From Tree to Graph - Experiments with E-Spring Algorithm

Pushpa Kumar[1], Kang Zhang[1], and Mao Lin Huang[2]

[1] University of Texas at Dallas, 2601 N. Floyd Road, Richardson, TX 75063, USA
{pkumar, kzhang}@utdallas.edu
[2] University of Technology, PO Box 123, Broadway NSW 2007, Sydney, Australia
{maolin}@ it.uts.edu.au

Abstract. Graph drawing and visualization represent structural information as diagrams of abstract graphs and networks. *E-Spring* Algorithm, derived from the popular spring embedder model, was proposed to eliminate node overlaps in the drawings of clustered directed acyclic graphs Gc. In this paper, we apply the E-Spring algorithm to general graphs by minimizing edge-node intersections. Initially, a tree structure is extracted from the original graph using the breadth-first search (BFS) algorithm. The extracted tree is then visualized without node overlaps using the E-Spring algorithm, and the remaining non-tree edges are appended to this visualization. A post-processing step that implements edge routing is performed on the obtained visualization to eliminate residual edge-node intersections. This method has been validated by visualizing eBay buyer-seller relationships and Graph Catalog benchmarking data.

Keywords: node overlaps, edge-node intersections, edge routing, graph visualization

1 Introduction

Graph visualization is an area of extensive research in recent years. Applications of graph visualization include genealogy, cartography, sociology, software engineering, VLSI design, and visual analysis. Cluttered drawings of graphs generally produce undesirable visual presentation that could increase the cognitive overhead in understanding of the structural information. Various criteria for determining the graph quality have been proposed [2][16][35]. A graph drawing is sometimes measured by how well it meets the following aesthetic criteria [29][32][37]:

- Minimization of *edge crossings.*
- Minimization of *edge-node intersections:* Edges should not intersect with finite-size labeled nodes.
- *Symmetry*: Nodes should be sufficiently far apart from their nearest neighbor to be easily distinguished and to avoid occlusion.
- *Readability*: Labels must have legible sizes.

- Minimization of *area*: Visual space is often at a premium and reducing the overall area of the bounding box of the graph is often desirable.
- Avoidance of *Overlap*: Labels should not overlap with other labels or other graphical features.

Various algorithms [2][14][15][16][23][29] have been proposed for removing node overlaps in graphs. Being most widely used, Spring algorithms [9] are a class of algorithms for drawing graphs in an aesthetically pleasing way by positioning the nodes of a graph in two or three dimensional space so that all the edges are of more or less equal length, and there are as few crossing edges as possible. They assign forces as if the edges were springs following Hooke's law, the forces are applied to the nodes, pulling them closer together or pushing them further apart. At equilibrium, node overlap is gradually reduced [14][29][36]. Spring algorithms are regarded as effective tools for visualizing undirected graphs.

E-Spring algorithm [20] derived from the popular spring embedder model [9] was proposed to eliminate node overlaps in clustered directed acyclic graphs (DAGs). A combination of spring and electrostatic forces act on the physical system to remove node overlaps at equilibrium. E-Spring algorithm uses animation for drawing clustered DAGs to preserve the user's mental map [11]. Enhancements to this algorithm include determination of a *stop condition* that utilizes graph geometry to determine an upper bound on graph expansion during animated drawing. DAG expansion automatically stops when this condition is met. This minimizes the overall area of the DAG visualization and leads to effective utilization of screen space. Further optimization of this algorithm includes performing *node interleaving* in the top and bottom DAG cluster regions to remove residual node overlaps particularly effective for graphs with long node labels [21]. *E-Spring* algorithm ensures removal of all node overlaps [22].

Graphs have been extensively used to represent various important concepts or objects. A graph is generally drawn or represented in the plane so that we can understand and study its structure and properties [33]. Directed acyclic graphs (DAGs) are an important *subset of graphs*, and have been used to visualize various eBusiness and other applications [17][28]. It is essential that E-Spring algorithm can not only visualize graph subsets but also graphs in general. The usefulness of a graph drawing depends on its *readability* [1]. In this paper, we apply the E-Spring algorithm to visualize graphs without node overlaps and edge-node intersections. The new method consists of the following four steps:

1. One or more trees are extracted from a given input graph using the graph traversal algorithm, *breadth-first search* (BFS) [3].
2. The extracted trees are visualized using the *E-Spring* algorithm, which removes all node overlaps.
3. Non-tree edges (that are not traversed by BFS algorithm) are then appended to the visualization so that the whole graph structure can be visualized.
4. An *edge routing* procedure is performed to remove any residual edge-node intersections.

The proposed method greatly improves the *readability* of node labels in the graph layouts. We evaluate our method by visualizing eBay social network data and Graph

Catalog benchmark data. Using our method, all node overlaps and edge-node intersections are removed in the obtained final layouts.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 presents details of the E-Spring algorithm. Section 4 describes our method for visualizing graphs which includes the clustered graph model, tree extraction, and removing edge-node intersections. Section 5 provides details of our implementation and experimental results. Finally, a conclusion is given in Section 6.

2 Related Work

We review related work on visualization of graphs without node overlap and edge-node intersections, tree search and retrieval algorithms, and edge routing in layouts that served as a motivation for this paper. The family of spring algorithms was developed for adjusting cluttered graphs without node overlaps, and includes force scan (FS) [23], force transfer (FT) [14], dynamic natural length spring (DNLS) and orthogonal dynamic natural length spring (ODNLS) [29]. The Force-scan method (FS) scans in both the horizontal and vertical directions of a graph to find overlapping nodes and uses a force along the line connecting the center of two overlapping nodes in an iterative fashion to remove node overlaps. In Force-transfer method (FT), forces are applied with left-to-right and top-to-bottom scans, and some nodes are moved either horizontally or vertically. Outputs using FS method have smaller area than those of FT. Dynamic natural length spring (DNLS) and orthogonal dynamic natural length spring (ODNLS) modify attractive and/or repulsive spring forces between nodes to achieve node overlap removal. Both DNLS and ODNLS have a better performance than FS and FT in terms of aesthetic criteria such as display symmetry and uniform edge. A fast algorithm (FADE) for two dimensional drawing, geometric clustering and multilevel viewing of large undirected graphs is presented in [36]. The decomposition tree provides a systematic way to determine the degree of closeness between nodes without explicitly calculating the distance between each node. For large clustered undirected graphs, three types of spring forces - internal spring, external spring and virtual spring that operate between the vertices of a graph are defined for the navigation and visualization of these graphs [10]. In addition to spring forces, there are gravitational repulsion forces between all nodes.

A general framework called "User Hints" treats the map labeling processes as an optimization task and supports human interaction along with automated methods [32]. *Simulated annealing* (SA) is a flexible optimization method and solves the problem of drawing nice-looking undirected straight-line graphs without node overlaps. The SA method uses rules that are derived from an analogy to the process in which liquids are cooled to a crystalline form, a process called annealing [5]. A three-dimensional system based on the simulated annealing algorithm extends straight-line two-dimensional drawings of general undirected graphs to three dimensions [4]. This system features an advanced 3D user interface that assists the user in choosing and modifying the cost function and the optimization components on-line. Various approaches for removing node overlapping based on constr*ained optimization* techniques are proposed in [31]. These include minimal linear scaling, formulating the node overlapping problem as a convex quadratic programming problem to be solved by any quadratic solver, local search methods based on adaptation of the EGENET solver, and form of Lagrangian multiplier method.

Graph theory algorithms such as Breadth-first search (BFS) and Depth-first search (DFS) efficiently *search* a given unweighted graph beginning with any source vertex [3]. They can used to determine whether a graph is connected and what components are connected. While BFS searches for discovered and undiscovered vertices across the breadth of the graph, DFS performs search along the graph depth. Two minimum-spanning tree (MST) determination algorithms for weighted graphs are Kruskal [18] and Prim [34]. Kruskal's algorithm is based directly on the generic minimum-spanning-tree algorithm to find a safe edge to the growing forest by finding and edge of the least weight. Prim's algorithm is a special case of the generic minimum-spanning-tree algorithm and has the property that the edges in a set always form a single tree. Dijkstra's algorithm [6] is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree and is often used in routing. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex.

An approach to remove *edge-node intersections* for graphs with non-trivial node sizes using the force-scan (FS) algorithm has been proposed in [24]. This method integrates the force-scan algorithm and applies it successfully to prevent edge-node intersections and overlapping edges. A new method for the analysis of the inter-relation of clusters and levels, and their influence on edge crossings and cluster/edge crossings has been proposed [12]. It applies two-level crossing reduction algorithms for clustered level graphs and retains the optimality of a one-sided two-level crossing reduction algorithm. To solve the problem of edge-node intersections, routing algorithms that originate in robot motion planning [25][26][30] have been proposed. They are designed for finding a shortest path between two specified points while avoiding intersections. Applications such as Microsoft Visio [41] and Concept Draw [38] provide object-avoiding connector routing. The routes are updated only after object movement has been completed rather than as the action is happening. Another well-known library for edge routing in graph layout is the Spline-o-matic library developed for GraphViz [40]. This supports poly-line and Bezier curve edge routing. An algorithm that draws only the edges with fixed node positions and does not make any use of the layered structure is provided in [7]. It starts by drawing the edge as a direct line between the source node and the destination node and then re-routes the line away from the nodes that intersect with the line. Another method for integration of edge-routing techniques into a force-directed layout is based on constrained stress majorisation [8]. It takes an initial layout for the graph, including poly-line paths for the edges, and improves this layout by moving the nodes to reduce stress and moving edge bend points to straighten the edges and reduce their overall length. Separation constraints between nodes and edge bend points are used to ensure that node labels do not overlap edges.

3 E-Spring Algorithm

E-Spring algorithm [20] was proposed to eliminate node overlaps in the drawing D(Gc) of clustered directed acyclic graphs (Gc), especially for labeled nodes with finite length string labels. In this algorithm, nodes are modeled as unequally charged particles, with springs representing edges of the graph. The spring force is derived from the following force formula according to Hooke's law:

$$F_s(d) = C_1 \log(d/C_2) \quad \textbf{(1)}$$

where d is the spring length and C_1 and C_2 are constants. The new electrostatic force generated between any two charged particles representing two connected nodes is derived from Coulomb's law as follows:

$$F_c = k_c q_1 q_2 / r^2 \quad \textbf{(2)}$$

where q_1 is the charge on one body, q_2 is the charge on the other body, r is the distance between them, and k_c is the Coulomb's force constant. The combination of F_s and F_c is the resultant force between two connected nodes. Note that for each pair of non-connected nodes, the combined force is equal to F_c.

The magnitude of charges is obtained by calculating the weight associated with each source node. The force function calculated from the weights modeled on electrostatic charges eliminates node overlap. The weight of each source node is determined by the number of sink nodes in its immediate vicinity, number of hierarchy levels in the tree, and a force scaling factor. A classification mechanism identifies nodes as parents or children. A parent node along with all its direct child leaf nodes constitutes a *cluster* c_i. A new force function based on the weights is applied and the additional displacement contributes to node overlap removal. The forces are applied to each node and the positions are updated accordingly. The attractive forces on the springs and the repulsive forces between the positive charges act together to generate a drawing free of node overlaps. A final layout is obtained with a locally minimum energy on the nodes when the total force on each node in |V| is zero. Enhancements to this algorithm includes introduction of a '*stop condition'* to reduce the overall size of DAGs leading to more efficient utilization of screen space, and '*node-interleaving*' to remove any residual node overlaps [21]. *E-Spring* algorithm ensures that all node overlaps in a DAG are removed [22].

E-Spring Algorithm

```
  input: a drawing $D(G_D)$ of clustered $G_c(V,E)$ with overlapping nodes
  output : a new drawing $D^{new}(G_c)$ with no node overlaps
  begin
    mark parent nodes $p_k$
    compute force factor f
        for each subtree i = 0 to k do
            compute weight $W_i$
            calculate new forces $F_{newi}$
    move nodes based on minimum distance threshold $\delta$ and compute final layout
    until stopping condition
end
```

4 Graph Visualization Methodology

To visualize a given graph using the E-Spring layout algorithm, we first extract the underlying tree structure from the graph. For this purpose, we utilize Breadth-First search (BFS) algorithm, because the derived tree is a 'fat' tree [27] that can used for generating tree clusters. A fat tree becomes fatter as one moves up the tree towards the root, so a parent node has more children in its vicinity to form clusters. Section 4.1 presents the clustered graph model. Section 4.2 describes the tree extraction method. An edge routing method that removes unnecessary edge-node intersections as the 4th step of our approach is described in Section 4.3.

4.1 Clustered Graph Model

Our clustered graph model is built on its own backbone tree *T* derived from the existing graph structure. This backbone tree *T* is a spanning tree which is extracted through the implementation of a Breadth-First-Search (BFS) algorithm.

Definition 1: Consider a graph $G(V, E)$ with a set of nodes $V = \{ v1,v2,...,vn\}$, and the set of edges $E \subseteq V x V$. A spanning tree $T(V_t, E_t, r)$ obtained through the *BFS* on *G* contains edges *in* E_t. A set of unreachable edges that are not included in *T* is contained in $E' = E - E_t$.

Definition 2: Consider a graph $G(V, E)$ with a set of nodes $V = \{v1,v2,...,vn\}$, and the set of edges $E \subseteq V x V$. The *BFS* algorithm expands node exploration uniformly across the breadth of the graph; a parent node v^p *in* V_t of the spanning tree *T* has a set of direct child leaf nodes $L_r \subset Vi$ in its immediate vicinity. A parent node together with all its direct child leaf nodes forms a *cluster* $c_i = \{v^p, L_i\}$, where v^p is a parent node of all children in L_i.

Definition 3: A clustered graph $G_C(C, V', E')$ consists of a set of clusters $C = \{c_1, c_2, ...c_k\}$ that are derived from the spanning tree *T* extracted from *G* as defined above, a set of non-tree nodes *V'* and a set of non-tree edges *E'*.

4.2 Tree Extraction

E-Spring algorithm [20][21][22] works well with graphs that follow a clustered tree structure, where a parent node has many child nodes in its vicinity to form clusters, and nodes have finite length string labels. For the purpose of using E-Spring algorithm for graph visualization, the method we choose for tree extraction has to produce fat trees [27]. A fat tree becomes fatter as one moves up the tree towards the root, so a parent node can have more direct child nodes. We therefore choose the graph search algorithm breadth-first search (BFS) that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes and so on, until it finds the shortest path and generates a minimum spanning tree 'T' starting from the root node. Depth-first search (DFS) is another algorithm for graph traversal that starts at the root and explores as far as possible along each branch before backtracking. But this method does not produce fat trees.

Extraction of the tree is performed using the BFS method. Breadth-First search (BFS) is a graph traversal algorithm that can be used to find connected components in a graph. Given a graph $G = (V, E)$ and a distinguished source vertex '*s*', BFS explores the edges of G to "discover" every vertex that is reachable from '*s*' by computing the distance (smallest number of edges) from '*s*' to each reachable vertex [3]. The algorithm is given below:

BFS(*G*,s)

```
unmark all vertices in $G$
choose some starting vertex '$s$' as root node
mark '$s$'
Enqueue($Q,s$)
$T \leftarrow s$
while $Q$ non-empty
    choose some vertex '$v$' from beginning of queue
    visit '$v$'
    for each unmarked neighbor '$w$'
        mark '$w$'
        Enqueue($Q,w$),
        $T \leftarrow vw$
```

BFS generates a spanning tree T with root r that contains all reachable vertices. The breadth first search tree T is the shortest path tree starting from its root r.

Using E-Spring layout algorithm, the extracted spanning tree T can be visualized with the elimination of node overlaps. The missing non-tree nodes in V' and edges in E' are then be appended to the visualization. Each edge is drawn as a straight line connecting the centers of the two nodes it links. The resultant visualization thus contains all nodes and edges from the initial graph G. To improvise the obtained layouts, we aim to remove residual edge-node intersections.

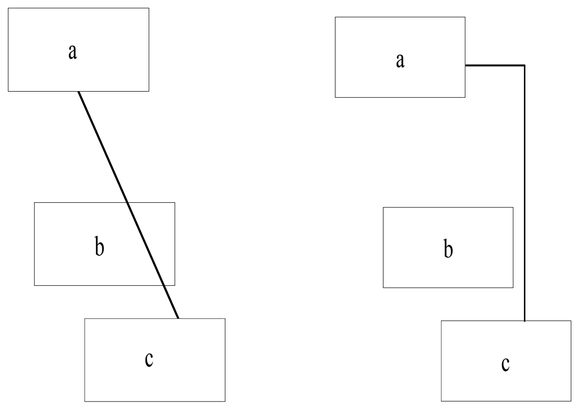
4.3 Removing Edge-Node Intersections

Typically, the drawing of a graph is accompanied by optimizing some cost function such as area, number of bends, number of edge crossings, uniformity in the placement of vertices, minimum angle etc. [33]. Computation of an aesthetic cost of a drawing involves node overlaps, and edge-node intersections [1]. We aim to minimize this cost by removing both node overlaps and edge-node intersections. The visualizations obtained after applying the E-Spring algorithm do not contain node overlaps, but suffer from intersections between edges and nodes, which greatly hinder *readability* of node labels. To remove these edge-node intersections, we apply edge routing on these layouts as a post-processing step. For edge routes in a graph layout, several relevant readability [24] criteria are:

- Minimizing the number of edge bends
- Minimizing the total length of edges
- Minimizing the number of edge crossings

Our clustered graph model is built on its own backbone tree *T* derived from the existing graph structure. This backbone tree *T* is a spanning tree which is extracted through the implementation of a Breadth-First-Search (BFS) algorithm.

'Orthogonal edge routing' was implemented for removing edge-node intersections in graphs with non-uniform nodes of finite length [24]. Since *E-Spring* algorithm works well with node labels of finite string label length, we aim to implement orthogonal edge routing for the obtained visualizations from Section 4.1. In this type of edge routing, a diagram's edges are routed using vertical and horizontal line segments only. An example of orthogonal edge routing is shown in Fig. 1a, b. Fig. 1a shows the actual edge-node intersection between edge e(*a,c*) and node 'b' before re-routing. If an algorithm for orthogonal routing drawings is chosen, the output is shown in Fig. 1b with removal of the intersection.



(a) Before Rerouting (b) After Orthogonal Routing

Fig. 1. Orthogonal Edge Routing.

For performing the post-processing step, we utilized '*yFiles*' Graph Layout Library available from yWorks [42]. Automatic edge routing is provided by 'O*rthogonalEdgeRouter*' which is a versatile and powerful layout algorithm. This tool also reduces the total number of edge crossings. We captured individual node, edge information such as node graphical position co-ordinates (*X,Y*) and edge connections from the obtained visualization data structure. Since 'yFiles' Library has a 'graphml' interface, we inserted the retrieved node, edge information for the visualization into a graphml file. This file was then loaded into the orthogonal edge routing application to remove edge-node intersections.

5 Implementation/Experimental Results

We have implemented the graph visualization method using *E-Spring* algorithm presented in Section 4 for eBay data and graph benchmark data. In this implementation, graph $G = (V, E)$ has the following parameters:

- Total number of nodes $1 \leq |N| \leq 100$
- Individual node label string length $3 \leq S \leq 10$

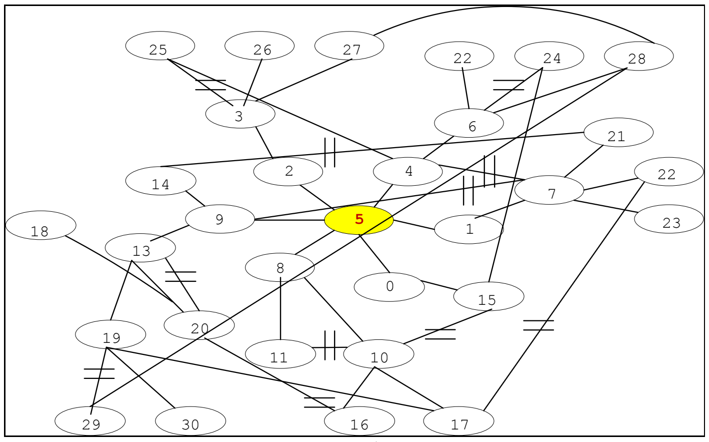
Input data for this implementation came from the following sources:

- eBay buyer/seller relationships obtained from eBay [19], where user-name string length '*S*' is chosen from the following two cases: i) S = 3 ii) S = 10.
- Graph catalog benchmark data (AT & T graphs) obtained from the Graph drawing Web site [39] for the following three cases: i) |N| = 60, S = 10, r_n = 9 ii) |N| = 60, S = 10, r_n = 39 iii) |N| = 100, S = 10.

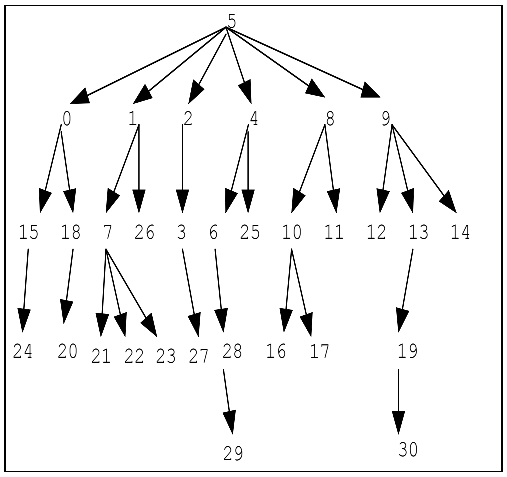
The Breadth-First search (BFS) algorithm presented in Section 4.2 was implemented using the Java programming language. The 4th step for edge routing presented in Section 4.3 was implemented using the 'yFiles' Graph Layout Library from yWorks [42]. The implementation results for the above cases are presented in this section.

5.1 eBay Data

We plotted the social network graph G(*V, E*) extracted from eBay [19]. eBay customer data for various buyers and sellers was mined from its production environment using the Web services API. The buyer-seller interaction was captured by the *social network graph*. The graph has total number of nodes |N| = 30, number of edges |E| = 41. This graph is depicted in Fig. 2a. The java code of the BFS algorithm was implemented on this graph with chosen root node r_n = 5 (highlighted in yellow), and the corresponding tree 'T' extracted is depicted in Fig. 2b.



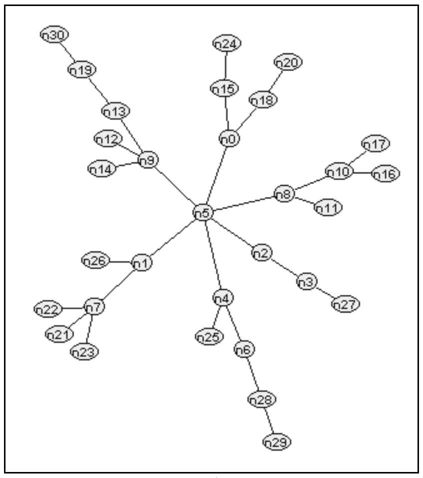
a)



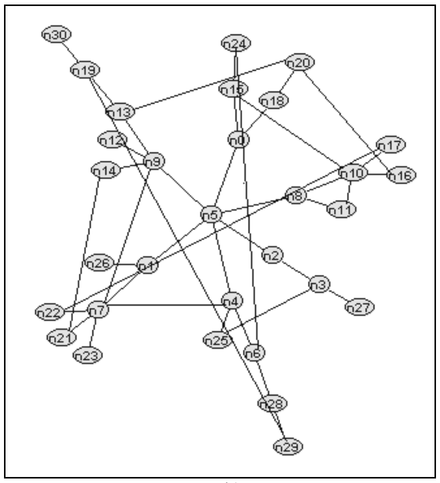
b)

Fig. 2. eBay Data Graph and extracted tree structure.

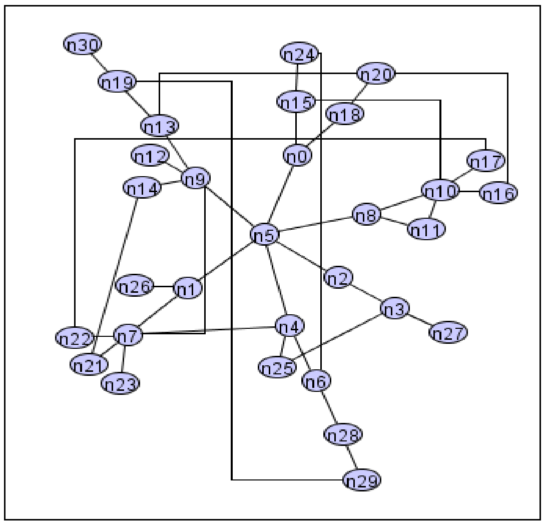
Fig. 3a, b depict screenshots of the tree visualization obtained using *E-Spring* algorithm, for *Case 1*: S = 3. Fig. 4a, b depict screenshots of the same for *Case 2*: S = 10. Fig. 3a, 4a represent extracted tree *T* while Fig. 3b, 4b represent screenshots after non-tree tree edges contained in *E'* were appended to the visualizations in Fig. 3a, 4a. Fig. 3c, 4c depict screenshots after 4th step using orthogonal edge routing method was applied to remove the edge-node intersections.



a)

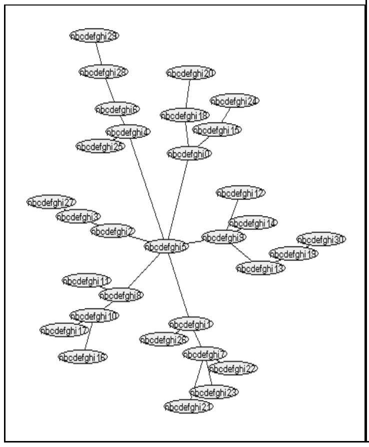


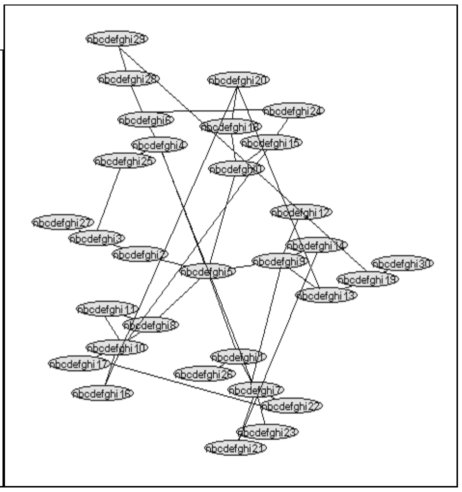
b)



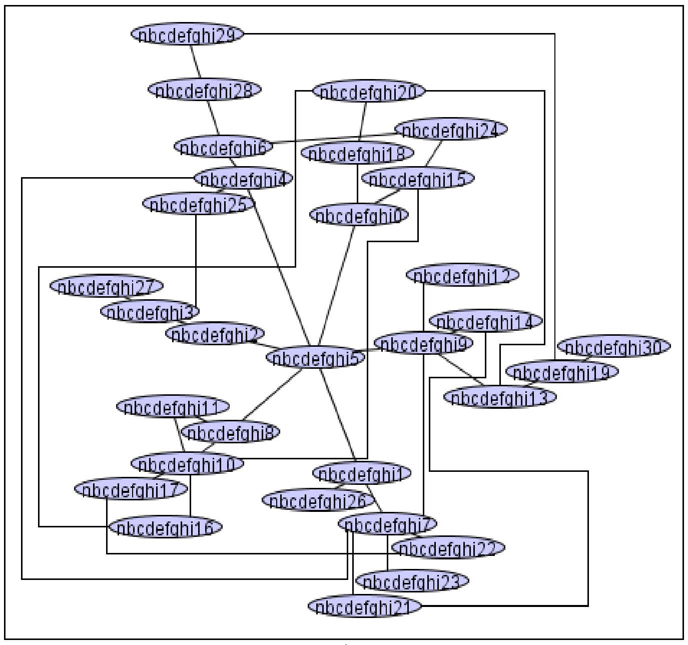
c)

Fig. 3. eBay data Case ‘1’ |N| = 30, |E| = 41, S = 3.





a) b)



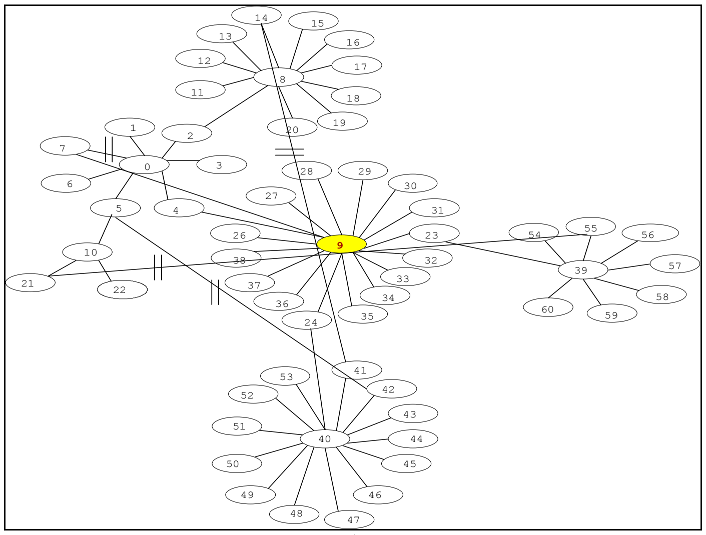
c)

Fig. 4. eBay data Case '2' |N| = 30, |E| = 41, S = 10.

5.2 Graph Catalog Data (AT & T Graphs)

To validate our method, we implemented graph benchmark data using bigger graphs containing higher number of nodes. The data files i) "*graph ug_97*" with |N| = 60 and ii) "*graph ug_379*" with |N| = 100 were downloaded from "*ug.gz*" zip file available on the graph drawing Web site [39]. Since *E-Spring* algorithm removes node overlaps especially for nodes of longer string label length, we implemented the downloaded AT & T graph structures for string label length S = 10.

BFS method extracts a spanning tree starting with a root node. The choice of this node can possibly generate different trees and visualization layouts. We consider two different root nodes based on the node degree. Fig. 5a, b shows the graph and extracted tree *T* respectively for Case 3 *i)* with selected root node r_n = 9. This node has the highest node degree N_D = 16. Fig. 6 represents tree *T* for Case 3 *ii)* with selected root node '39'. This node has the lowest node degree of N_D = 7. Fig. 7a, b, c depict screenshots of the visualization obtained using *E-Spring* algorithm, for *Case 3i)*: |N| = 60, r_n = 9. Fig. 8a, b, c depict visualization screenshots for *Case 3ii)*: |N| = 60, r_n =39. Fig. 9a, b, c depict visualization screenshots for *Case 4*: |N| = 100. Fig. 7b, 8b, 9b are screenshots with non-tree tree edges contained in *E'* appended to the visualization in Fig. 7a, 8a, 9a. Fig. 7c, 8c, 9c depict screenshots after 4th step using orthogonal edge routing was applied to remove edge-node intersections.


a)

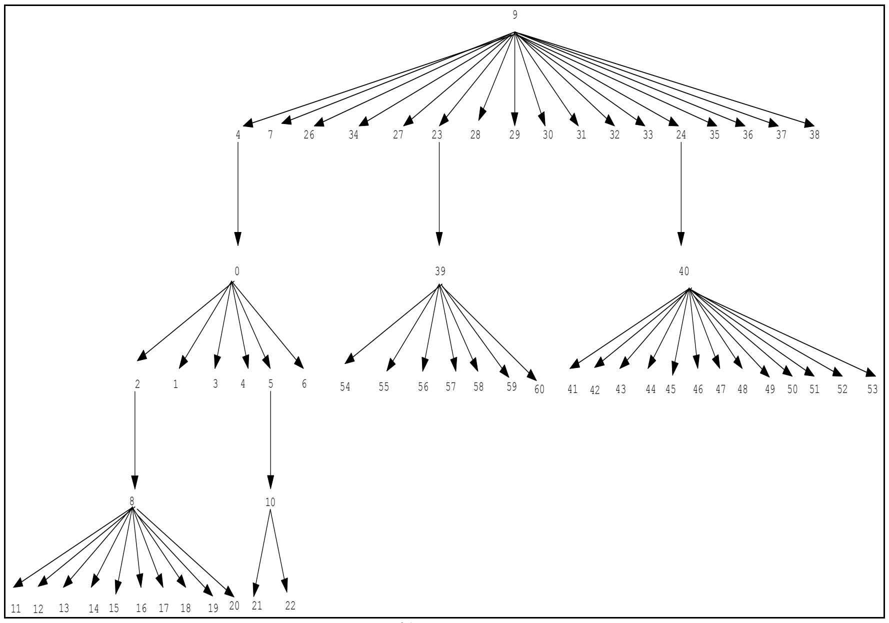

b)

Fig. 5. Catalog data graph and extracted tree structure for Case ‘3 i)’.

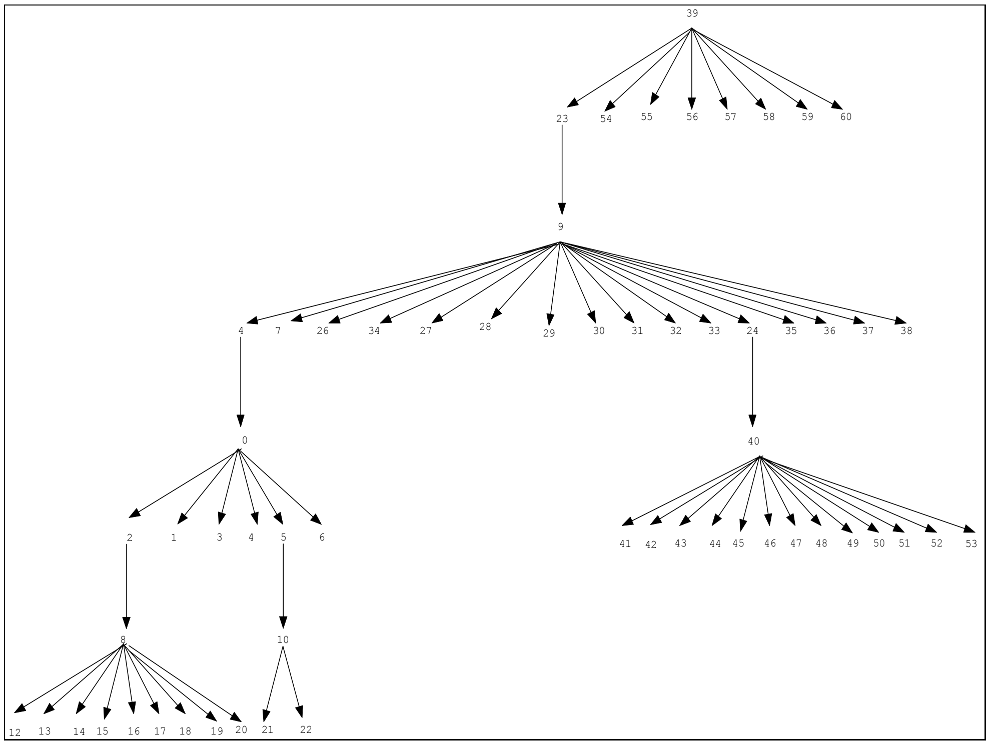
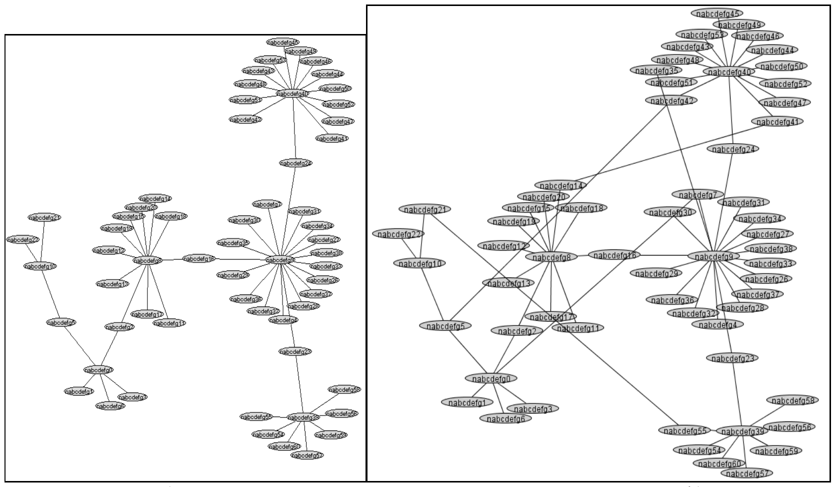
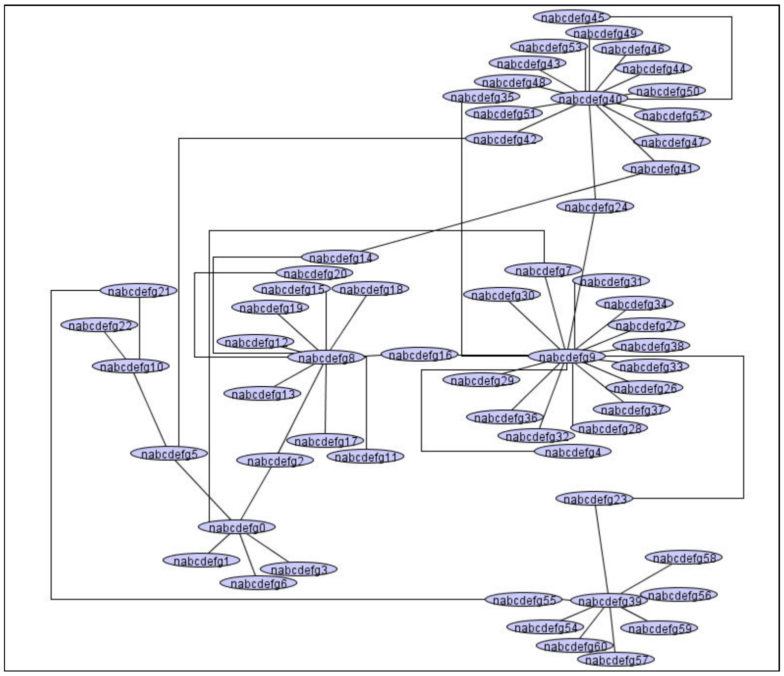


Fig. 6. Extracted tree structure for catalog data graph Case '3 ii)'.

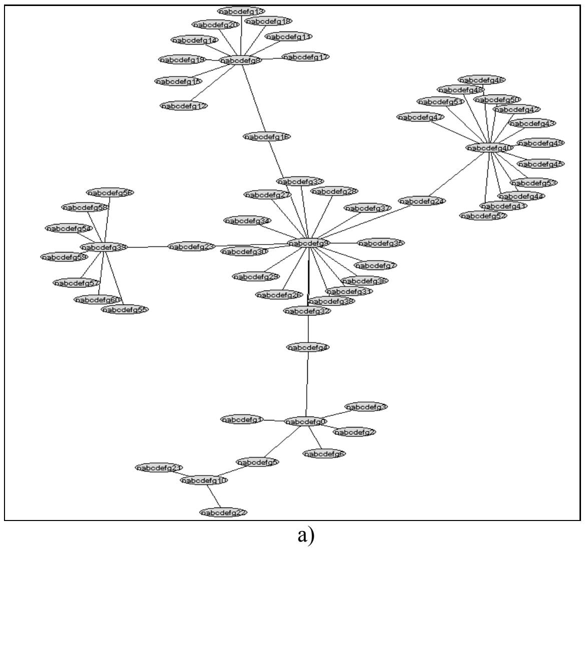


a) b)

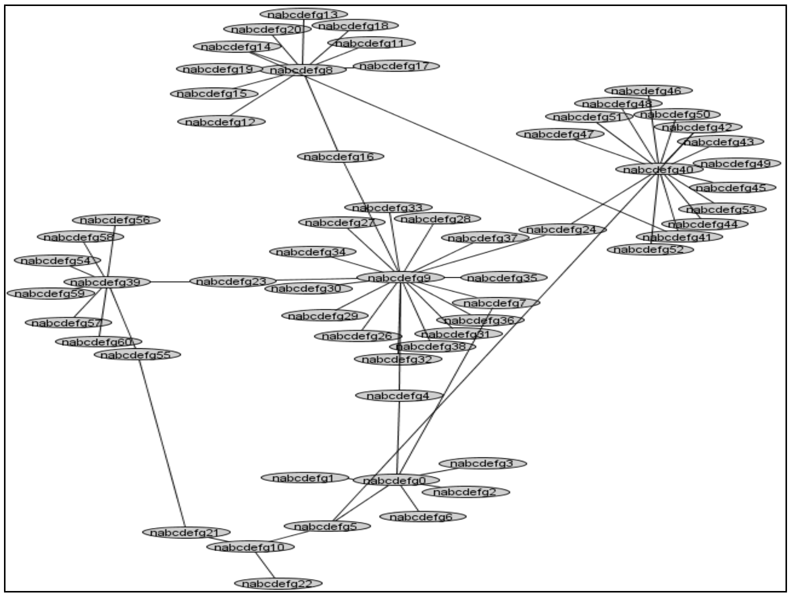


c)

Fig. 7. Catalog data graph Case '3 i)' |N| = 60, |E| = 59, S = 10, Root node (r_n) = 9.



a)


b)

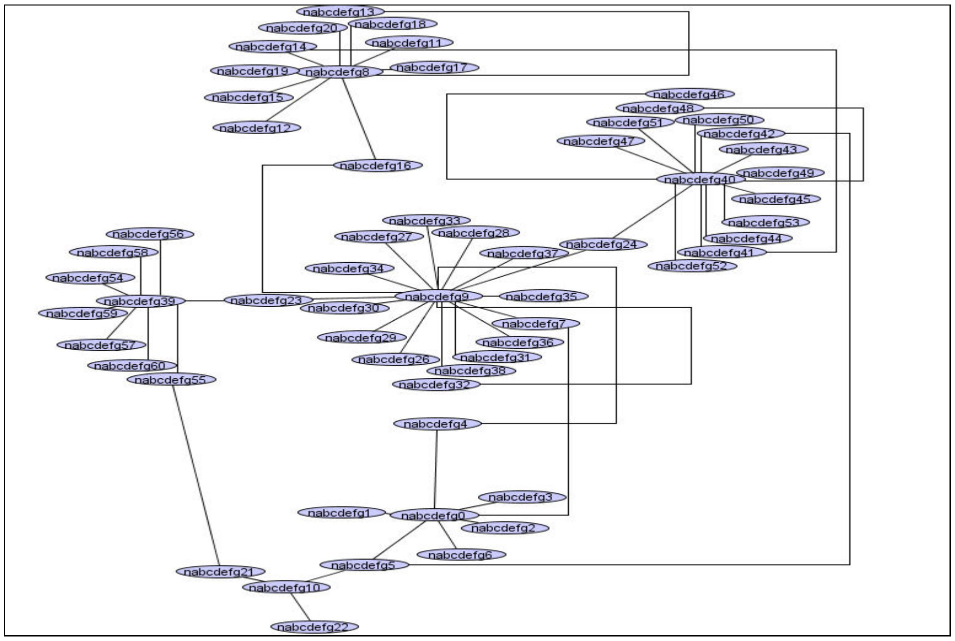
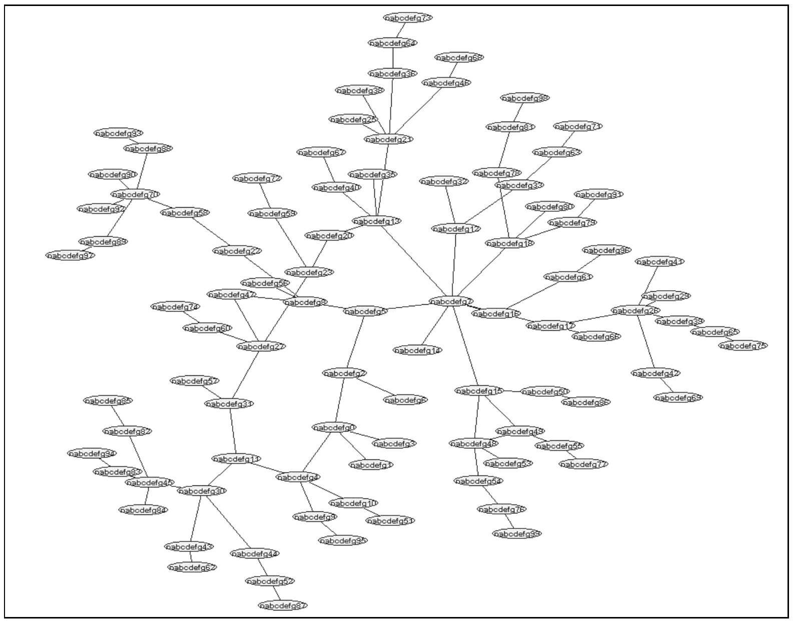
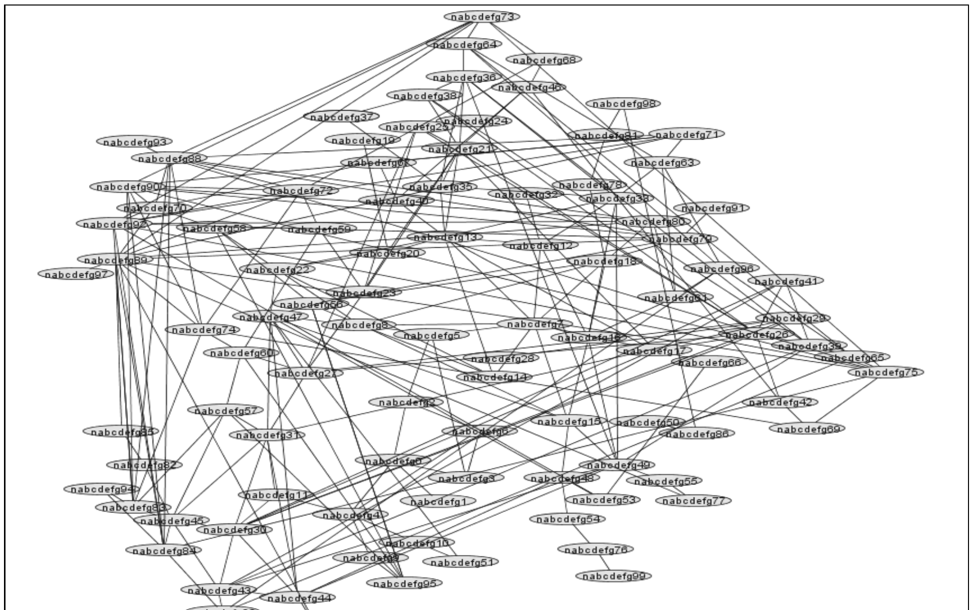
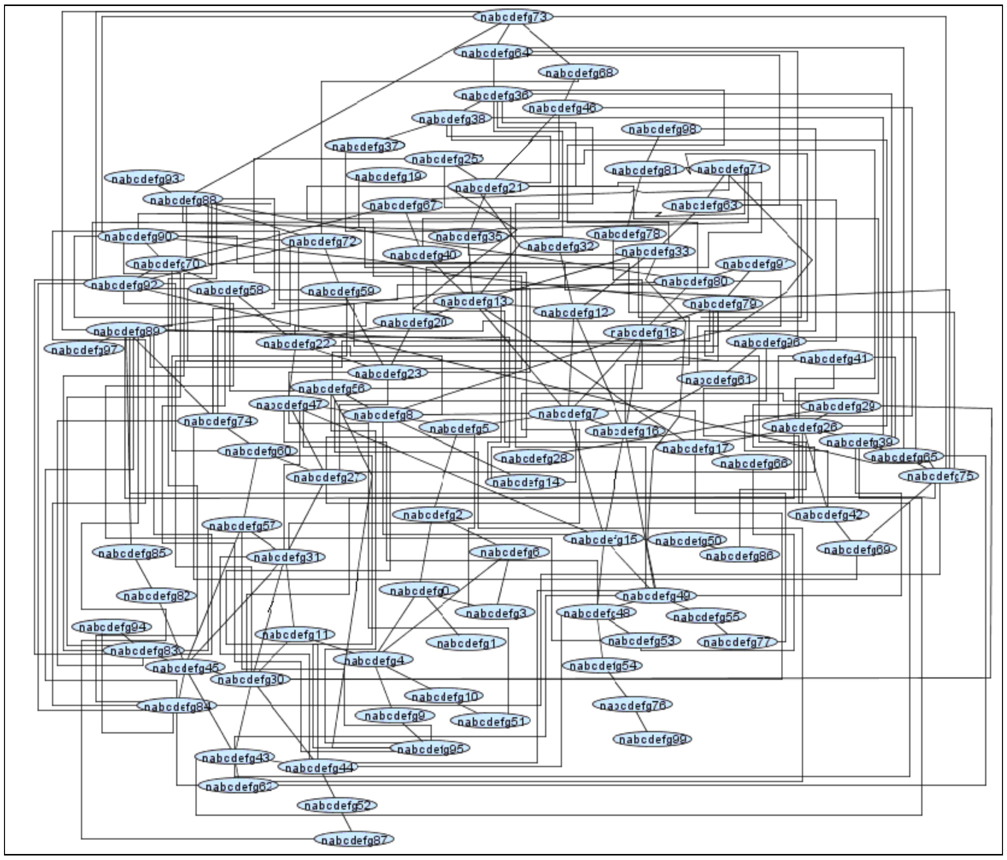

c)

Fig. 8. Catalog data graph Case '3 ii)' |N| = 60, |E| = 59, S = 10, Root node (r_n) = 39.


a)


b)



c)

Fig. 9. Catalog data graph 'Case '4' |N| = 100, |E| = 250, S = 10.

5.3 Observations

For each of the above implemented cases, we retrieved information about node traversal in the graph for *BFS* algorithm, extracted tree *T*, a set of non-tree edges *E'*, depth of the tree 'D_T', and number of edge-node intersections in the drawing before the 4th step. This data is depicted in Table 1. We also measured overall bounding areas of graph visualizations before and after post-processing. Area measurements (pixels^2) are depicted in Table 2. For our plots and measurements, we use terminology "*Implementation Data source*" 1, 2, 3, 4, 5 respectively for the five implemented cases a) eBay data Case 1, b) eBay data Case 2, c) Graph Catalog benchmark data Case 3 i), d) Graph Catalog benchmark data Case 3 ii), and e) Graph Catalog benchmark data Case 4. We have observed that as the graph size increases, the extracted tree size, the number of non-tree edges, the depth of the tree and the number of edge-node intersections also increase proportionately for all the five data sources. This is due to the presence of larger number of nodes and edges in the input graphs. These measurements are depicted as plotted curves in Fig. 10.

We observed that the choice of the source root node (r_n) affects the tree depth, number of edge-node intersections, and the overall graph area. When source node with the highest node degree is chosen (r_n = 9) in Case *3 i)*, the extracted tree depth is lower (D_T = 5) so more children are present for every parent node in each cluster. The corresponding tree structure has higher tree depth (D_T = 7) when source node with lowest node degree is chosen (r_n = 39) in Case *3 ii)*. After missing edges are appended, the obtained visualizations indicate the presence of higher number of edge-node intersections (18) for Case *3 ii)* with lowest degree source root node (r_n = 39). The higher depth of the tree also causes the overall graph area to increase by 20% for Case *3 ii)* for Graph Catalog data with lowest degree source root node (r_n = 39). Hence, the selection of the *highest degree* node as the starting node for breadth-first search (BFS) algorithm produces better visualizations for our method.

After the 4th step of our approach, we observed that the obtained visualizations have greater area for nodes with longer string labels ($S = 10$) and larger graphs ($|N| = 60$, $|N| = 100$). This is because smaller label size nodes contain more in-between node gaps for performing orthogonal edge routing internally as compared to longer label size nodes. For nodes with longer string label lengths, the edge route sometimes has to be re-routed externally causing an increase in the graph area. Area measurement curves are plotted in Fig. 11. A trade-off exists between removal of edge-node intersections and graph area reduction with our method, especially for larger graphs with greater label sized nodes. From the obtained layout visualizations we observed that no node overlaps occurred in spanning trees drawn by *E-Spring* algorithm. The 4th step preserves the mental map, since the node positions were captured from the obtained visualization. We also observe that after the 4th step, all edge-node intersections in graph layouts have been removed.

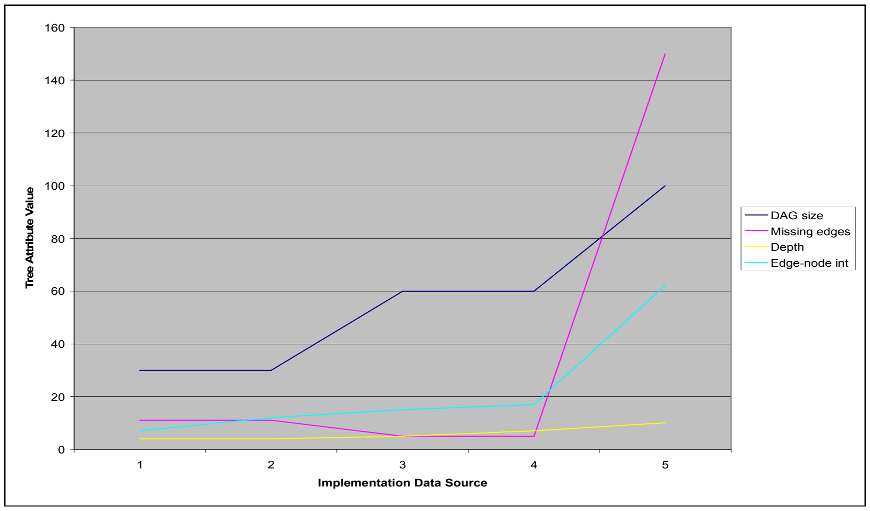


Fig. 10. Attribute values of extracted tree.

Table 1. Data Attributes of Extracted Tree Structure.

Data Source	Extracted tree size	Non-tree edges	Tree Depth	Edge-node intersections
1	30	11	4	7
2	30	11	4	12
3	60	5	5	15
4	60	5	7	18
5	100	150	10	62

Table 2. Graph Area Measurements.

Data Source	Area before post-processing	Area after post-processing
1	101775	101775
2	155800	171000
3	534435	562275
4	671824	704740
5	870499	900467

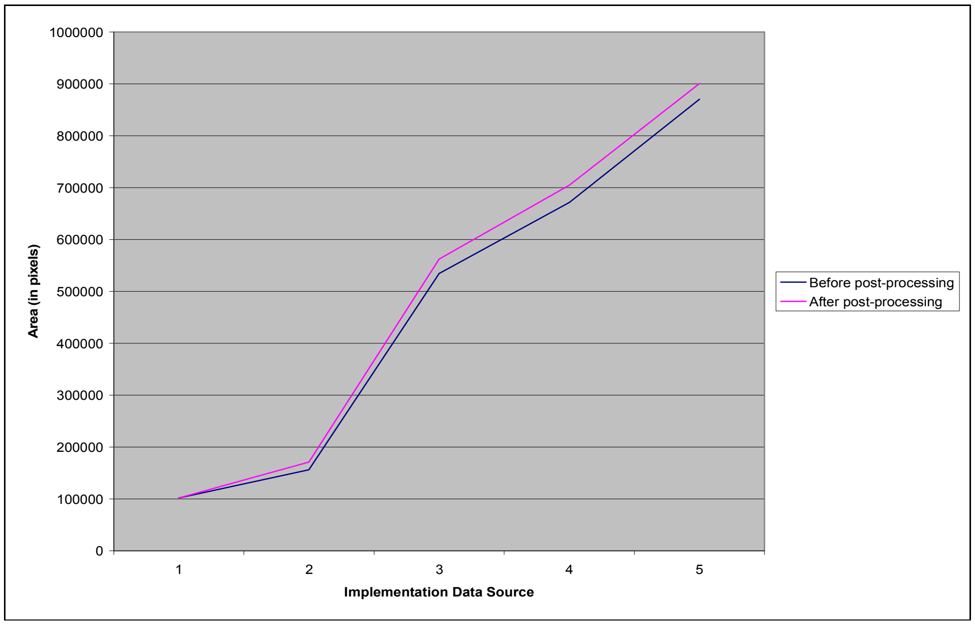


Fig. 11. Area sizes before and after post-processing.

6 Conclusion

E-Spring algorithm was proposed to draw clustered directed acyclic graphs (DAGs), and uses physical properties of electric charges and spring constants to remove overlaps between node labels. Since DAGs are only a *subset* of graphs, we apply E-Spring algorithm to include *general graphs*. The usefulness of a graph drawing depends on its *readability*, which is the capability of conveying the meaning of a diagram quickly and clearly. Readability issues are expressed by aesthetics such as minimization of crossings among edges and nodes, and removal of node overlaps. Our clustering structure of the graph is built on its own backbone tree derived from the existing graph structure. This backbone tree *T* is a spanning tree which is extracted through the implementation of breadth-first-search algorithm. The E-Spring algorithm guarantees that the visualized tree is free of node overlaps. We further improved graph readability by removing residual edge-node crossings using the edge routing method after the non-tree edges are inserted. The choice of the root node with the *highest node degree* for breadth-first search (BFS) algorithm results in better visualizations. Implementation was performed on eBay data and Graph catalog benchmark data with varying graph sizes, nodes with varying label string lengths, different source node selections and our experimental visualization results are promising. The resultant graph layouts are free of node overlaps and edge-node intersections. Future work includes enhancing E-Spring algorithm to accept input graphs for visualization, aesthetic comparisons for various graphs, and improving the quality of resultant drawings for large graphs.

References

1. Battista, G.D., Eades, P.: Algorithms for drawing graphs: an annotated bibliography. Computational Geometry. 4, 235--282 (1994)
2. Battista, G.D., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, New Jersey, USA (1999)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press and McGraw-Hill, New York, USA (2001)
4. Cruz, I.F., Twarog, J.P.: 3D Graph Drawing with Simulated Annealing. In: Proceedings of the Symposium on Graph Drawing, pp. 162--165. Springer-Verlag, London, UK (1995)
5. Davidson, R., Harel, D.: Drawing Graphs Nicely Using Simulated Annealing, ACM Transactions on Graphics. 15, 301--331 (1996)
6. Dijksta, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik. 1, 269--271 (1959)
7. Dolulil, J., Katreniakova, J.: Edge Routing with Fixed Node Positions. In: Proceedings of the 12th International Conference Information Visualisation, pp. 626--631. IEEE Computer Society Washington, DC, USA (2008)
8. Dwyer, T., Marriott, K., Wybrow, M.: Integrating edge routing into force-directed layout. In: Proceedings 14th Intl. Symp. Graph Drawing (GD '06), pp. 8--19. Springer-Verlag, Berlin, Heidelberg (2007)
9. Eades, P.: A heuristic for graph drawing. Congressus Numerantium. 42, 149--160 (1984)
10. Eades, P., Huang, M.L.: Navigating Clustered Graphs using Force-Directed Methods, Journal of Graph Algorithms and Applications. 4, 157--181 (2000)

11. Eades, P., Lai, W., Misue, K., Sugiyama, K.: Layout adjustment and the mental map. Journal of Visual Languages and Computing. 6, 83--210 (1995)
12. Forster, M.: Crossings in Clustered Level Graphs, Ph.D.Dissertation (2004)
13. Healy, P., Nikolov, N.S.: Graph Drawing. Springer-Verlag, Berlin, Heidelberg (2006)
14. Huang, X., Lai, W., Sajeev, A.S.M., Gao, J.: A new algorithm for removing node overlapping in graph visualization. Information Sciences. 177, 2821--2844 (2007)
15. Kakoulis, K.G., Tollis, I.G.: A unified approach to labeling graphical features. In: Proceedings 14th Annual ACM Symposium of Computational Geometry (SoCG'98), pp. 347--356. ACM New York, NY, USA (1998)
16. Kaufmann, M., Dorothea, W.: Drawing graphs, methods and models. Springer-Verlag, Berlin, Heidelberg (2001)
17. Korb, K.B., Nicholson, A.E.: Bayesian Artificial Intelligence. Chapman & Hall, London, UK (2004)
18. Kruskal, J.B.: On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. American Mathematical Society. 7, 48--50 (1956)
19. Kumar, P., Zhang, K.: Social Network Analysis of Online Marketplaces. In: Proceedings IEEE International Conference on e-Business Engineering, pp. 363--367. IEEE Computer Society Washington, DC, USA (2007)
20. Kumar, P., Zhang, K., Wang, Y.: Visualization of Clustered Directed Acyclic Graphs without Node Overlapping. In: Proceedings 12th International Conference on Information Visualization, pp. 38--43. IEEE Computer Society Washington, DC, USA (2008)
21. Kumar, P., Zhang K.: Visualization of Clustered Directed Acyclic Graphs with Node Interleaving. In: Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC '09), pp. 1800--1805. ACM New York, NY, USA (2009)
22. Kumar, P., Zhang, K.: Node Overlap Removal in Clustered Directed Acyclic Graphs. Journal of Visual Languages and Computing (JVLC). Article in press, doi:10.1016/j.jvlc.2009.04.007 (2009)
23. Lai, W.: Layout Adjustment and Boundary Detection for a Diagram. In: Proceedings of Computer Graphics International (CGI '01), pp. 351--354. IEEE Computer Society Washington, DC, USA (2001)
24. Lai, W., Eades, P.: Removing edge-node intersections in drawings of graphs. Information Processing Letters. 81, 105--110 (2002)
25. Larson, R.C., Li, V.O.K.: Finding minimum rectilinear distance paths in the paths in the presence of barriers. Networks. 11, 285--304 (1981)
26. Lee, D.T., Preparata, F.P.: Euclidean shortest paths in the presence of rectilinear barriers. Networks. 14, 393--410 (1984)
27. Leiserson, C.E.: Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing. IEEE Transactions on Computers. 34, 892--901 (1985)
28. Leymann, F.: Web Services Flow Language (WSFL 1.0), IBM (2001)
29. Li, W., Eades, P., Nikolov, N.: Using spring algorithms to remove node overlapping. In: Proceedings of the 2005 Asia-Pacific symposium on Information visualization (APVIS '05), pp. 131--140. Australian Computer Society, Inc. Darlinghurst, Australia (2005)
30. Lozano-Perez, T., Wesley, M.A.: An algorithm for planning collision-free paths among polyhedral obstacles. Communications of ACM. 22, 22560--22570 (1979)
31. Marriott, K., Stuckey, P., Vam, T., He, W.: Removing Node Overlapping in Graph Layout Using Constrained Optimization. Constraints. 8, 143--171 (2003)
32. Nascimento, H.A.D., Eades, P.: User Hints for map labeling. Journal of Visual Languages and Computing. 19, 39--74 (2008)
33. Papakostas, A., Tollis, I.G.: Interactive Orthogonal Graph Drawing. IEEE Transactions on Computers. 47, 1297--1309 (1998)
34. Prim, R.C.: Shortest connection networks and some generalizations. Bell System Technical Journal. 36, 1389--1401 (1957)

35. Purchase, H.C., Cohen, R.F., James, M.: Validating Graph Drawing Aesthetics. In: Proceedings of the Symposium on Graph Drawing (GD '95), pp. 435--446. Springer-Verlag, London, UK (1995)
36. Quigley A., Eades, P.: Graph Drawing, Clustering, and Visual Abstraction. In: Proceedings of the 8th International Symposium on Graph Drawing (GD '00), pp. 197--210. Springer-Verlag, Berlin, Heidelberg (2000)
37. Taylor, M., Rodgers, P.: Applying Graphical Design Techniques to Graph Visualization. In: Proceedings of the Ninth International Conference on Information Visualization (IV '05), pp. 651--656. IEEE Computer Society Washington, DC, USA (2005)
38. ConceptDraw, http://www.conceptdraw.com/en/products/cd5/main.php
39. Home of Graphdrawing, http://www.graphdrawing.org
40. Graphviz - Graph Visualization Software, http://www.graphviz.org
41. Microsoft Office Online – Visio, http://office.microsoft.com/en-us/visio/default.aspx
42. yWorks, http://www.yworks.com/products/yfiles/doc/developers-guide/index.html