shape, etc. should be unaffected.
  - **Make adjustments temporary.** All adjustments should be temporary. One way to minimize the overall interference with the graph is to ensure that after use, all adjusted edges will be returned to their original locations and shapes.
- **Support interactive control:**
  - Support selection of individual edges or groups of edges.
  - Provide naturalistic visual response by following the plucking metaphor. For instance, the edge(s) should stay with the mouse cursor and should bend in an intuitive manner.
  - Provide the ability to pin plucked edge(s) to support the possibility of sequential unravelling of a heavily congested region.
  - Minimize introduction of edge crossings, sharp edge bends and inflection points.

The intention is to provide an intuitive metaphor with which this interactivity can be achieved, while minimizing layout disturbance and keeping interactions temporary.

For metaphor development, consider if one was to imagine for a minute being confronted with a physical representation of a node and edge diagram made, for instance, with rubber bands to represent edges. If the number of rubber bands was high, a natural response would be to sequentially separate edges, plucking them apart with one's fingers. We created an interactive tool based on this plucking metaphor and adhering to our set of design criteria.

5. THE DEVELOPMENT OF EDGE PLUCKING

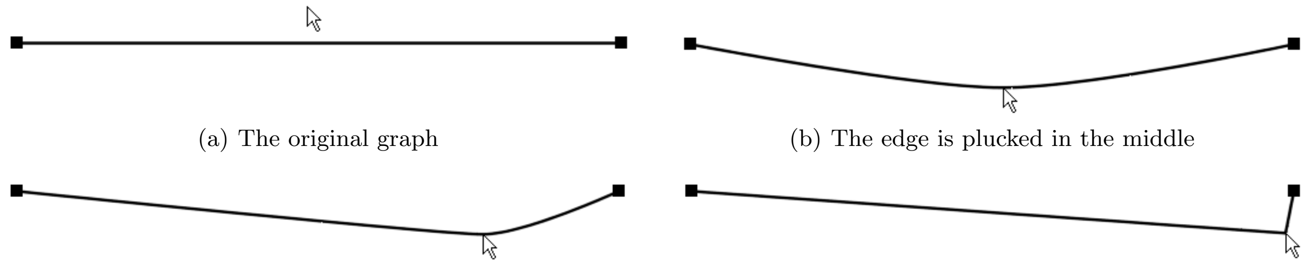
To develop Edge Plucking, we need to think about how to shape the plucked edges. A simple approach would be to separate an edge into two lines when it is plucked. The two lines would connect at the mouse cursor, with one line attaching to each node (Figure 7(a)). This is simple and reasonably effective but we felt that the visual response was not natural and the connecting point of the two straight lines formed a sharp angle. Another approach is to use a spline with a control point at the cursor. Once again the edges can be moved aside but interaction is difficult because the mouse and the edges that are being moved are not connected (Figure 7(b)).



(a) Using two straight lines (b) Using a single spline

Figure 7. Plucking action with two different early approaches

To get the visual response we were looking for we tried several different splines including interpolating variations.[20] The following is the most satisfactory results we obtained. Visually, it combines a direct connection with the mouse cursor and slightly curved edges. In addition, it includes a variant response depending on the location of the point on the edge from which plucking is initiated. This variation in visual response appears quite natural in that the longer part of the edge appears more stretched and the shorter part of the edge seems to have somewhat less tension. Also, the shape of the curve gets tighter, in that the bend gets much sharper, as the pluck point approaches the node. While this does introduce the possibility of a sharp bend, it also keeps the shape of the edge more neatly within the boundaries set by the edge's nodes. This is illustrated in Figure 8: Figure 8(a) is the untouched edge; Figure 8(b) through 8(d) shows different shapes of a plucked edge from smooth to a sharper curve.



(a) The original graph

(b) The edge is plucked in the middle

(c) The edge is plucked on the right side of the edge

(d) The edge is plucked very close to the right node

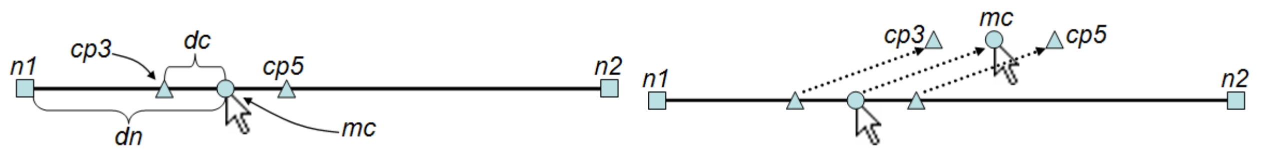
Figure 8. An edges is being plucked in different ways

A plucked edge is composed of two connecting cubic Bézier curves. They are controlled by seven control points (*cp*), where *cp1*, *cp2*, *cp3*, and *cp4* control one curve, and *cp4*, *cp5*, *cp6*, and *cp7* control the other curve. Control points *cp1* and *cp2* have the same location at node *n1*, and control points *cp6* and *cp7* are both located at node *n2*. The middle control point, *cp4*, the one that is shared by both curves is located at the mouse cursor *mc*.

When the mouse cursor touches the edge at *mc*, the edge is considered as two line segments joined at *mc*. The first step is to assess which line segment *n1* to *mc* or *mc* to *n2* is shorter. The control point on the shortest line segment will be calculated first,and the remaining control point will be calculated afterwards. In Figure 9(a) this is the segment *n1* to *mc* and *cp3* will be calculated using the formula:

$$dc = dn \cdot r \tag{1}$$

where *dc* is the distance from *cp3* to *mc*, *dn* is the distance from *n1* to *mc*, and *r* is a number between 0 and 1. While *r* can be interactively adjusted, the current default for *r*, as used in the illustrations in this paper, was arrived at through observation and is 0.3. The control point for the longer line segment, in Figure 9(a) *cp5*, is placed on longer line segment using the same distance *dc* just calculated for the shorter line segment. In the example in Figure 9(a) *cp5* is placed on the segment *mc* to *n2* at a distance *dc* from *mc*. This places the two calculated control points equidistant from *mc* and all three are on the original edge and thus are collinear.



(a) Illustrate how *cp3* and *cp5* are calculated

(b) *Cp3* and *cp5* move in the same direction and distance as the cursor

Figure 9. Illustrations of Edge Plucking algorithm

At this point the mouse has touched the edge and we are ready for the edge to be plucked. The distance from *cp3* to *mc* equals the distance from *mc* to *cp5* and all three control points are collinear. When the mouse cursor moves, and consequently *mc*, this co-linear relationship will be maintained. The three points move in the same direction and distance as *mc* moves, maintaining a parallel relationship to the original edge (Figure 9(b)).

That is, all three points will be moved in unison. Maintaining this co-linearity and using cubic Bézier splines, in spite of the fact that the end control points must be doubled at the nodes in order to get four control points per segment, does provide us with C^2 continuity at the location of the mouse cursor. This visual continuity seems important to provide the feedback that the edge is still a single edge and is responding as a unit.

Note that the locations of *cp3* and *cp5* depend on how close *mc* is to *n1* or *n2*. This dependency is essential to the characteristic of variable smoothness of plucked edges and allows the shape of the edge to be different on each side of the cursor. The closer the mouse cursor is to the centre of the edge, the smoother the plucked edge will be. As an opposite effect, the closer the cursor is to the end of the edge, the sharper the plucked edge will be. However, the previous discussion about visual continuity remains true even when the curve at *mc* is much sharper due to the choice of a pluck point close to a node.

5.1. The Interactive Control of Edge Plucking

To pluck edges, one must touch the edge(s) with the cursor, click, hold down the mouse button, and move the mouse as desired. Moving across edges while holding the left mouse button down collects the edges touched under the cursor. Continued motion will pull the edges with the cursor. When the middle mouse button is held down instead, the cursor only plucks the first edge it comes in contact with while keeping all other edges untouched. Edges can then be either released and they will return to their original positions or edges can be pinned at any chosen location.

One can pin edges that are being plucked by clicking the right mouse button while still holding the left or middle mouse button down. Once edges are pinned, their shapes and locations are fixed, i. e., they will not be affected by other plucking actions. Pins can be removed at will by right-clicking where the pins are located, at which point in time all edges fixed by the pin just removed will return to their original position. However, one cannot remove pins when the cursor is still plucking edges. This allows the possibility to pin multiple edges to the same location at different times. Note that in this situation where multiple pins are placed at the same location, they can be removed in a single action. This means that when right-clicking on that location, all pins are removed and all edges are returned to their original shapes. Also note that edges pinned as a group cannot be unpinned individually.

6. EDGE PLUCKING IN ACTION

In this section we show how Edge Plucking can be used to address the graph readability issues introduced in Section 2.

1. **Occlusion:** Plucking an edge near a node will reveal whether the edge is connected to or passing across the node (Figure 10). Or in the case where many edges obscure the structure of the graph, Edge Plucking can be used to move the edges away from a region of interest to reveal hidden structures (Figure 11). Also, one can pluck an edge or a collection of edges and move them aside to read background information (Figure 12(c)).

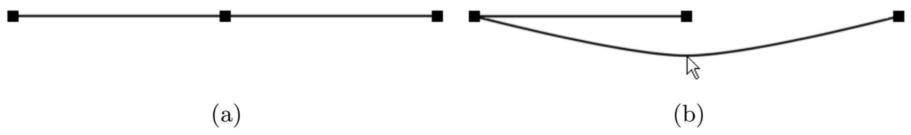
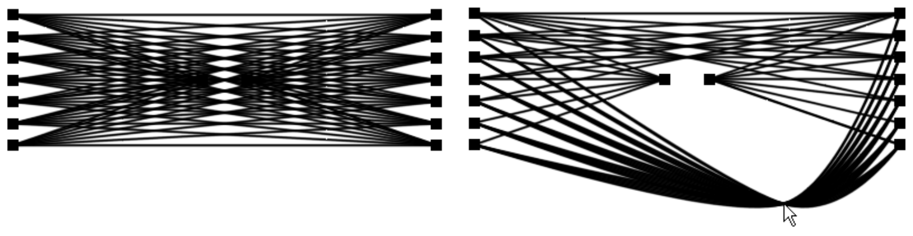


Figure 10. Using Edge Plucking to disambiguate a three-node graph

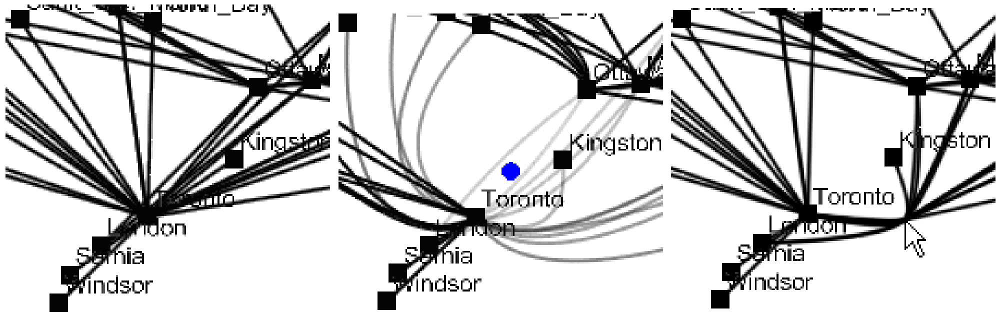
2. **Ambigious edges:** Figure 10 shows how Edge Plucking can be used to disambiguate overlapping edges. The node-edge relationship in the graph in Figure 11(a) is unclear. With Edge Plucking, we can see the actual configuration of the graph (Figure 11(b)).

3. **Low angular resolution:** In Figure 13(a), the angular resolution of the node on the left is low, but can be interactively increased when Edge Plucking is applied (Figure 13(b)).



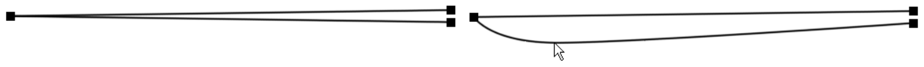
(a) (b)

Figure 11. Using Edge Plucking to reveal hidden nodes



(a) The original graph (b) EdgeLens is applied (c) Edge Plucking is applied

Figure 12. A graph with the label "Toronto" occluded



(a) (b)

Figure 13. Using Edge Plucking to increase angular resolution

4. **Long edges:** Ware and Bobrow's[18] study confirmed that motion can help graph exploration. When a long edge (or group of edges) is plucked, it draws attention to its nodes when wiggled.

5. **Edge bends and inflection points:** Since plucking an edge changes its shape it can be used to clarify ambiguities arising from existing edge bends and inflection points.

7. CASE STUDY

In this section, we present a case study: using Edge Plucking and EdgeLenses separately to explore a graph that represents a social network. The graph in Figure 14 consists of two clusters of nodes. Each cluster represents a class of students. Nodes represent students and edges represent friendships between students.

The tasks are 1) to find out if Ross is in Class A or B; 2) to find out whether Ross has friends in the other class; 3) if he has, find out their names.

Figure 14(a) shows the original graph where most names are occluded by edges. Using Edge Plucking, we can pluck edges away to read the originally hidden names. As shown in Figure 14(b), Ross can be found in Class B by plucking edges. Once we located Ross, we can then pin the edges so that they do not move back. This allows us to clearly see the name while we continue to do the tasks. Using EdgeLens, we can look for Ross by pushing edges away from the lens. We need to leave the EdgeLens at that location in order to keep the name shown. The difference between Edge Plucking and EdgeLens in this situation is that the name can be clearly disclosed by plucking edges while that name is partially overlapped by semi-translucent edges with EdgeLens (Figure 15(a)).

The next step is to check if Ross has friends in Class A. By observing the graph we can see that there is an edge connecting Ross to someone in Class A. To make it clear to where that edge is connected, we can pluck that edge away from all other edges (Figure 14(c)) and then pin it so that we do not lose the targeted node in Class A. Notice that only one edge is plucked in this step. Using EdgeLens is an alternative to find out the connecting node in Class A. We need to add a second EdgeLens to the graph because the first EdgeLens needs to stay at the original place to keep the name Ross shown. The connected node can be discovered by wiggling

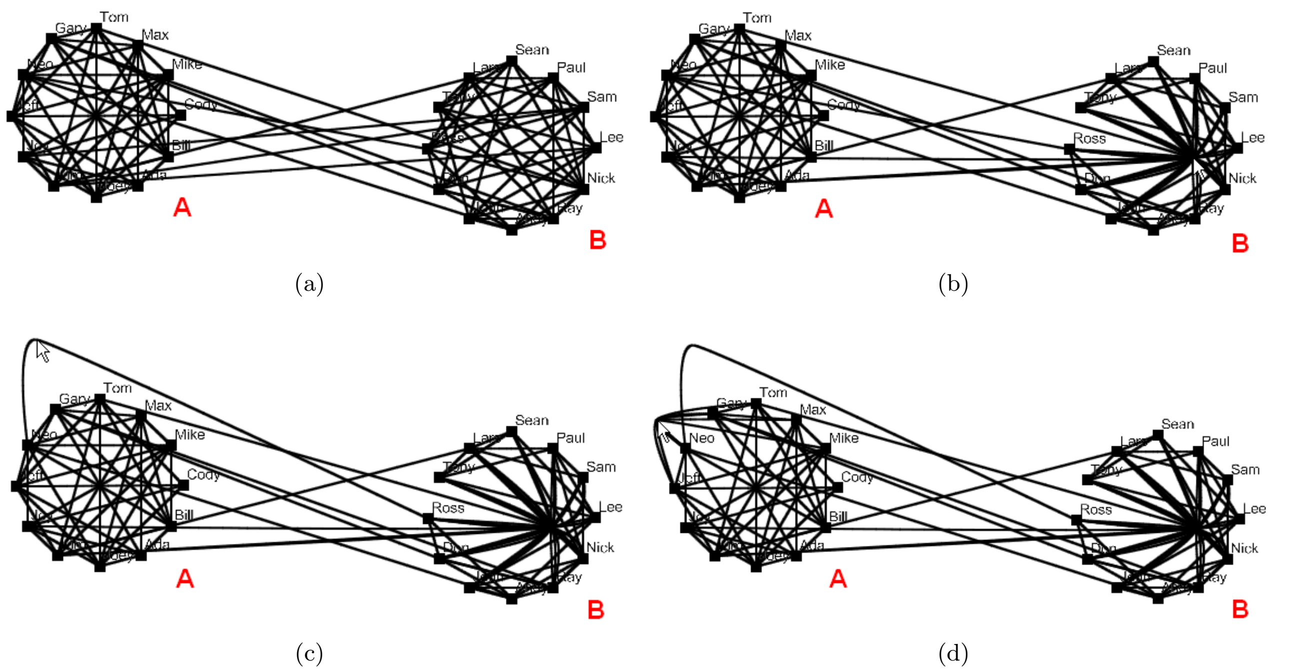


Figure 14. Exploring a social network with Edge Plucking: (a) the original graph; (b) find out Ross is in Class B; (c) the edge connecting Ross to Class A is plucked and pinned so that the edge is pointing right towards the targeted node; (d) edges are plucked away to reveal the name Neo

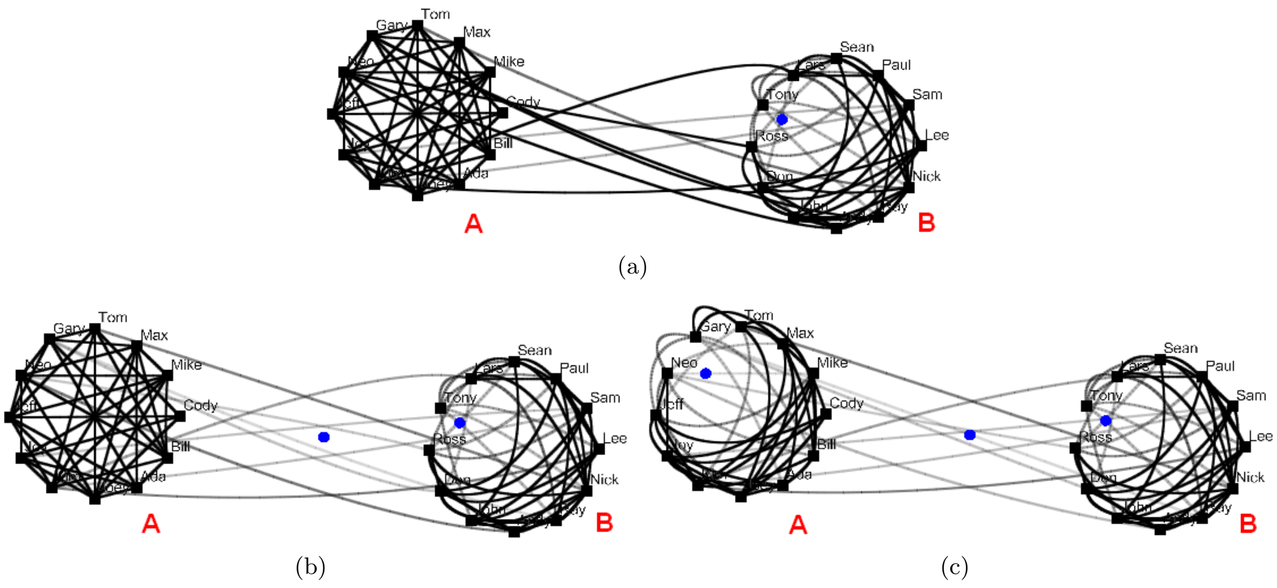


Figure 15. Exploring a social network with EdgeLenses: (a) find out Ross is in Class B; (b) to find the connecting node by wiggling an EdgeLens; (d) edges are pushed away from the lens centre to reveal the name Neo. Note that the names are covered by semi-translucent edges.

the second EdgeLens across the long edges (Figure 15(b)). However, this is also a limitation of EdgeLens. In order to keep a focus on the targeted node, we need to produce constant motions; whereas, with Edge Plucking, we can pin the edge in a way that it points directly to the node (Figure 14(c)).

The last step is to find out the name of Ross' friend. We can pluck away the edges that are covering the name (Figure 14(d)) and see that it is Neo. This step is relatively difficult to accomplish using EdgeLens. Since constant motions are necessary to keep the targeted node in focus, we have to remember the location of the targeted node once we stop moving the EdgeLens. To reveal the name we add a third EdgeLens to the node that we memorized (Figure 15(c)).

In this case study, we use Edge Plucking to pluck edges away to reveal hidden information (names). To confirm the relationship between nodes, we pluck a single edge while keeping all other edges unaffected. We can explore the graph and complete the tasks by plucking (groups of edges and a single edge) and pinning. We also completed the tasks using EdgeLenses. However, this involves extra memory load as we need to remember node positions which interrupts the flow of sequential tasks.

8. COMPARING AND CONTRASTING EDGE PLUCKING WITH EDGELENS

While Edge Plucking and EdgeLens are very different in metaphor, there are strong similarities between Edge Plucking and EdgeLens. They are both able to address many readability issues in edge congested graphs, however, Edge Plucking does have advantages over EdgeLens in certain situations. For instance, in the situation of revealing background information, EdgeLens reveals the background partly by pushing edges aside and partly through use of translucency (Figure 12(b)); in contrast, Edge Plucking can just move the occluding edges aside (Figure 12(c)). Also, as shown in the previous section, Edge Plucking has better support for sequential tasks than EdgeLens.

Since Edge Plucking can be applied to one or more edges, while EdgeLens is applied to a given region, Edge Plucking seems more amenable to a piece by piece exploration of graph structure. For example, consider exploring the small portion of a circular graph structure shown in Figure 16(a). There are seven nodes connected by several edges that are all lying along the slight circular arc. Applying an EdgeLens to the region of interest does reveal the structure (Figure 16(b)); however, it can be difficult to choose just the right position to achieve a reasonable spread. The edges and the structure is still somewhat difficult to decipher. Instead, one can use Edge Plucking (Figure 16(c) to 16(h)) more deliberately. Following through from Figure 16(c) each edge is plucked and pinned to the side, gradually revealing the structure. Pinning all six edges clarifies the relationships between nodes. While Edge Plucking is effective with larger graphs, we have illustrated this discussion with small graphs so that the static images will be clear. Much of the effectiveness is a result of the motion plucking causes.

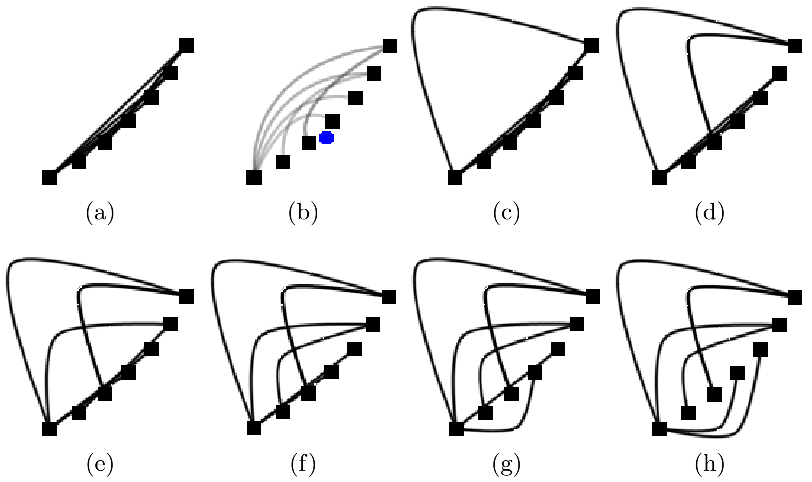


Figure 16. Exploring a cluster of nodes in a graph: (a) the original graph; (b) EdgeLens is applied; (c) through (h) Edge Plucking is used to separate the edges

Table 1 compares and contrasts Edge Plucking with EdgeLens on five readability issues listed in Section 2. Note that for disambiguating edges that are closely located or overlapped and for making long edges easier to follow both techniques seem equivalently effective. On the other hand, for addressing occlusion and for clarifying the relationships among edges with low angular resolution Edge Plucking is more effective. However, Edge Plucking can introduce new bends, particularly when the edge is plucked close to a node. Algorithmically ensuring that these bends are always C^2 continuous helps clarify the plucked edge as still being a single edge.

Edge Plucking and EdgeLens each facilitate different graph exploration techniques. EdgeLens provides a quick overview of basic structure, while Edge Plucking lends itself to more careful and precise exploration of the structures. We intend to continue with this comparison and use empirical means to determine which methods are best suited to which tasks.

Problems	**EdgeLens**	**Edge Plucking**
Occlusion (e. g. node layout, labels, or background information are covered)	• edges can be slightly pushed away from the occluded area • edges cannot be entirely removed from the area • translucency helps to reveal the occluded information	• edges can be plucked away from the occluded area as far as wanted • edges can be entirely removed from the area
Ambiguous edges (e. g. closely located or overlapped edges)	• ambiguous edges can be separated	• ambiguous edges can be separated
Low angular resolution	• edges can be slightly pushed away from each other to increase the angular resolution	• edges can be plucked away from each other as far as wanted; thus significantly increasing the angular resolution
Long edges are difficult to follow	• long edges become easy to follow because of the motion EdgeLens causes	• long edges become easy to follow because of the motion Edge Plucking causes
Edge bends	• no sharp bends	• it is possible to introduce sharp bends depending on where the edges are plucked, though C^2 continuity is always preserved

Table 1. Comparing and contrasting Edge Plucking with EdgeLens on readability issues in edge congested graphs

9. CONCLUSION

In this paper we introduce Edge Plucking, an interactive technique for mitigating edge congestion in graphs. Edge Plucking has the following merits: interactively and temporarily moves edges; preserves node positions; separates overlapped edges; increases angular resolution; reveals hidden information that is occluded by congested edges; and makes long edges easy to follow.

In addition, Edge Plucking offers some advantages over the earlier interactive technique EdgeLens. The interaction metaphor behind EdgeLens is based on repelling edges from the cursor's location, the metaphor of plucking and moving edges aside may be more natural. Edge Plucking is more effective at clearing edges off background information and seems better suited to the exploration of graph structure. One can also use Edge Plucking to pluck a single edge and pin it in a way that the edge points directly towards a node to keep the focus; thus, has better support for sequential tasks. As we continue this work we will study these differences empirically.

REFERENCES

1. N. Wong, S. Carpendale, and S. Greenberg, "Edgelens: An interactive method for managing edge congestion in graphs," in *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2003)*, pp. 51–58, 2003.
2. S. Carpendale and X. Rong, "Examining edge congestion," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2001), Interactive posters: visualizing, Vol. 2*, pp. 115–116, 1993.
3. H. Purchase, "Evaluating graph drawing aesthetics: Defining and exploring a new empirical research area," in *Computer Graphics and Multimedia: Applications, Problems and Solutions*, J. DiMarco, ed., pp. 145–178, Idea Group Publishing, 2004.
4. A. Inselberg, "Multidimensional detective," in *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '97)*, pp. 100–107, 1997.
5. H. Purchase, R. Cohen, and M. James, "Validating graph drawing aesthetics," in *Proceedings of the Symposium on Graph Drawing (GD '95)*, pp. 435–446, 1995.
6. G. Di Battista, P. Eades, R. Tamassia, and I. Tollis, "Algorithms for drawing graphs: An annotated bibliography," *Computational Geometry: Theory and Applications* **4**(5), pp. 235–282, 1994.
7. I. Herman, G. Melancon, and M. Marshall, "Graph visualization and navigation in information visualization: A survey," *IEEE Transactions on Visualization and Computer Graphics* **6**(1), pp. 24–43, 2000.
8. G. Wills, "Nicheworks—interactive visualization of very large graphs," *Journal of Computational and Graphical Statistics* **8**(2), pp. 190–212, 1999.
9. T. Henry and S. Hudson, "Interactive graph layout," in *Proceedings of the ACM SIGGRAPH Symposium, Proceedings ACM Siggraph Symposium on UI Soft-ware*, pp. 55–64, 1991.
10. M.-A. Storey and H. Mller, "Graph layout adjustment strategies," in *Proceedings of the Symposium on Graph Drawing (GD '95)*, pp. 487–499, 1995.
11. T. Jankun-Kelly and K. Ma, "Moiregraphs: Radial focus+context visualization and interaction for graphs with visual nodes," in *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2003)*, pp. 59–66, 2003.
12. C. Ware and M. Lewis, "The dragmag image magnifier," *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '95), Videos* **2**, pp. 407–408, 1995.
13. Y. Leung and M. Apperley, "A review and taxonomy of distortion-orientation presentation techniques," *ACM Transactions of Computer-Human Interaction* **1**(2), pp. 126–160, 1994.
14. S. Mukherjea and J. Foley, "Visualizing the world-wide web with the navigational view builder," *Computer Networks and ISDN Systems* **27**(6), pp. 1075–1087, 1995.
15. R. Becker, S. Eick, and A. Wilks, "Visualizing network data," *IEEE Transactions on Visualization and Computer Graphics* **1**(1), pp. 16–28, 1995.
16. K. Cox, S. Eick, and T. He, "3d geographic network displays," *ACM Special Interest Group on Management of Data Record* **25**(4), pp. 50–54, 1996.
17. T. Munzner, E. Hoffman, K. Clarify, and B. Fenner, "Visualizing the global topology of the mbone," in *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '96)*, pp. 85–92, 1996.
18. C. Ware and R. Bobrow, "Motion to support rapid interactive queries on node-link diagrams," *ACM Transactions on Applied Perception* **1**(1), pp. 3–18, 2004.
19. K. Misue, P. Eades, W. Lai, and K. Sugiyama, "Layout adjustment and the mental map," *Visual Languages and Computing* **6**(2), pp. 183–210, 1995.
20. N. Dyn, D. Levin, and J. Gregory, "A four-point subdivision scheme for curve design," *Computer Aided Geometric Design* **4**(4), pp. 257–268, 1987.