Ambiguity-Free Edge-Bundling for Interactive Graph Visualization

Sheng-Jie Luo, Chun-Liang Liu, Bing-Yu Chen, *Member, IEEE,* and Kwan-Liu Ma, *Senior Member, IEEE*

Abstract—Graph visualization has been widely used to understand and present both global structural and local adjacency information in relational datasets (e.g., transportation networks, citation networks, or social networks). Graphs with dense edges, however, are difficult to visualize because fast layout and good clarity are not always easily achieved. When the number of edges is large, edge bundling can be used to improve the clarity, but in many cases, the edges could be still too cluttered to permit correct interpretation of the relations between nodes. In this paper, we present an ambiguity-free edge-bundling method especially for improving local detailed view of a complex graph. Our method makes more efficient use of display space and supports detail-on-demand viewing through an interactive interface. We demonstrate the effectiveness of our method with a public coauthorship network data.

Index Terms—Graph Visualization, Network Visualization, Edge Ambiguity, Edge Congestion, Edge Bundling, Detail-on-Demand, Interactive Navigation.

1 Introduction

Visualizing relational datasets as graphs is a common approach. In graph visualization, a graph consists of nodes and edges where the nodes represent some entities and the edges represent the relationships between the entities (nodes). Its aim is to succinctly reveal the structural and relational information in the datasets. As the size of a graph grows, with a limited display space, laying out the graph directly would lead to two potential problems: *edge-congestion* and *edge-ambiguity*. The former is due to dense edges which may decrease the readability of the graph, while the latter is due to inappropriate edge layout, which significantly decreases the number of possibly and correctly perceivable relations from the data.

Edge-congestion and *edge-ambiguity* problems make a graph difficult to read and use. Several techniques have been introduced to relieve these problems. Among these techniques, *edge-bundling* [1] [2] [3] [4] [5] [6] is a particularly effective solution, which reduces the visual clutter caused by dense edges by merging some groups of edges together to use less screen space for the same amount of information. It has been quickly adopted by the graph visualization community, and there are many approaches to bundle edges, such as Hierarchical Edge Bundling [2], Geometry-Based Edge Bundling [4], and Force-Directed Edge Bundling [6] , etc. They all improved the edge layout to some degree and can help to visualize an entire graph with excessive edges. Specifically, these methods provide better visualization of the structural information of the whole graph.

- *S.-J. Luo, C.-L. Liu, and B.-Y. Chen are with National Taiwan University. E-mail: {forestking,being31}@cmlab.csie.ntu.edu.tw, robin@ntu.edu.tw*
- *K.-L. Ma is with University of California at Davis. E-mail: ma@cs.ucdavis.edu*

The goals for visualizing a graph in overview and detail-view are different. Visualizing a graph in overview seeks to provide a clearly visible high-level edge pattern in the whole graph. Under this criteria, the perception of precisely which two nodes are connected is relatively less important, and thus displaying detailed connectivity information in local areas of the graph is usually compromised. On the contrary, visualizing a graph in detail-view seeks to provide accurate connections between the nodes. Precisely discriminating which two nodes are connected is essential to allow the viewer to trace a path and answer some specific questions. If the subgraphs of interest have highly connected nodes, we still need to address both the *edge-ambiguity* and *edge-congestion* problems in such local areas.

In this paper, we present an ambiguity-free edge-bundling method coupled with an interactive (un-)grouping interface. This method was especially designed to reduce the confusion caused by the cluttered edges in the detail-view. Our method improves users' ability to accurately perceive individual relationships since the edges with the same target or source node(s) are merged and curved and the curvature is automatically adjusted to avoid ambiguities. We employ a quadtree structure to overcome the time complexity issue, so users can freely interact with the graph. Through the easy-to-use, interactive detail-on-demand interface, the users can select some local areas to focus on by simply enclosing the corresponding subgraphs with a stroke. Our method effectively removes the shortcomings of the original edge-bundling layout method and, in particular, enhances the visualization of graphs with locally dense edges. As shown in each example presented in this paper, the subtle difference that our method can be critical in correctly comprehending and dissecting a graph.

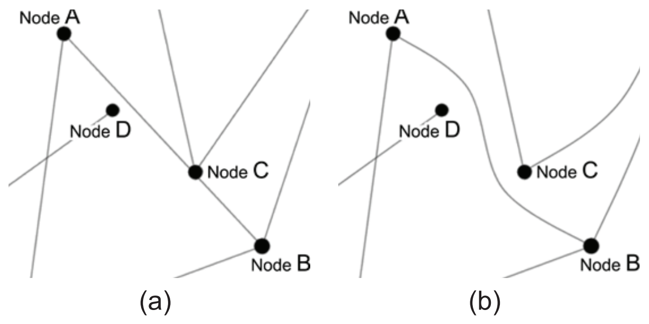

Fig. 1. (a) The case in which an edge passes near or even through some unrelated nodes causes incorrect perception of relationships. (b) The result of routing and curving the edge away from the unrelated nodes.

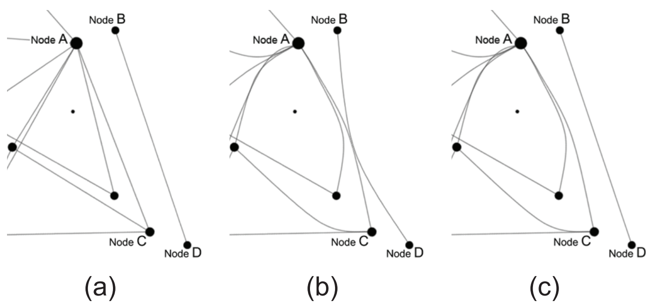

Fig. 2. (a) The original graph. (b) The result of an existing edge-bundling method, and there are some incorrect relations due to the edge-bundling. (c) The result of our edge-bundling while considering to avoid the edge-ambiguity problem.

1.1 Edge Ambiguity

Most of the edge-ambiguity problems happen when an edge overlaps unrelated nodes. As illustrated in Fig. 1(a), the center edge, according to the real dataset, actually connects *Node A* and *Node B* while *Node C* is only incidentally located on the path of the edge. This case may cause users to incorrectly perceive that *Node A - Node C* and *Node B - Node C* are connected, respectively. Moreover, even though the edge does not actually overlap *Node D*, the geometrical nearness could still confuse a user's visual perception of the connections and then cause an ambiguity problem. The basic idea of solving the edge-ambiguity problem is illustrated in Fig. 1(b). Obviously, the inappropriate overlapping and nearness problems are avoided. The routed edge uses a smooth curve to bypass the unrelated nodes. The smoothness improves the users' ability to trace the curve. Furthermore, a routed edge can always choose a bypassing path that minimizes the number of crossings.

Although edge-bundling is usually used to reduce the visual clutter problem, it may sometimes introduce its own edge-ambiguity problem. Most edge-bundling methods merge the edges together if those edges are both geometrically close and in similar direction as illustrated in Fig. 2(b). According to the actual relationship shown in Fig. 2(a), *Node A - Node C* and *Node B - Node D* are connected, respectively, and these two relations are represented by two individual edges. However, since the two edges are close and in similar direction, if we bundle these two edges together as shown in Fig. 2(b), it becomes uncertain whether *Node A* is connected by either *Node C* or *Node D* or both. Using edge-bundling for showing a graph's overview is reasonable since its goal is to provide a clearly visual edge pattern. However, when it comes to the detail-view, any perception problem becomes crucial, since precise and accurate relationships between nodes is usually necessary. With our improved edge-bundling method, the ambiguity problem is addressed. Hence, our result has no such edge-ambiguity problem as shown in Fig. 2(c).

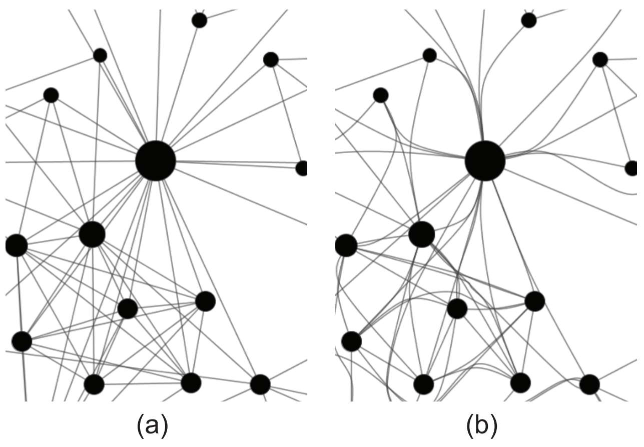

Fig. 3. (a) The graph contains several dense nodes and edges in some local regions, where high-degree nodes gathering in the local regions decrease the readability of the graph. (b) Bundling the edges, which connect to the high-degree nodes, together can relieve the local congestion.

1.2 Edge Congestion

Most edge-congestion problems in the detail-view are caused by high-degree nodes, which connect to many other nodes. These high-degree nodes are usually clustered into one small region. This is because most relational datasets have small-world characteristics [7] [8] [9], which tend to make the graphs contain more subgraphs with high-degree nodes gathering in a few local regions and the edges in such regions usually have relatively short lengths as is shown in Fig. 3(a).

Our ambiguity-free edge-bundling method conditionally bundles the edges to relieve the edge-congestion problem in local regions. Ambiguities in the edge-bundling process are considered, and only the edges which share a common node can be merged. Fig. 3(b) shows our result. Comparing it to Fig. 3(a), the bundled edges better utilize the screen space which helps users to discriminate the connections of the high-degree nodes.

The smoothly curved-edges also help the users to trace the edges. Inappropriate overlapping and nearness problems, which often occur in traditional edge-bundling methods, are avoided.

2 Related Work

Correctly perceiving individual relations between nodes is often the goal of requesting and examining an detail-view of a graph in graph visualization. Several techniques have been introduced to address this requirement. van Ham used a matrix-based representation to reduce the visual clutter caused by edges [10]. However, the matrix-based representation is often less intuitive than using node-link graphs. Eades *et al.* proposed a method that only draws the edges between node clusters [11] [12]. Becker *et al.* proposed half-lines [13] to visualize a directed edges, which reduces the visual clutter by drawing only the first half part of the edge between two nodes. Existing popular methods that reduce the edge clutter may be classified into two major types, which are interactive approaches and edge-routing approaches. Interactive approaches refer to the ability to discriminate additional information by interactively selecting or filtering out current information. Fisheye views [14], a distortion-oriented method, is one of the examples, which distorts the content while still maintaining the user's mental map model [15]. Many extensive methods have been presented in the fields of information visualization [16] [17] [18] [19] as well as human-computer interaction [20] [21]. Through this method, a user can interactively manipulate a fisheye-like view to enlarge some local areas of a graph/network and examine the details of the confusing subgraph.

Another group of methods under the interactive approaches can be treated as the edge-dispersing techniques, which disperse the edges away from one local region, so that the underlying pattern can be revealed and edge-ambiguity can be temporarily relieved. For example, EdgeLens [22] allows users to interactively bend the edges away from one's focus without modifying the nodes' positions. The lens open up sufficient space to disambiguate the relationship between the underlying nodes. Edge Plucking [23] is another technique that allows users to pull edges away from the region of interest. The two methods also help users to clarify the clustered edges without edge-ambiguity problems. However, they require a lot of efforts from the users to interact with the local regions and to memorize the temporal discrimination between the edges.

On the other hand, edge-routing approaches tend to use a mathematical model to reduce edge crossings, which cause many of the edge-congestion and edge-ambiguity problems. Topology preserving constrained graph layout [24] treats the edges as rubber bands, which is a constrained graph layout algorithm that supports polyline edges and clusters where clusters/nodes/edges may not overlap. However, this algorithm only adapts well if the number of edges in the graph is relatively small and recognizable node displacements are produced.

Recently, many methods were proposed to provide clearer visual edge patterns even with an excessive number of edges. The ability to perceive individual relations is compromised, since perceiving the overall structure of the graph is the primary goal of visualizing the whole graph. In spite of this, the idea of relieving the edge-congestion problem can still be extended for visualizing the detail-view of the graph if we carefully consider the perception of individuals. Carpendale *et al.* first proposed an edge-displacement technique [25] by using a distortion-oriented formula to curve the edges that are passing by the circular region of interest. Dickerson *et al.* proposed a technique called confluent drawing [26] to visualize non-planar graphs in a planar way. The idea is to allow groups of edges to be merged together. Phan *et al.* presented the flow map layout [1] to visualize flow data. They automatically generate a flow map that depicts the backbone of a graph in an overview. However, it may cause the edge-ambiguity problem in the detail-view. Force-directed edge bundling [6] used a self-organized approach and modeled the edges as flexible springs, which can be pulled together to form bundles. In the detail-view, this method may also cause the edge-ambiguity problems due to inappropriate node-edge overlapping and bundling of unrelated edges. Hence, users may likely perceive incorrect relations, like the result shown in Fig. 2(b).

Cui *et al.* proposed an edge-bundling method [4] for revealing the high-level edge pattern based on the nodes' geometrical information. They used a uniform grid structure to sample the excessive edges, and then generate a control mesh to guide the edge-curving by bundling the edges that are geometrically close and directionally similar. The result of this method also provides a clear visual pattern of dense edges. However, the effectiveness of this method highly relies on the quality of the control mesh, and there is no guarantee that an effective mesh can always be obtained automatically. If this method is applied directly to visualize the detail-view of a graph, the result also shows that the curved-edges are forced to pass close to all nearby nodes. Thus, the relationship ambiguity for the bundled edges frequently happens because only the edges' geometries are taken into consideration. Hierarchical edge bundles [2] was designed to visualize the data that contains both adjacency relationship and hierarchical structure. To draw an edge linking two tree leaves, the edge is curved according to the path connecting the two nodes on the hierarchical tree structure. The algorithm bundles these edges together if they share a common path segment on the hierarchical tree. This method reduces the visual clutter and also visualizes the hierarchical pattern embodied in the straight-line graph. It performs well if the dataset contains hierarchically structured nodes.

There is another approach to deal with the edge-

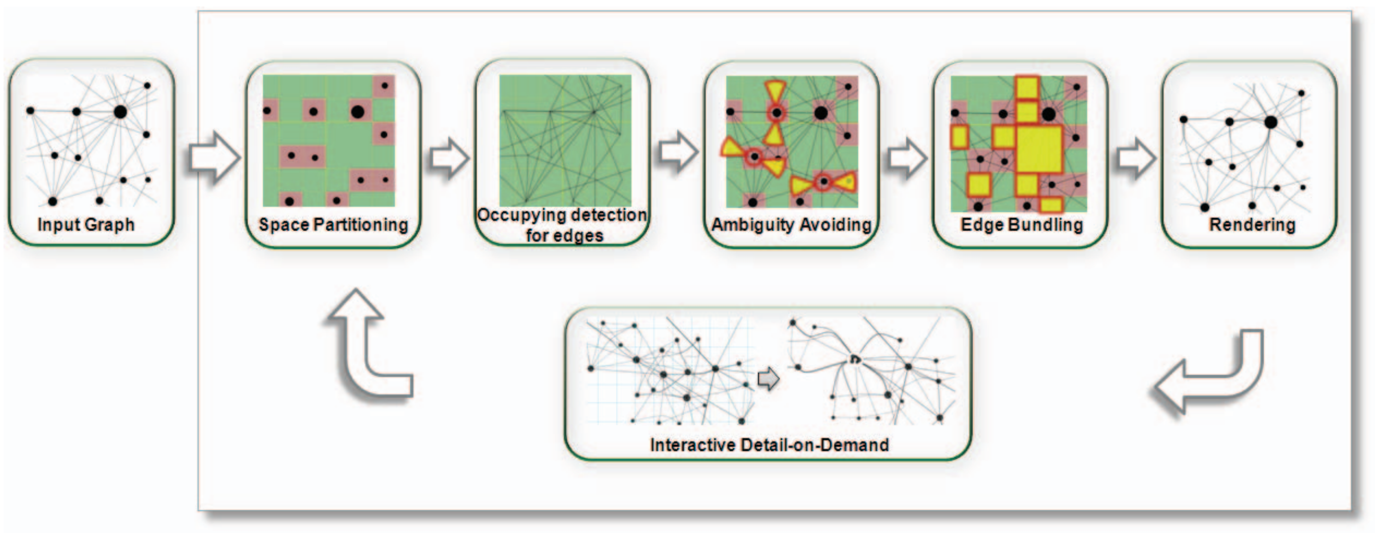

Fig. 4. An overview of our ambiguity-free graph visualization process.

congestion problem called the node-adjustment approach, which adjusts the nodes' positions instead of curving edges. To avoid content confusion, the approach rearranges the nodes' positions in such a way as to minimize the edge density, crossing, and occlusion. However, this approach is difficult to obtain an appropriate rearrangement with dense edges [27] [28]. Furthermore, it may not be suitable for the graphs, which the nodes' positions have semantic meanings, e.g., the nodes represent the cities in a map.

3 Ambiguity-Free Edge-Bundling

Fig. 4 illustrates an overview of our design. First, we perform a force-directed layout algorithm to place nodes in the display space. Then we use a quadtree [29] structure to decompose the two-dimensional display space according to the nodes' positions in the input graph, and this hierarchical data-structure provides a subdivision-based approach for further enhancing the efficiency in the following six major steps:

1) Space partitioning (Sec. 3.1).
2) Occupancy detection (Sec. 3.2).
3) Edge-ambiguity avoiding (Sec. 3.3).
4) Curved-edge bundling (Sec. 3.4).
5) Rendering (Sec. 3.5).
6) Interaction (Sec. 3.6).

The *space partitioning* step constructs a quadtree structure by using the nodes' positions. During this process, it also identifies which quadtree cell is occupied by which node(s) in the graph. The *occupancy detection* step detects which quadtree cell is occupied by which edge(s) in the straight-line mode. The *edge-ambiguity avoiding* step detects if there exists edges passing near one or more unrelated nodes. If such instances are detected, the algorithm creates some control points for routing these edges away from the unrelated nodes. The *curved-edge bundling* step geometrically bundles some matched edges together while conforming to the edge-ambiguity requirement. It further enhances the visual discrimination of the bundled edges between the related nodes and unrelated ones by separating them when necessary. Finally, the *rendering* and *interaction* steps provide the final visualization and interaction controls for users.

3.1 Space Partitioning

In order to achieve realtime interaction, an efficient data structure, quadtree, is first deployed to partition the space. The quadtree is subdivided according to the nodes' positions. For some applications, if the nodes in the dataset have no exact geometric information in advance, e.g., social network data, a force-based model [30] is first applied to compute the initial nodes' positions. A quadtree cell is recursively subdivided into four children, and the subdivision of a cell stops when either no node or exactly one node is in that cell. Fig. 5 illustrates a simple example with only seven nodes. For identification purposes, we define two types of cells in the quadtree structure, which are *Node Cells* and *Empty Cells*. A *Node Cell* contains exactly one node in its local region, and an *Empty Cell* contains no node and represents a usable area for later steps. Unfortunately, the size of a *Node Cell* may be too large, which would waste valuable screen space. Therefore, after constructing the quadtree, we further subdivide large *Node Cells* until they fall below a threshold size U, releasing more *Empty Cells* in the process. The result of this subdivision is shown in Fig. 5, where the nodes are all put into sufficiently smaller *Node Cells*. Note that, after this subdivision, *Node Cells* have a maximum size of U, but no minimum size, and *Empty Cells* have neither a maximum nor a minimum size.

The thresholding value U used as the stopping criterion plays an important role in constructing the quadtree. In our observations, if the value of the threshold U is too small, the following edge-bundling mechanism would produce a frequently-turning curve layout,

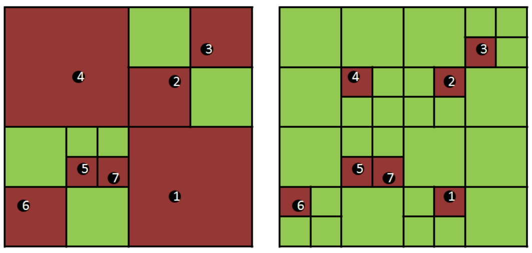

Fig. 5. The comparison between the space partitioning without (left) and with (right) further subdivision. The red cells (*Node Cells*) are the regions occupied by the nodes, and the green cells (*Empty Cells*) are free regions.

a.k.a. the zigzag effect. The zigzag curves change their edge direction frequently, which makes it more difficult to perceive the connections between the nodes. Moreover, the meshes in the quadtree will become much more dense. A dense space partition significantly decreases the performance of edge-bundling because the widths of the *Empty Cells* also become smaller. On the other hand, if the value of the threshold U is too large, the edge-ambiguity avoiding mechanism might not be able to properly find enough suitable *Empty Cells* for rerouting the ambiguous edges. With less *Empty Cells* being identified, the result of edge-bundling and searching the *Empty Cells* for avoiding ambiguity will become worse. In our experience, the value of the threshold U should be decided by considering both the number of nodes on the display and the area of the display space. Hence, we suggest the threshold U to be defined using the following equation:

$$U = \sqrt{A/4N}, \tag{1}$$

where A is the area of the display space and N is the number of nodes on the display.

3.2 Occupancy Detection

Quadtree-based space partitioning provides the ability to quickly identify any query related to node positions. If the available display space is sufficient for a standard edge layout, we consider straight-line edges as the best drawing style due to their simplicity. However, ample amounts of display space is not always enough to solve the visual clutter problem. In our design, the idea is to improve the edge style from the straight-lines to curves if the straight-line edge style has edge-congestion or edge-ambiguity problems. Thus, it is necessary to analyze which regions are occupied by which edge(s) in the straight-line style, and then use this information to detect the regions which have edge-congestion or edge-ambiguity problems.

To detect the edge-occupied cell, we propose a divide-and-conquer collision detection method based on the quadtree data structure. For each edge, the system finds two *Node Cells* C_1 and C_2 that contain two vertices of the edge. Then, the parent cell C_P in the lowest level of the quadtree that contains both C_1 and C_2 is found. The collision detection is performed recursively from C_P. It maintains a queue of cells that need to perform collision detection and adds C_P in the beginning of the queue. At every step, it takes a cell C from the queue, detects the collision of the edge and the four children of C, and adds the children that are collided by the edge back to the queue. Detection of edge-cell collision is done by computing the intersection of the edge and the cell's four boundaries. If the cell C is a leaf of the quadtree and is not a *Node Cell*, then C is intersected by strait-line edges (the sky-blue cells in Fig. 6). The process stops when the queue is empty.

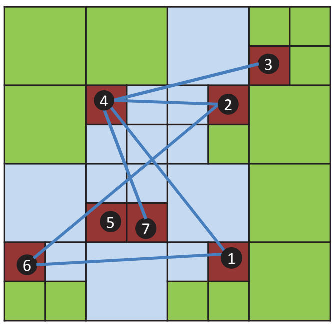

Fig. 6. Cells colored in sky-blue indicate the result of searching *Empty Cells* which are passed by specific straight-line edges.

The hierarchical directory of the space partition forms a cell map that allows us to quickly find which cells are occupied by which straight-line edge(s). If an edge on the cell map only passes through *Empty Cells*, there is a very low possibility that the edge will cause edge-ambiguity problems. On the other hand, if it passes through one or more *Node Cells*, the edge-ambiguity problems might occur in those *Node Cell* regions.

3.3 Edge-Ambiguity Avoiding

The edge-ambiguity problem in graph visualization often leads users to perceive incorrect relationships between nodes or requires considerable effort to read the graph. In the previous step, most possible edge-ambiguity locations are identified by detecting the *Node Cells* that intersect with straight-line edges whose destination or source is not one of the nodes contained in that cell. After determining all such *Node Cells*, which may contain an inappropriate overlap or nearness problem, we then calculate the control point(s) for every edge and curve the local part of the involved edge(s) to pass through the control point(s). Thus, the involved edge(s) avoid any inappropriate overlap or nearness problem. The position of the control point for bypassing is determined by the following three crucial conditions:

- Whether this edge-ambiguity problem is solved or not.
- Whether additional edge-ambiguity problem(s) will occur or not by bypassing this location.

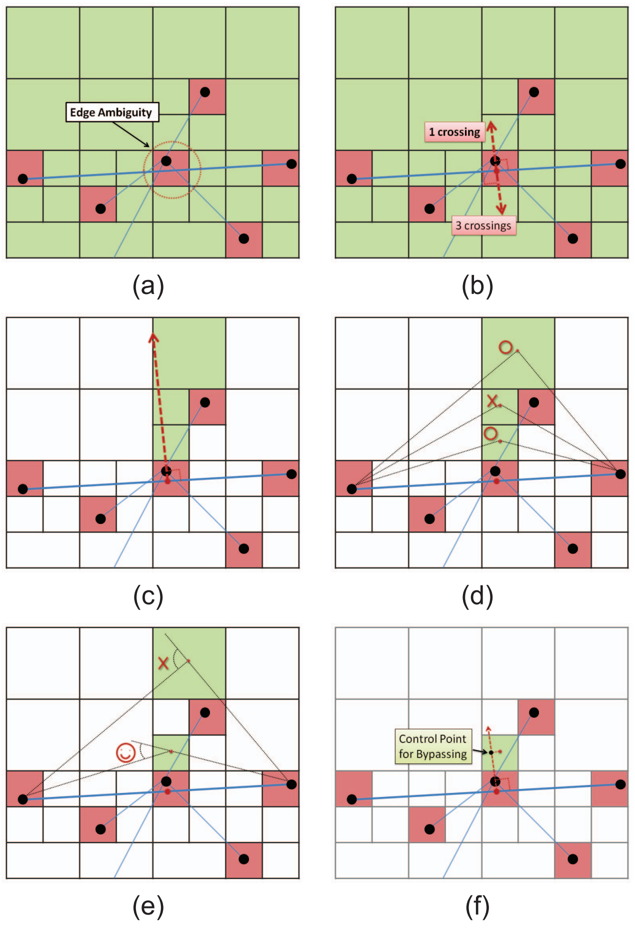

Fig. 7. The step of determining the control points for bypassing an edge. (a) Find the cell with an edge ambiguity problem. (b) Calculate the two vectors that are perpendicular to the involved edge, and choose the one causes the least number of edge-crossings. (c) Find the candidate *Empty Cells* to curve the edge. (d) Remove the candidates that will make the curved-edge cause edge-ambiguity problems. (e) Choose the cell from the remaining candidates that has the best continuity for the curved-edge. (f) Calculate the precise position of the control point.

- Whether the representation of this bended edge improves the readability of the graph and makes it easier to read or not.

Based on some graphic aesthetic guidelines [31] [32], we define the rules for bending the edges as follows:

1) The number of edge-crossings should be reduced.
2) The continuity of the curved-edge should be maintained.
3) The number of bends of the curved-edge should be as small as possible.

Keeping the number of bends of the curved-edge as small as possible and considering the continuity of the curved-edge could make the curved-edge smoother and easy-to-track, while reducing the number of edge-crossings could reduce the visual clutter. Hence, the procedures for deciding on a control point for bypassing an edge are listed in sequence as follows (also see Fig. 7):

1) Calculate the two vectors that are both perpendicular to the line-segment of the involved edge. (Fig. 7(a))
2) In these two vectors, choose one as the vector **V** which will make the bypassing curve cause the least number of edge-crossings with the connected edges of the involved node. (Fig. 7(b))
3) Find the candidates of the bypassing *Empty Cells* by searching a line-segment starting from the projection point of the involved node along the direction of the vector **V**. (Fig. 7(c))
4) Remove any candidate of the bypassing *Empty Cells* that will make the curved-edge cause edge-ambiguity problems for other nodes. (Fig. 7(d))
5) If there remains more than one candidate, choose the one that has the best continuity for the curved-edge. (Fig. 7(e))
6) Calculate the precise position of the control point for bypassing the curved-edge. The specific position of the control point is the projection point of the center of the best candidate on the line-segment formed by the position of the involved node and vector **V**. (Fig. 7(f))

In order to ensure that the control point(s) for bypassing an edge will guide the curved-edge away from the inappropriate overlap or nearness problem, the algorithm first chooses two searching directions that are effective at leaving the problem region. In the second procedure, the rule for reducing the edge-crossings is taken into account. Thus, the searching direction with fewer possible edge-crossings is chosen. This procedure can be completed by examining the directions of the involved node's connected edges. More precisely, it takes all of the involved node's connected edges and calculates the vectors of these edges from the node to destinations. The angle between each candidate vector and the edge vector is specified, respectively. Possible edge-crossings are those edges that form an acute angle with the candidate vector. In the third procedure, the algorithm searches the candidates of the bypassing *Empty Cells*. It is reasonable to only search *Empty Cells* because the *Empty Cells* are the regions which are not occupied by any node and can be used with less cost. We suggest a search line that is one-fourth the length of the involved edge. The algorithm then checks and removes any candidate which will cause additional edge-ambiguity problem(s) by detecting pairs of potential line segments. The line segments are formed from the center of each candidate and two points, which are the vertices of the involved edge or the nearest existing control point. In the fifth step, if there remains more than one candidate, the best candidate is the one that affects curvature the least according to the readability rule of continuity. Geometrically, the algorithm chooses the candidate with the minimal change to the turning angle.

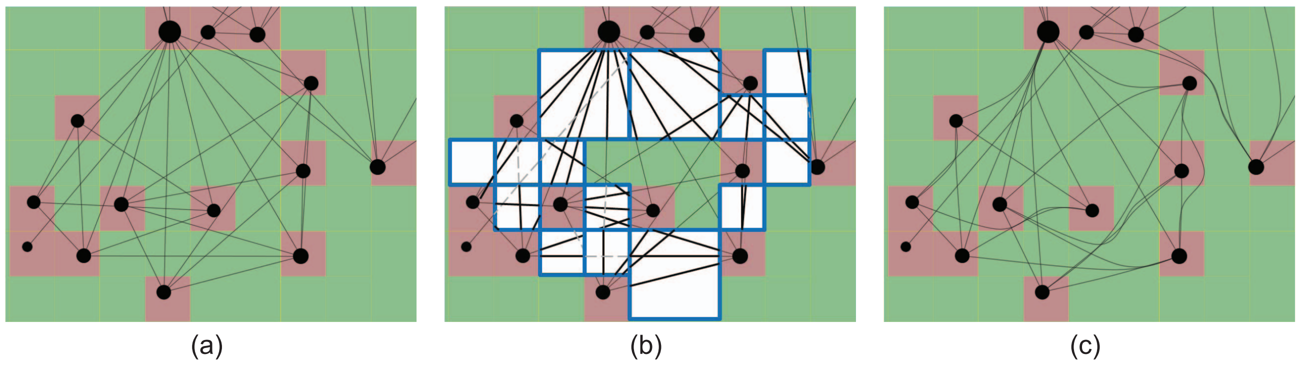

Fig. 8. Determine which edges to be bundled within one *Empty Cell*. (a) Original layout and the quadtree cells. (b) The *Empty Cells* are found that a group of edges sharing a common node passes through. (c) The control points are calculated in these *Empty Cells* and the edges are bundled together.

Note that after rerouting an edge, the rerouted edges are taken as a set of polylines formed by the endpoints and control points. Therefore, the edge-crossing detection of a rerouted edge is performed on these polylines instead of original straight edge. Although the order of rerouting might affects the final result, the visual complexity does not change too much because an edge's direction does not change severely after rerouting.

3.4 Curved-Edge Bundling

In most graph layouts, the largest proportion of visual clutter is caused by edge-congestion. To alleviate this problem, our framework utilizes edge-bundling. By bundling some groups of edges together, less display space is cluttered by the edges, which significantly helps users to perceive individual relations in the detail-view of the graph. The basic idea behind our edge-bundling mechanism, which differs from the traditional approaches, is to bundle the edges that share a common node. In other words, only related edges that connect to the same node can be bundled together.

Traditional edge-bundling methods, however, have one main drawback: it is very difficult to trace individual edges from source to destination out of the entire bundled collection. This makes it problematic to discern which nodes connect to which others without interactive highlighting. To avoid this type of edge-ambiguity problem, in our system, the edges are bundled together if and only if they are geometrically close and also share one common node. This condition guarantees that, even if two edges are geometrically close, they will not be bundled together if they do not share at least one endpoint. If this condition is not enforced, incorrect relationships, like the one shown in Fig. 2(b), may be generated.

A bundle of edges is created when a group of edges shares a common node and passes through the same *Empty Cell*. The bundling is achieved by curving the edges and forcing them to pass through their respective control points. Catmull-Rom splines are used in our implementation due to their local control properties. Fig. 8 illustrates the steps for finding the groups of edges to bundle. It finds the *Empty Cell* that a group of edges sharing a common node passing through, and calaulates the control points for those lines in these cells. If the ambiguity-avoidance step is performed in advance, it creates a set of polylines according to the control points generated in the ambiguity-avoidance step, and uses these polylines instead of original straight-lines to check the bundling criteria. In order to distinguish the edges within a bundle from one another and, thus, retain continuity, the edges only pass through their own respective control points instead of a single point. Although these control points do not coincide, the bundling effect is still achieved because the points are geometrically close in a local spatial region. The equation for calculating the control point for each bundled edge is defined as following:

$$CP = P' + \frac{2U}{U + L_b} * \vec{P'P}, \qquad (2)$$

where CP is the control point for the spline, P is the center of the *Empty Cell* in which the involved edge

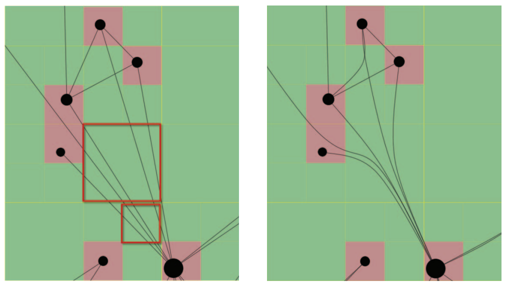

Fig. 9. The strength of edge-bundling varies according to the width of the *Empty Cell*. In a smaller *Empty Cell*, the strength is larger to avoid further overlap or nearness problems. On the contrary, in a larger *Empty Cell*, the strength is relatively smaller to discriminate the bundled edges.

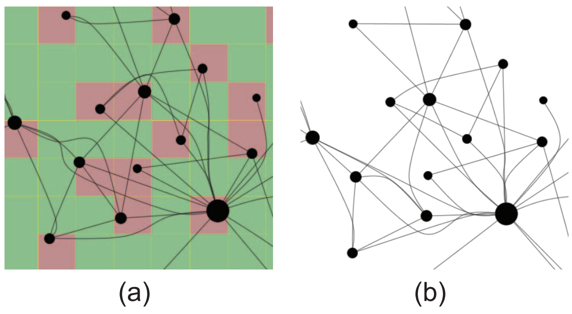

Fig. 10. (a) Before node adjustment, some nodes accidentally locate at the border of their *Node Cells* and this may reduce the effectiveness and niceness of the bundled edges. (b) After adjusting the nodes slightly toward their *Node Cells'* center respectively, this problem is relieved.

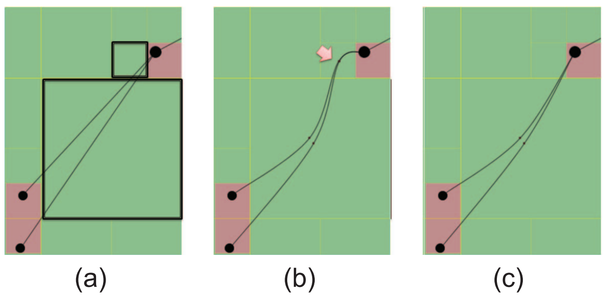

Fig. 11. (a) Two edges are bundled together in the solid-black *Empty Cells*. (b) The curvature of the bundled edges changes dramatically due to the square cells. (c) After removing some unnecessary control points, the problem is alleviated.

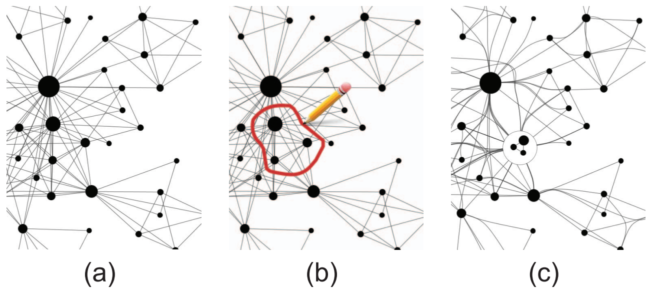

Fig. 12. (a) High-degree nodes gathered in the center cause the edge-congestion and edge-ambiguity problems. (b) The detail-on-demand user interface adopts the lasso selection tool that allows users to create a selection by drawing free-hand. (c) After drawing free-hand with the lasso selection tool, connected nodes in the area are combined together to make the region clean.

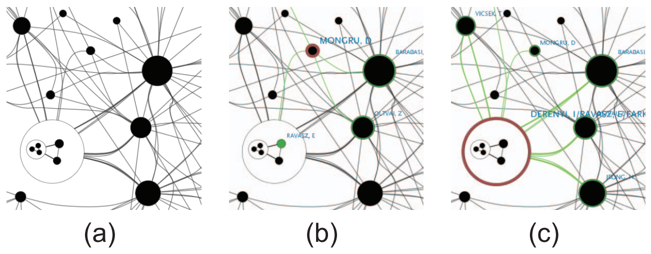

Fig. 13. An interactive way to reveal individual relations of the combined nodes and the outer ones. The detail relationships can be understood and the edge-congestion problem is significantly relieved.

will be bundled, P' is the projection point of P on the involved edge in straight-line style, U is the threshold used in constructing the quadtree, L_b is the width of the *Empty Cell* in which the part of curved-edge will be bundled, and $\vec{P'P}$ is the vector from P' to P. Note that L_b may be either larger than or less than U, and in the latter case CP is located beyond P on the other side of the *Empty Cell*. This was a deliberate design choice so that, if the cell is small, we do not limit the control points to be located in a crowded area. After calculating a control point for a bundled edge, it is added into the edge's list of control points. Then, the list of control points is sorted by their projections on the original straight-line edge, which aligns the control points in the correct sequence for drawing a smooth spline.

The quadtree decomposes the two-dimensional space into adaptable cells, and the width of each cell depends on the distribution of the nodes, which makes the width of each *Empty Cell* variable according to the area of free regions as shown in Fig. 9. If the width of an *Empty Cell* is large, then any edge bundled in this region has a higher degree of freedom, and the sparsely bundled edges are easier to visually separate as a result. On the contrary, if the width of one *Empty Cell* is small, these edges should be tightly bundled in order to prevent further edge-ambiguity problems. The term $U/(U + L_b)$ in Eq. (2) provides this control feature.

3.5 Rendering

In Sec. 3.3 and 3.4, we introduced how to determine the control point(s) for each straight-line edge, so that inappropriate overlapping and nearness problems are avoided by bundling related edges together. To represent the bundled edges, each edge is forced to pass through all of its control points. In some cases, a node may be too close to the boundary of its *Node Cell*, which could lead to proximity ambiguity problems. To alleviate this problem, we slightly adjust those nodes' positions by displacing it to the mid-point of its original position and the center of its *Node Cell*. Fig. 10 shows the nodes' positions before and after the node displacement. Furthermore, the nature of edge-bundling causes an edge-overlap problem. To discriminate bundled edges apart, they are drawn with 50% transparency, which helps to reveal the underlying patterns.

Using square cells for space partitioning simplifies implementation and computation time. However, in some cases, detecting the territory of edges within square

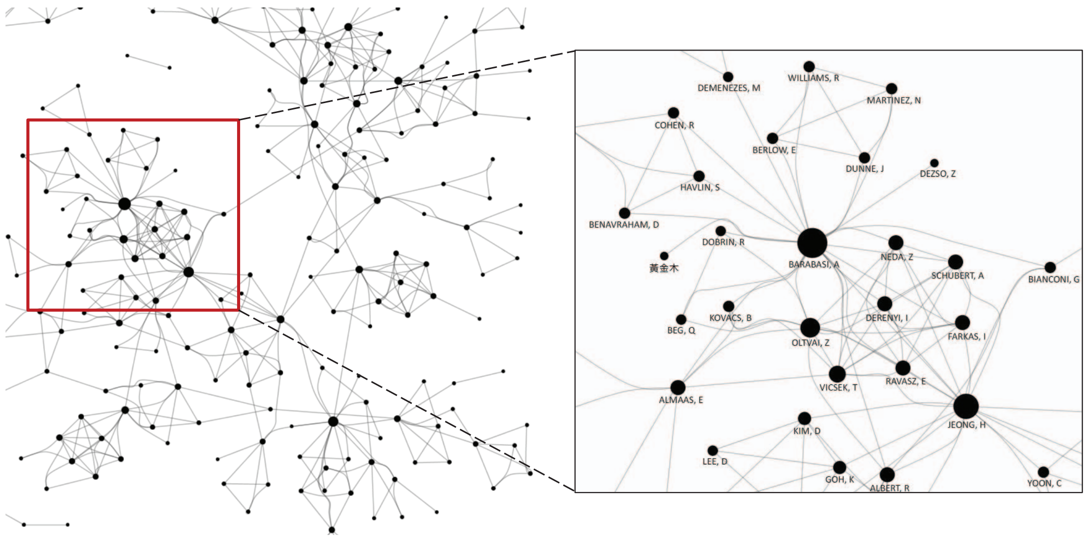

Fig. 14. Our system shows the global view of the public coauthorship data compiled by Newman [33] (left). Users can interactively explore the information in the detail-view (right).

cells causes an aliasing problem, and the bundled edges would tend to over-fit the square cells. For example, as shown in Fig. 11(a), if two edges pass through the same *Empty Cell* EC_i, located on the left-side of their common node, and another *Empty Cell* EC_j, located at the lower-left-side of the common node, these two edges will be bundled together in both *Empty Cells* according to our edge-bundling mechanism. However, the result in Fig. 11(b) shows that our method dramatically changes the curvature of the two edges in such a situation. To avoid this kind of zigzag problem, every edge's control points are examined before drawing. For each control point CP_i, we determine if the previous control point CP_{i-1} and the next control point CP_{i+1} are located in its *8-adjacent* neighboring cells or not. If CP_{i-1} and CP_{i+1} are located in its *8-adjacent* neighborhood, the control point CP_i is removed from the edge's control-point list to avoid the zigzag problem. Fig. 11(c) illustrates the result of this refinement step after it is applied to a set of control points.

3.6 Interaction

In our graph visualization system, an interactive detail-on-demand mechanism is utilized to provide a comprehensible visualization for dealing with the graphs or networks that contain small-world characteristics [7], which often exist in relational datasets such as social networks, bibliography reference data, software structure data, etc. Graphs or networks with small-world characteristics usually have many high-degree nodes gathering in a few local regions, which also cause short edges when compared with random graphs of the same scale. The high-degree clustering makes the graphs or networks incomprehensible. Hence, we introduce a detail-on-demand mechanism that allows users to interactively modify the visual representation of the clustered, short-edged subgraphs without losing their underlying relationships. The users may modify the distribution of settings via an interactive user interface to obtain the desired results as shown in Fig. 12.

During the interaction process, we provide the lasso selection tool that allows a user to create a selection by drawing free-hand. For example, if a user wants a local region like Fig. 12(a) to be more abstract, he or she can simply use the lasso tool to make the region sparser by drawing free-hand as shown in Fig. 12(b). The system then merges connected nodes in the area into a larger meta node, applys a simple graph layout to its children, and converges their external edges, as shown in Fig. 12(c). On the contrary, the user can also make the region denser if local details need to be seen. If a meta node is assigned for release, its node children will be freed to their original subgraphs.

As shown in Fig. 12(c), a circular layout is used to show the subgraph contained within a meta node, and the meta node may also be combined by another higher-level meta node if they meet the combining criterion under the same cell. Fig. 13(a) illustrates the result of a two-layer combination. Furthermore, if the user wants to visualize the details of one child node inside a meta node, he or she can click to highlight it in the simplified subgraph layout as shown in Fig. 13(b) and Fig. 13(c).

4 Results

To demonstrate the effectiveness of our method at relieving the edge-ambiguity and edge-congestion problems in the detail-views, we show the layout with a public coauthorship data compiled by Newman [33]. The data contains a coauthorship network of network theory and experiment field. There are totally 1,589 vertices and

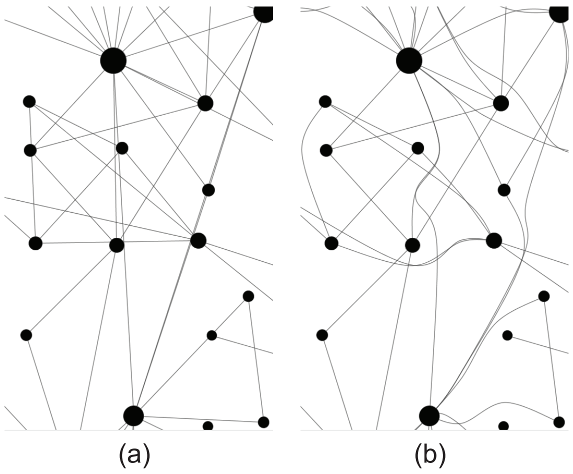

Fig. 15. (a) The graph contains a surprisingly large number (about 10) of edge-ambiguities in a relatively small region. (b) Our result.

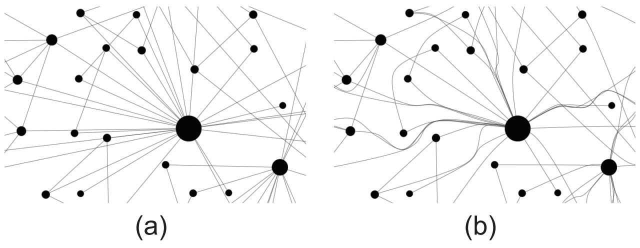

Fig. 16. (a) The node in the center has high-degree of connections. The edges of this node tend to be in similar directions and acute angles, which make these edges hard to be traced and read. (b) After bundling the edges together with less similarity of edge directions, the bundled edges are easier to be discriminated and hence the edge-ambiguity problem is relieved.

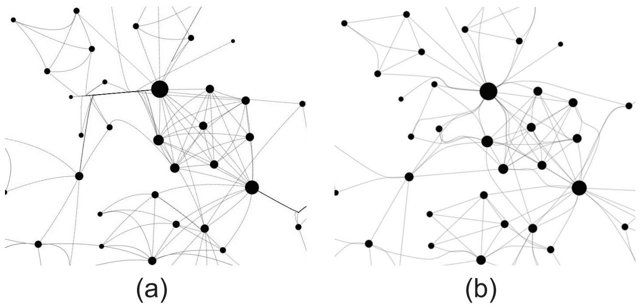

Fig. 17. The result of the coauthorship data using (a) force-directed edge bundling [6] and (b) our method.

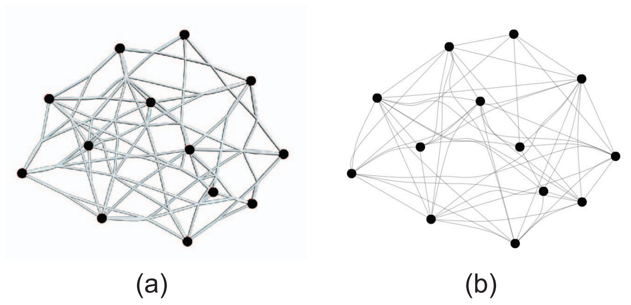

Fig. 18. The comparison of (a) energy-based hierarchical edge clustering [5] and (b) our method.

2,742 edges in the network. We use the data to show how our method deals with the edge-congestion problems formed by completely connected subgraphs and the subgraphs with high-degree nodes in local regions. We also compare our method with previous methods [3] [5] [6]. Finally, we show some snapshots of manipulating the graphs with our interactive user interface. The interface allows us to modify the detail-level of the graphs in order to alleviate the degree of clutter.

Our method scales well with several hundreds of nodes. It can provide realtime interaction on a laptop PC with an Intel Core 2 Duo Mobile T5500 1.66 GHz CPU and 2GB memory. Fig. 14 shows a global view and a local view of a coauthorship data. Since our approach mainly focuses on the accuracy of the topology in the detail-view, we therefore show some more results of the detail-view in the following. Fig. 15 and Fig. 16 show the side-by-side comparisons of the original layouts and our own results in a local area. Compared to the original edge-layout in the straight-line style, the effects of relieving the edge-ambiguity problem are clear. By considering the perception issues related to decreasing the edge-crossings and smooth edge turning, our method is able to enhance the readability of the curved-edge layout. The improved edge-layout helps users to easily understand the correct relationship between individual nodes, and no user interaction is needed to reveal the actual relations on the whole display space. Relational datasets, in practice, often contain high-degree nodes gathering in some local regions, and this phenomenon results in dense cluttered edges and numerous edges pointing to a few common nodes (Fig. 16). The visual clutter caused by these kinds of dense edges can be relieved by bundling them together. As our results show, compared to the straight-edges, the results of bundling edges together frees a significant portion of space, which improves visual discrimination of edges, and the bundled edges tend to decrease the number of edge-crossings, which are common in the straight-line edge-layout.

We compare our result with [6] [5] [3] as shown in Fig. 17, 18, and 19 respectively. From the comparison, the bundling of the force-directed method [6] results in more ambiguity. Some unrelated edges are bundled together and thus users cannot figure out the accurate topology of the network. Although the force-directed edge bundling introduces the compatibility measures to prevent incompatible edges to be bundled, two unrelated but compatible edges are still probably bundled (i.e., two close and parallel edges). Instead, our approach bundles only the edges which share a common node. In addition, the involved edges are curved when am-

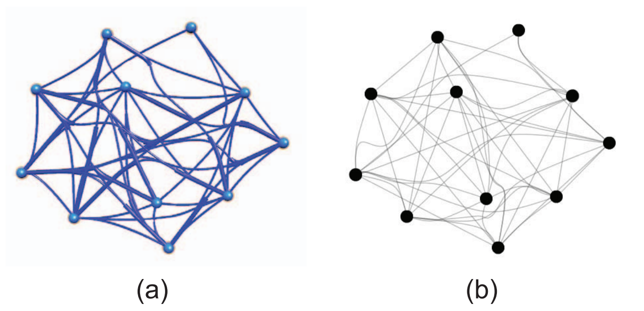

Fig. 19. The comparison of (a) controllable and progressive edge clustering (without control points) [3] and (b) our method.

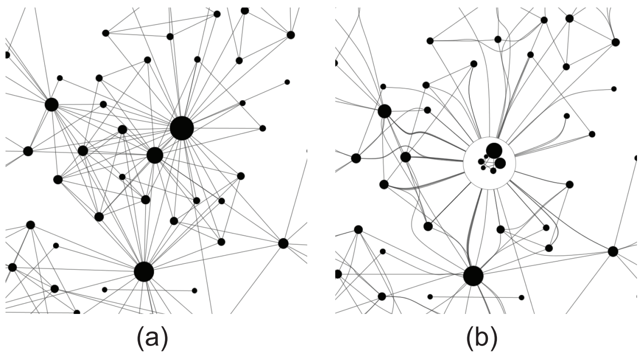

Fig. 20. (a) The subgraph has serious problems of edge-congestion and edge-crossings. (b) The simplified subgraphs generated by our method.

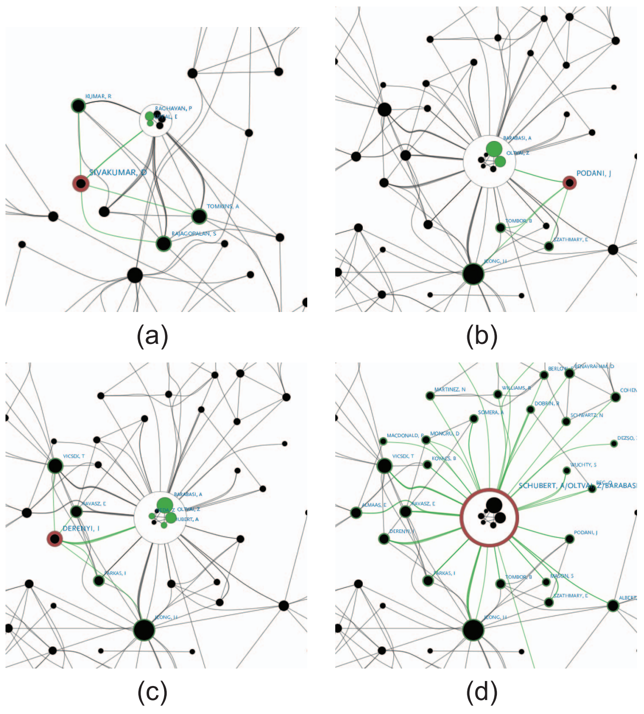

Fig. 21. The interaction for highlighting the connected edges and nodes provides the ability to reveal individual relations between the combined nodes within the meta node and its outer nodes. A user can just move the mouse cursor to the node in which the user is interested, and all connected edges and nodes will be highlighted with light-green color to reveal their individual relations.

biguity happens. Therefore, our approach can provide a more accurate visualization while exploring the network in the detail-view. The energy-based hierarchical edge clustering method [5] introduces the ambiguity problem(Fig. 18) along the edges. Many edges cross at the same points, therefore it's not easy to figure out where these edges go. Our approach can avoid this problem. Fig. 19 shows that the controllable and progressive edge-clustering approach [3] handles the ambiguity problem near nodes in a similar fashion to our method. However, their approach would bundles unrelated edges, and therefore makes the edges ambiguous. On the contrary, our approach prevent such bundles and make the visualization more accurate.

In the regions with densely packed edges, a simplified subgraph layout can be generated by combining the selected nodes based on user preference. As shown in Fig.20, the simplified subgraphs, when compared to the original subgraphs, have a higher degree of edge-convergence, which makes them easier to read and comprehend while still conveying the relations between combined nodes.

Individual relations between the child nodes that are inside the meta node and the nodes outside are lost by the simplification method. Nevertheless, our method provides an interactive mechanism that allows users to highlight the nodes of interest, which in turn highlights the nodes' connected edges and the neighboring nodes. As shown in Fig. 21, the relations between the nodes inside the meta node and the nodes outside are highlighted according to which outside node is in focus. The relations between the nodes outside are also highlighted to ease comprehension.

5 Discussion

5.1 Excessive Edge Bends in Local Regions

In some cases, if the nodes in a small region are highly connected to one another, the subgraphs suffer from numerous overlaps between the nodes and edges, and this problem can cause the relations to be misrepresented and confusing. Although our method can relieve the edge-ambiguity problems for this kind of subgraphs, it may cause frequent edge-bends in a small region as shown in Fig. 22. Frequent, sharp turns can sometimes make tracing edges a difficult task for users.

5.2 Ambiguity Avoidance and Edge Bundling

Edge bundling tries to reduce the visual clutter due to dense edges by merging some edges together, while ambiguity avoidance bypasses an edge if it passes unrelated node(s). Actually, avoiding ambiguous bundling

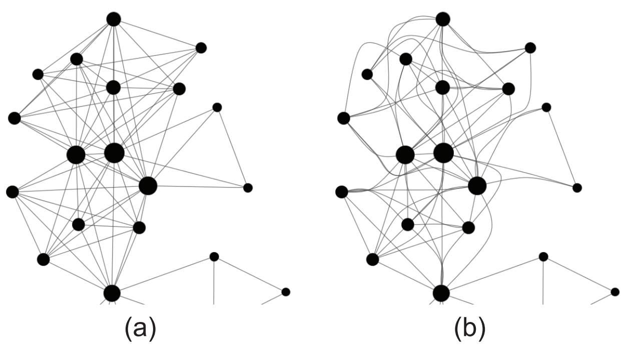

Fig. 22. (a) The nodes in the subgraph are highly connected to themselves in one small region and the edge layout in this region has serious problem of excessive node/edge overlaps gathering in the small region. (b) Our method does relieve the problem of inappropriate overlaps. However, there are too many edge-bends in one small region, and frequent edge-bends make the user need to take more visual efforts to trace with bended edges.

also reduces the maximum bundles. This is an important issue while performing edge bundling in detail-view. When exploring information in the detail-view, users are concerned about the accuracy of the network topology. Therefore, the ambiguity-avoidance step plays the role of preventing the bundling step from being ambiguous. The ambiguity-avoidance step can be turned off if we want to maximize the bundling. Therefore, our algorithm can be generalized to overview by only performing the edge bundling step.

The order of the ambiguity-avoidance step and the edge-bundling step is also important. In our algorithm, the ambiguity-avoidance first creates a set of control points to bypass the edges, and then the edge-bundling step creates another set of control points to bundle the related edges. According to the aesthetic rules of line drawing mentioned before, the number of bends of the curved-edge should be as small as possible. After some experiments, we found that the bends of an edge increase if the edge-bundling step is performed first. The reason is that if an edge becomes longer, they have more opportunities to intersect with *Node Cells*. If the edge-bundling step is performed first, it creates a set of polylines and takes them into the ambiguity-avoidance step. Then, the ambiguity-avoidance step creates more control points because the total length of these polylines is larger than or equal to the length of the straight-lines. As such, we choose to do the ambiguity-avoidance step before the edge-bundling step for the aesthetic guideline.

5.3 Limitations

Our mechanism for avoiding edge-ambiguity is to route any involved edge away from unrelated nodes. However, in some rare cases, if there is no *Empty Cell* in either of the two searching regions for an unrelated node, our method may find no candidate *Empty Cell* for solving the edge-ambiguity problem on this node. In such a case, our method does not force the edge to pass through farther *Empty Cells* since such a bypassing would make the edge become a disturbance in this graph. Instead, we highlight it with a color to draw the user's attention. Another limitation is that compare with flow map layout [1], which visualizes a tree's backbone structure in clear fashion, our approach can't depict a graph's backbone. The reason is that our approach was designed for bundling general graphs, the bundles between two edges are done only at the sources or destinations instead of the entire edges. It is possible to extend our method to generate results similar to theirs by bundling more parts of the edges that share a common source when their destinations are near to each other.

6 Conclusion

Our work addresses the fundamental problem of visualizing locally dense graphs by building on edge-bundling, a novel technique invented to generate graph visualizations that are more visually pleasing and better at conveying structural information when compared to standard methods. In addition to being ambiguity free, our method adds to the edge-bundling layout a level of the control for abstract and detailed views through an interactive user interface.

Graph visualization remains an on-going area of research. Our method is simple and efficient, and can be straightforwardly integrated into existing graph visualization systems. For very large graphs, since our method is based on local operations on the graph, it can co-exist with most large graph visualization solutions.

We propose a few of future work directions as follows: (1) Combine confluent graphs [26] with our bundling technique. With confluent drawing, we can merge crossing edges and draw many graphs in a planar way without causing ambiguity problem. By combining confluent drawing, we can bundle edges not only at the sources and destinations of edges, but also in the middle of edges forming complete bipartite subgraphs. It would enable more bundling and further reduce visual clutter while preventing any ambiguity problem. (2) Adaptive threshold U for space partition. In the cases of failing to find *Empty Cells* for ambiguity avoidance, the regions are always dominated by *Node Cells*. To reduce such cases, we could employ a modification to the threshold U by decreasing its value only in these local regions. The local regions with smaller threshold U would be decomposed more exhaustively, such that more *Empty Cells* would be identified for solving inappropriate overlaps.

Acknowledgments

The authors would like to thank the reviewers for their insightful comments which greatly helped improving the manuscript, and also to Chad Jones for proof-reading.

References

[1] D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd, "Flow map layout," in *IEEE Information Visualization 2005 Conference Proceedings*, 2005, p. 29.

[2] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 741–748, 2006, (Information Visualization 2006 Conference Proceedings).

[3] H. Qu, H. Zhou, and Y. Wu, "Controllable and progressive edge clustering for large networks," in *Proceedings of 2006 International Symposium on Graph Drawing*, 2006, pp. 399–404.

[4] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li, "Geometry-based edge clustering for graph visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1277–1284, 2008, (Information Visualization 2008 Conference Proceedings).

[5] H. Zhou, X. Yuan, W. Cui, H. Qu, and B. Chen, "Energy-based hierarchical edge clustering of graphs," *IEEE Pacific Visualization Symposium 2008*, pp. 55–61, 2008.

[6] D. Holten and J. J. van Wijk, "Force-directed edge bundling for graph visualization," *Computer Graphics Forum*, vol. 28, no. 3, pp. 983–990, 2009, (Proceedings of 2009 Eurographics/IEEE Symposium on Visualization).

[7] S. Milgram, "The small world problem," *Psychology Today*, vol. 1, no. 1, pp. 60–67, 1967.

[8] D. J. Watts, *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Princeton University Press, 1999.

[9] F. van Ham and J. J. van Wijk, "Interactive visualization of small world graphs," in *IEEE Information Visualization 2004 Conference Proceedings*, 2004, pp. 199–206.

[10] F. van Ham, "Using multilevel call matrices in large software projects," in *IEEE Information Visualization 2003 Conference Proceedings*, 2003, pp. 227–232.

[11] P. Eades, Q. Feng, X. Lin, and H. Nagamochi, "Straight-line drawing algorithms for hierarchical graphs and clustered graphs," *Algorithmica*, vol. 44, no. 1, pp. 1–32, 2006.

[12] Q. Feng, "Algorithms for drawing clustered graphs," Ph.D. dissertation, University of Newcastle, 1997.

[13] R. A. Becker, S. G. Eick, and A. R. Wilks, "Visualizing network data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 1, pp. 16–28, 1995.

[14] G. W. Furnas, "Generalized fisheye views," *ACM SIGCHI Bulletin*, vol. 17, no. 4, pp. 16–23, 1986.

[15] K. Misue, P. Eades, W. Lai, and K. Sugiyama, "Layout adjustment and the mental map," *Journal of Visual Languages and Computing*, vol. 6, no. 2, pp. 183–210, 1995.

[16] T. A. Keahey and E. L. Robertson, "Techniques for non-linear magnification transformations," in *IEEE Information Visualization 1996 Conference Proceedings*, 1996, pp. 38–45.

[17] ——, "Nonlinear magnification fields," in *IEEE Information Visualization 1997 Conference Proceedings*, 1997, pp. 51–58.

[18] G. W. Furnas, "The FISHEYE view: a new look at structured files," in *Readings in information visualization: using vision to think*. Morgan Kaufmann Publishers, 1999, pp. 312–330.

[19] D. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, S. Dubs, and M. Roseman, "Navigating hierarchically clustered networks through fisheye and full-zoom methods," *ACM Transactions on Computer-Human Interaction*, vol. 3, no. 2, pp. 162–188, 1996.

[20] M. Sarkar and M. H. Brown, "Graphical fisheye views of graphs," in *ACM CHI 1992 Conference Proceedings*, 1992, pp. 83–91.

[21] Y. K. Leung and M. D. Apperley, "A review and taxonomy of distortion-oriented presentation techniques," *ACM Transactions on Computer-Human Interaction*, vol. 1, no. 2, pp. 126–160, 1994.

[22] N. Wong, S. Carpendale, and S. Greenberg, "Edgelens: An interactive method for managing edge congestion in graphs," in *IEEE Information Visualization 2003 Conference Proceedings*, 2003, pp. 51–58.

[23] N. Wong and S. Carpendale, "Using edge plucking for interactive graph exploration," in *IEEE Information Visualization 2005 Poster Compendium*, 2005.

[24] T. Dwyer, K. Marriott, and M. Wybrow, "Topology preserving constrained graph layout," in *Proceedings of 2008 International Symposium on Graph Drawing*, 2008, pp. 230–241.

[25] M. S. T. Carpendale and X. Rong, "Examining edge congestion," in *ACM CHI 2001 Extended Abstracts*, 2001, pp. 115–116.

[26] M. Dickerson, M. T. Goodrich, and J. Y. Meng, "Confluent drawings: Visualizing non-planar diagrams in a planar way," in *Proceedings of 2003 International Symposium on Graph Drawing*, 2003, pp. 1–12.

[27] P. Eades and R. Tamassia, "Algorithms for drawing graphs: An annotated bibliography," Tech. Rep., 1988.

[28] I. Herman, G. Melançon, and M. Scott Marshall, "Visualiation and navigation in information visualization: A survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, no. 1, pp. 24–43, 2000.

[29] H. Samet, "The quadtree and related hierarchical data structures," *ACM Computing Surveys*, vol. 16, no. 2, pp. 187–260, 1984.

[30] J. Barnes and P. Hut, "A hierarchical O(N log N) force-calculation algorithm," *Nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[31] H. C. Purchase, "Which aesthetic has the greatest effect on human understanding?" in *Proceedings of 1997 International Symposium on Graph Drawing*, 1997, pp. 248–261.

[32] C. Ware, H. Purchase, L. Colpoys, and M. McGill, "Cognitive measurements of graph aesthetics," *Information Visualization*, vol. 1, no. 2, pp. 103–110, 2002.

[33] M. E. J. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Physical Review E*, vol. 74, no. 3, p. 036104, 2006.

Sheng-Jie Luo received the B.S. and M.S. degrees in Information Management from the National Taiwan University, Taipei, in 2007 and 2009, respectively. He is currently a Ph.D. candidate in the Graduate Institute of Networking and Multimedia of the National Taiwan University. His research interests are mainly for image processing, information visualization, and human-computer interaction.

Chun-Liang Liu received the B.S. and M.S. degrees in Information Management from the National Taiwan University, Taipei, in 2007 and 2009, respectively. His research interests are mainly for information visualization.

Bing-Yu Chen received the B.S. and M.S. degrees in Computer Science and Information Engineering from the National Taiwan University, Taipei, in 1995 and 1997, respectively, and received the Ph.D. degree in Information Science from the University of Tokyo, Japan, in 2003. He is currently an associate professor in the Department of Information Management, the Department of Computer Science and Information Engineering, and the Graduate Institute of Networking and Multimedia of the National Taiwan University. His research interests are mainly for computer graphics, image and video processing, and human-computer interaction. He is a member of the ACM, ACM SIGGRAPH, Eurographics, IEEE, IEICE, IICM, and IPPR.

Kwan-Liu Ma received the Ph.D. degree in computer science from the University of Utah in 1993. He is a professor of computer science at the University of California, Davis, and he directs the DOE SciDAC Institute for Ultrascale Visualization. His research interests include visualization, high-performance computing, and user interface design. He received the US National Science Foundation (NSF) PECASE Award in 2000, the Schlumberger Foundation Technical Award in 2001, and the UC Davis College of Engineerings Outstanding Mid-Career Research Faculty Award in 2007. He is the paper chair of the IEEE Visualization 2009 Conference. He also serves on the editorial boards of the IEEE Computer Graphics and Applications and the IEEE Transactions on Visualization and Graphics. He is a senior member of the IEEE.