DAGView: An Approach for Visualizing Large Graphs

Evgenios M. Kornaropoulos[1,2] and Ioannis G. Tollis[1,2]

[1] Department of Computer Science, University of Crete, Heraklion, Crete, Greece
[2] Institute of Computer Science, Foundation for Research and Technology-Hellas, Vassilika Vouton, P.O. Box 1385, Heraklion, GR-71110 Greece
{kornarop,tollis}@ics.forth.gr

Abstract. In this paper, we propose a novel visualization framework called *DAGView*. The aim of DAGView is to produce clear visualizations of directed acyclic graphs in which every edge and the potential existence of a path can be immediately spotted by the user. Several criteria that users identified as important in a layout are met, such as underlying grid, crossings and bends that appear perpendicular. The main algorithm is based on the layout of directed acyclic graphs but can be extended to handle directed graphs with cycles and undirected graphs, taking into account user preferences and/or constraints. Important tasks that are used in user studies are performed efficiently within the DAGView framework.

1 Introduction

Node link diagrams are a well studied representation of graphs. There are numerous techniques proposed by the graph drawing community that study geometric features, readability and aesthetic rules [4]. One standard technique for visualizing graphs is the orthogonal model, a model that has been widely studied and advanced through the years [2,3,16,17]. In this model each edge is mapped into a chain of horizontal and vertical line segments. The fact that bends are drawn solely as right angles assists the readability of the diagram. Not only bends, but also crossings appear perpendicular between the edges in orthogonal drawings.

Some interesting questions were posed in the work of Ghoniem et al. [8], where the authors describe a user study comparing the matrix representation over the node link representation of graphs. Matrix-based representation is a visualization technique where a boolean-valued n-by-n connectivity matrix is used. Columns denote the origin-node of the edge, while the rows denote the target-node of the edge. If there is an edge (u, v) the cell at the intersection of column u and row v, is colored black. In [8] users were asked to give fast and correct answers on several tasks such as counting the number of nodes and edges, finding the most connected node, finding if there was an edge between two specified nodes as well as finding a path between nodes. The overlapping of links and the lack of topology were addressed by the users as the two main factors that caused difficulties in reading node-link representations.

W. Didimo and M. Patrignani (Eds.): GD 2012, LNCS 7704, pp. 499–510, 2013.

Several tools implement the matrix representation technique. MatrixExplorer [10] is a network visualization system that uses both node-link diagrams and matrices. The user can hop from one representation to the other as well as interacting with the matrix representation. Zoomable Adjacency Matrix Explorer (ZAME) [7] is a tool that takes advantage of the scalability of matrix representation and enables exploration of graphs with millions of nodes and edges. MatLink [9] is a tool implementing a composite representation, where links overlay on the border of the matrix visualization. Another recent paper [1] partially uses the matrix-based representation to produce hybrid visualizations.

In recent years there is a trend to study graph representation from a cognitive point of view. Experimental studies were deployed in order to evaluate the aesthetics and the readability of different graph layouts [21,22]. Examples of these aesthetics include the minimization of edge crossings and the minimization of bends. In [12] an eye tracking experiment was conducted in order to study the effects of crossing angles. Among the findings, authors concluded that when edges cross at 90 degrees, eye movement may be slightly slowed down, but it remains smooth. Grid drawing is an important goal for graph drawing techniques. Purchase et al. note that besides the minimization of edge crossings, aligning nodes and edges to an underlying grid is equally important [20]. In [6] the authors conduct an experiment to compare user-generated and automatic graph layouts. Users chose graphs based on general aesthetics, most commonly a symmetric, ordered, or clean look. We use the knowledge compiled by several human factors studies regarding the readability of layouts in order to design algorithms that aim to optimize the most important criteria.

In this paper, we propose the *DAGView* visualization framework which contains a family of algorithms based on the Overloaded Orthogonal technique [13]. The DAGView framework presents several characteristics that users identified as important: underlying grid, crossings that appear perpendicular, easy check for the existence of an edge and/or path. We extend the main algorithm which is based on the layout of directed acyclic graphs, in order to visualize directed graphs with cycles and undirected graphs, taking into account user preferences and/or constraints. Finally, we discuss important features of our approach that respect the user's choices as described in [8,20,21].

2 The DAGView Model

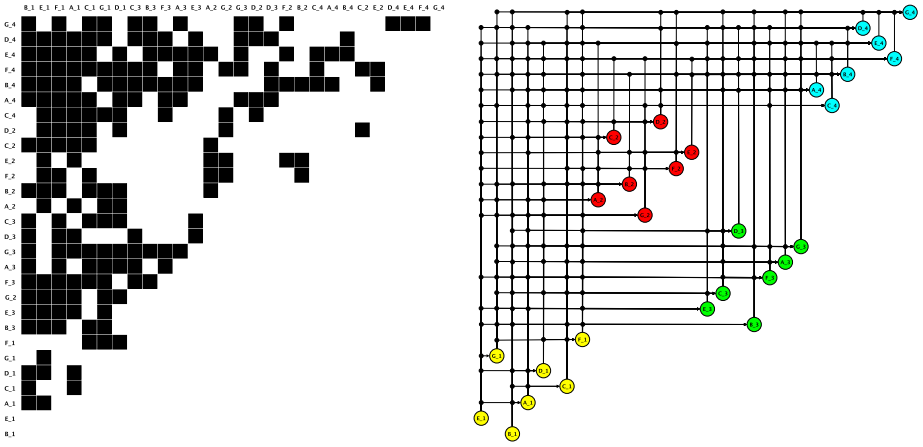
The DAGView model contains a family of algorithms based on the Overloaded Orthogonal technique [13]. Since dominance drawings cannot visualize every directed acyclic graph [14,15], a relaxed condition, called *weak dominance property*, was proposed [13]. The efficiency of this model comes from combining dominance and row/column reuse. In general, we want to find a pair of topological sortings that produce a visualization which is aesthetically pleasing and has a desired property. A possible direction of study is to find such a pair of topological sortings that place in different quadrants, clusters of nodes which are not

connected with each other. So, different desired visualization properties will require a different pair of topological sortings.

As for the edges, a straight line approach will produce several crossings, a situation that users indicated as the most confusing aspect of node-link representations [12]. Therefore we prefer to use the underlying grid in order to route the edges in a more clear manner. Each outgoing edge of a node uses the same column in order to reach its corresponding destination node. Intuitively, edges flow from bottom-to-top and from left-to-right. In order to resolve possible ambiguities of the model, we use a special class of points called *e-points*. An *e*-point is a small black point placed on the grid in order to indicate the connection from *e*-point's column to *e*-point's row. An example is shown in Figure 4.

Algorithm. DAGView ( )

1. Compute a topological sorting T_1 and a topological sorting T_2
2. Use the rank of each node in T_1 and T_2, as the assignment of X and Y coordinates, respectively.
3. For each edge (u, v):
   Draw a vertical edge segment from $(X(u), Y(u))$ to $(X(u), Y(v))$
   Draw a horizontal edge segment from $(X(u), Y(v))$ to $(X(v), Y(v))$
   Draw an *e*-point at $(X(u), Y(v))$



(a) (b)

Fig. 1. Two visualizations of the same directed acyclic graph containing 28 nodes and 197 edges. (a) depicts a matrix representation where columns appear in decreasing order of out-degree. (b) depicts a DAGView visualization.

Furthermore, the model can be easily extended in order to depict the transitive closure of the graph without rearranging the already drawn DAGView. If there

is an edge (u, v) and an edge (v, z), then node z is reachable from u. If node u and z are not directly connected with an edge, then there would be no e-point at $(X(u), Y(z))$. Thus, in the transitive extension of the DAGView model we simply add edge (u, z) by placing the vertical and the horizontal segment as we described before, but now the new e-point will be colored gray. These points that represent a path, are called p-points (p for path). With this technique we can have all the information about reachability in our directed acyclic graph preserving the 'mental map' created by the user from the original DAGView drawing.

3 Directed Graphs: Handling Cycles

The DAGView visualization technique, as described so far can only visualize directed graphs with no cycles. In this section we discuss techniques in order to be able to visualize directed graphs with cycles.

3.1 Presenting Feedback Arcs

By following the weak dominance property, we cannot visualize all the directed edges that are contained in a cycle. Besides, we cannot place the nodes in the grid since the existence of a cycle does not allow the construction of a topological sorting. Therefore, we will temporarily invert the direction of a set of edges such that the graph will become a directed acyclic graph. This set of edges is also known as feedback arc set. Computing the minimum feedback arc set is NP-hard, thus we use heuristics that compute a minimal feedback arc set. One such algorithm is the Greedy-Cycle-Removal algorithm [4].

Let $G = (V, E)$ be a directed graph with cycles and Γ its visualization according to the DAGView framework. First we compute the minimal feedback arc set and denote it as E'. Next, we invert the direction of the edges in E'. The DAGView model is used to visualize the new directed acyclic graph since it contains no cycles. Then we visualize the edges of the graph $G = (V, E - E')$ according to the DAGView model. We focus on how to visualize an edge $(v, u) \in E'$ that belongs to the minimal feedback arc set. Notice that even though u is reachable from v, node u is placed in the lower-left quadrant of v. Also note that all the incoming edges of node u in Γ are assembled in the row $Y(u)$ of the grid. Thus, the $Y(u)$ row has no edge segment to the right of point $(X(u), Y(u))$. Similarly, all the outgoing edges of node v are assembled in the column X(v) of the grid. Thus, the $X(v)$ column has no edge segment below the point $(X(v), Y(v))$. We use these empty segments a) to the right of u and b) below v, to draw the feedback arc (v, u). But with this edge visualization of (v, u) we violate the weak dominance property. In order to highlight this violation, the edges in E' are colored red and are oriented from top-to-bottom and from right-to-left. Formally, each edge $e' = (v, u) \in E'$ will have a vertical segment from point $(X(v), Y(v))$ to point $(X(v), Y(u))$, and a horizontal segment from point $(X(v), Y(u))$ to point $(X(u), Y(u))$. Given a feedback arc (v, u) the corresponding e-point is defined as

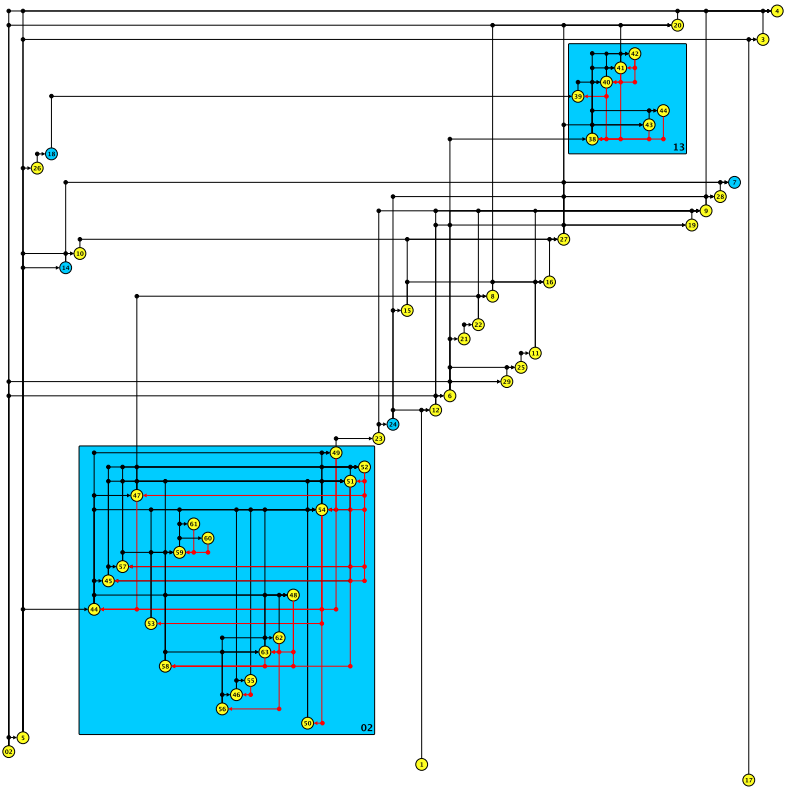


Fig. 2. A directed graph with 55 nodes and 135 edges, out of which 31 are marked as feedback arcs with red color. Nodes that are colored blue are strongly connected components that have been contracted into a single node. Inside the blue frames are depicted in detail two out of the six contracted strongly connected components.

an unlabeled red point that is placed on point $(X(v), Y(u))$ to indicate a direct connection from v to u.

3.2 Handling Strongly Connected Components

Under certain circumstances users might want to inspect a more high-level picture of a directed graph with cycles. A clustering scheme for strongly connected components with the DAGView model is discussed. An example is depicted in Figure 2.

Strongly connected components can be computed in linear time by using standard algorithms. Next, components are contracted resulting in a new directed acyclic graph G' that has a super-node for each strongly connected component. Algorithm DAGView is applied in order to visualize G'. The visualization of each strongly connected component can be computed separately following the technique described in the previous subsection.

A strong advantage of our framework is that the detailed visualization of a strongly connected component can be easily added within the already drawn visualization of G'. An interactive feature of the DAGView model is the following: by clicking on the contracted strongly connected component $G_i = (V_i, E_i)$ of G', a square $|V_i|$-by-$|V_i|$ will be extended where the detailed visualization of G_i will appear. All the nodes that reach G_i have been placed in the lower-left quadrant of G_i, whereas the nodes that can be reached from the nodes of G_i appear in the upper-right quadrant of G_i. Thus, the rest of the visualization does not change and we preserve the user's mental map of G' [19].

Algorithm. Strongly Connected Clustering ( )

1. Compute strongly connected components $S = \{S_1, ..., S_k\}$ of G
2. Contract each component S_i into a super-node s_i, obtaining G'
3. Compute X, Y coordinates of G' and construct the visualization according to the DAGView model
4. For each strongly connected component S_i compute a minimal feedback arc set, construct and save separately the DAGView of S_i.
5. Replace the node s_i of the DAGView with its detailed visualization whenever the user clicks on s_i

3.3 User-Chosen Feedback Arcs

In the previous subsection we chose the feedback arc set according to the output of an algorithm. Here we discuss the scenario where the user chooses a number of edges that (s)he wants to be a part of the feedback arc set. An example may come from the model of Resource Description Framework (or RDF). Several techniques visualize RDF graph ontologies from top-to-bottom. But there are classes of edges that represent a different relation between the nodes and are directed from bottom-to-top. The visualization of such graph ontologies has been extensively studied [23].

In this approach the user chooses an edge set E_u that will be a subset of the feedback arc set. If the set E_u is not a feedback arc set, the Greedy-Cycle-Removal algorithm is applied on the directed graph with edge set $E - E_u$. At the output of Greedy-Cycle-Removal, edges of E_u are added resulting in an augmented feedback arc set. Then we temporary inverse the direction of the edges of the augmented feedback arc set. A DAGView visualization is produced according to the new directed acyclic graph. Finally the edges of the augmented feedback arc set are visualized in red following the orientation from top-to-bottom and from right-to-left, according to the technique described before.

4 Undirected Graphs

In this section we propose methods to visualize undirected graphs within the DAGView model. One mainstream technique that has been widely used, is to

orient the undirected edges in order to obtain a directed graph. The first approach in this framework is to orient edges in order to avoid cycles. The computation of an *st*-orientation is in this direction. Specifically, one can use a parametrized technique in order to control the length of the longest path in the resulting *st*-oriented directed graph [18]. The second approach is to construct a directed graph with cycles.

In this scenario the user gives as an input a set of node-disjoint cycles $C = \{C_1, ..., C_k\}$ of an undirected graph $G = (V, E)$. The goal is to orient the edges of each cycle C_i in order to form a strongly connected component c_i which can be easily contracted into a super-node as described in the previous section.

We will discuss how to handle cycle $C_i \in C$. Let (w, z) be an undirected edge of C_i. Let us also assume that the user chose a direction for the edges of the cycle, in which the undirected edge (w, z) is oriented from w to z. Each undirected chordal edge of C_i is oriented so as to conform with the direction from z to w. After we have oriented all the edges of cycle C_i and its chordal edges, we temporarily invert the direction of (w, z) and put it in the edge set E'. The set E' is formed in order to visualize the feedback arc set. We repeat the above process for every cycle in C.

Next, we compute an *st*-orientation of the remaining undirected edges of G, considering each cycle C_i as a contracted super-node c_i. The reason we handle cycles as super-nodes is because we want to have consecutive X and Y coordinates for the nodes of each cycle. The directed acyclic graph $G = (V, E - E')$ is visualized according to the DAGView model. The direction of edges of E' is inverted back to the original direction and are drawn in red as feedback arcs. In the final visualization the strongly connected components can be visualized either in their original form or as super-nodes.

5 Visualizing Large and Disconnected Graphs

In this section we will address some issues on the topic of visualizing graphs with many nodes. Specifically, we discuss techniques to handle different connected components of the same graph as well as a feature to assist the readability of the drawing.

5.1 Connected Components and Tiles

We show how to visualize graphs that have a number of different connected components. We will handle each connected component as a separate graph and consider its visualization as a tile (or rectangle). With the assumption that each connected component is treated independently, we reduce the total area of the visualization. For instance one can visualize components separately and then place them horizontally in decreasing size order. In this case exactly one node from each of the k components, will have Y coordinate equal to 1.

Thus, the visualization of the graph is performed in two steps, in the first we visualize the connected component and in the second we place the k tiles

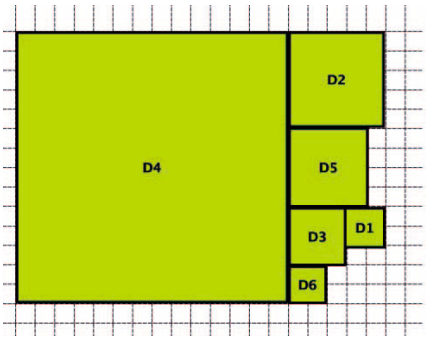


Fig. 3. A graph with six connected components $D = \{D1, ..., D6\}$. The connected components(rectangles) are placed so as to reduce the overall area of the visualization.

according to an algorithm. The placement of the tiles can be described as a rectangle packing problem [11]. It is known that this optimization problem is NP-hard. Therefore, heuristics and approximation algorithms must be used in order to visualize the graph in a compact manner.

5.2 Edge Highlight

Highlighting an edge is a common feature in graph drawing tools, especially when dealing with large graphs. This feature is implemented in the DAGView model as follows: when the user places the mouse over an e-point, the edge is highlighted. This approach assists the user in locating nodes that are placed in a great distance by inspecting the e-point that connects them. This simple but effective interactive enhancement helps users focus on graph elements. In most graph drawings the user has to visually follow a highlighted edge which changes orthogonal orientation at every bend. In our case the user anticipates a certain geometry in the edge that will be highlighted, a vertical segment followed by a horizontal. The feature that we propose will be of great help in experimental studies where users are answering an extensive series of questions and they may need assistance in performing more complicated tasks. An illustrative example is depicted in Figure 4.

6 Comparing with User Studies

According to user studies performed in [8], for small graphs node link diagrams are always more readable and more familiar than matrices. On the contrary, for larger graphs the performance of users on node link diagrams deteriorates quickly while matrices remain readable with comparable answer time.

Here we discuss the features of the DAGView model in terms of the purely relational method which is extensively used in the experimental user studies. In the relational method, the primary concern is the accuracy and efficiency with

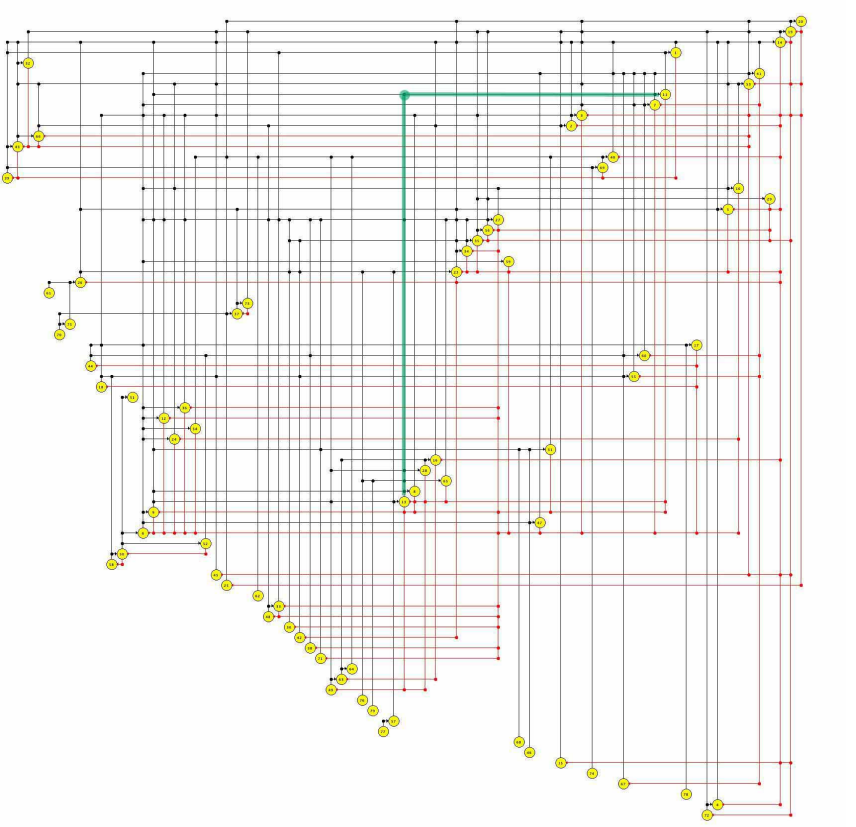


Fig. 4. A DAGView visualization of a directed graph with cycles with 79 nodes and 335 edges. In order to check if there is an edge from node u to node v in a DAGView visualization, we have to inspect only one point of the grid $(X(u), Y(v))$. If there is an e-point, then there is an edge. In this visualization the edge is highlighted with a green color.

which people can read the graph structure and answer questions about it. The tasks that we discuss are posed to users at several experimental studies [8,12,22]. Just the vertex placement alone contains the matrix representation for any directed acyclic graph. Notice that the placement also implies the weak dominance property [14,15].

Find an Edge. In a DAGView visualization in order to confirm the existence of an edge (u, v) the user has to focus on only three points of the visualization. The node u which is located in $(X(u), Y(u))$, the node v which is located in $(X(v), Y(v))$ and the point $(X(u), Y(v))$ of the underlying grid. If there is an e-point at $(X(u), Y(v))$, then edge (u, v) is part of the graph. Note that if node v is in the lower-left quadrant of node u then the edge is a backedge and it is depicted in red. Whereas, in the matrix representation the user has to locate the column that corresponds to node u, and the row that corresponds to node v.

Then if the point in the intersection of the column of u and the row of v is black, the edge (u, v) belongs to the edge set of G. So, the task of finding an edge in a DAGView visualization is as effective as in a matrix-based representation.

Locate Nodes. The DAGView model is a node-link representation. Therefore the task of locating a node is very intuitive. The user has to inspect the plane in order to locate the node, a common procedure to users that are familiar with graphs. However in a matrix-based representation, nodes are depicted as linearly ordered labels at the column/row. In this case users might have difficulty recalling the exact position of the column that they spotted in a previous task, since there isn't any distinctive feature (such as location on the grid or topology) to assist their memory.

Count Nodes. Since each node has a distinct coordinate, columns and rows of a DAGView visualization represent nodes of the graph. This fact reduces the task of counting nodes into the task of counting the number of rows in the underlying grid of the DAGView visualization. In a node-link representation, as studies have pointed out [8], users have difficulty when confronting this task in large graphs. A possible reason for this behavior is that the user has to scan the two-dimensional plane of the visualization while counting nodes. On the other hand in the matrix representation model (as in the DAGView model), the problem is reduced to counting in one dimension instead of two.

Find a Path. Paths can be easily visualized in the DAGView model for directed acyclic graphs. Let Γ be a DAGView visualization of a directed acyclic graph G. If the user is aware of the existence of a path from u to v and has located node u, then due to the weak dominance placement (s)he has to inspect only the upper-right quadrant of u in order to locate v. This property reduces the area of the two dimensional plane that the user has to check for node v. Moreover, the feature of the transitive closure extension offers all the reachability information of a directed acyclic graph with the touch of a button. In a matrix-based representations this task is complex since nodes are represented twice (once on both axes of the matrix), which forces the user's eye to move from the row representing a node to its associated column back and forth. Whereas in a DAGView visualization, the user has both the row and the associated column once (s)he locates the node in the plane.

Find the Most Connected Node. In a DAGView visualization the outgoing edges of a node u are located in the column $X(u)$, and the incoming edges in the row $Y(u)$. To locate the most connected node, the user has to glance at a row/column that is proportional to the degree of the node. Even though in this task we are interested only in the number of neighbors, when processing e-points of a column the name of the neighbors comes in at no cost since there can be only one node in each row. Users might have better performance on this task in a node-link representation since they are inspecting locally a node in order to count the edges that have u as an origin/destination. By following mainstream graph drawing techniques for visualizing dense graphs, the user will have difficulties counting the edges that are associated with a node.

Besides these typical tasks, there are generic characteristics of a DAGView visualization that are helpful to users. Several of the following characteristics were addressed in experimental studies by the users themselves as 'important features' [12]. Also some of the addressed features were observed in studies [20] where the user was free to ideally visualize the graph from scratch.

Perpendicular Crossings. As indicated in [12] crossings that appear perpendicular have the minimum effect on the readability of the visualization. The DAGView model not only constructs perpendicular crossings, but also 'hides' a great number of them due to the column/row reuse. This general characteristic of this model enables us to study the problem of minimizing edge crossings from a completely different perspective.

Visualize on an Underlying Grid. According to experimental studies [20], when users are asked to draw a graph from scratch, they tend to use a unit grid formation to place the nodes. Moreover they draw edges vertically and horizontally, this indicates that the structure used by the DAGView follows the tendencies of the users.

Scalable Model. The DAGView model aims at providing a scalable visualization technique for navigation and easy exploration through large scale graphs. As in a matrix-based representation [8], the readability of a DAGView visualization does not deteriorate when the size of the graph and the link density increases. Large graphs are modularly displayed in order to obtain clear and readable visualization. On the other hand straight-line graph drawing techniques are sensitive to density and size variation [8].

Preserving the Mental Map. User's mental map of the DAGView mental map [19] is preserved by the following important features: the clustering of strongly connected components, the user-chosen cycles and the transitive closure extension. This is especially important when the user has to perform several tasks sequentially on the same graph. In that case the user might need to recall the topology of the layout. In the clustering scheme the algorithms modify the visualization locally while preserving the relative position of the nodes. Whereas the transitive closure feature extends the visualization of the directed acyclic graph without changing the position of the already drawn features of the DAGView visualization.

In conclusion the DAGView model has several of the advantages of matrix-based and node-link representations, while avoiding several of the disadvantages of each one.

References

1. Batagelj, V., Brandenburg, F.J., Didimo, W., Liotta, G., Palladino, P., Patrignani, M.: Visual Analysis of Large Graphs Using (X, Y)-Clustering and Hybrid Visualizations. IEEE Trans. Vis. Comput. Graph. 17(11), 1587–1598 (2011)
2. Biedl, T., Thiele, T., Wood, D.R.: Three-Dimensional Orthogonal Graph Drawing with Optimal Volume. Algorithmica 44(3), 233–255 (2006)
3. Binucci, C., Didimo, W., Liotta, G., Nonato, M.: Orthogonal drawings of graphs with vertex and edge labels. Journal Comput. Geom. Theory Appl. 32(2), 71–114 (2005)

4. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of graphs. Prentice - Hall, New Jersey (1998)
5. Di Battista, G., Tamassia, R., Tollis, I.G.: Area Requirement and Symmetry Display of Planar Upward Drawings. Discrete and Comput. Geom. 7(4), 381–401 (1992)
6. Dwyer, T., Lee, B., Fisher, D., Inkpen Quinn, K., Isenberg, P., Robertson, G.G., North, C.: A Comparison of User-Generated and Automatic Graph Layouts. IEEE Trans. Vis. Comput. Graph. 15(6), 961–968 (2009)
7. Elmqvist, N., Do, T.-N., Goodell, H., Henry, N., Fekete, J.-D.: ZAME: Interactive Large-Scale Graph Visualization. In: Proc. of IEEE Pacific Vis., pp. 215–222 (2008)
8. Ghoniem, M., Fekete, J.-D., Castagliola, P.: A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations. In: Proc. of the IEEE Symposium on Information Visualization, pp. 17–24 (2004)
9. Henry, N., Fekete, J.-D.: MatLink: Enhanced Matrix Visualization for Analyzing Social Networks. In: Baranauskas, C., Abascal, J., Barbosa, S.D.J. (eds.) INTERACT 2007, Part II. LNCS, vol. 4663, pp. 288–302. Springer, Heidelberg (2007)
10. Henry, N., Fekete, J.-D.: MatrixExplorer: a Dual-Representation System to Explore Social Networks. IEEE Trans. on Visualization and Computer Graphics 12, 677–684 (2006)
11. Huang, E., Korf, R.E.: New improvements in optimal rectangle packing. In: International Joint Conference on Artificial Intelligence, IJCAI 2009, vol. 6, pp. 511–516 (2009)
12. Huang, W.: Using eye tracking to investigate graph layout effects. In: Proc. of Int. Asia Pacific Symposium on Visualization, APVIS, pp. 97–100 (2007)
13. Kornaropoulos, E.M., Tollis, I.G.: Overloaded Orthogonal Drawings. In: van Kreveld, M., Speckmann, B. (eds.) GD 2011. LNCS, vol. 7034, pp. 242–253. Springer, Heidelberg (2012)
14. Kornaropoulos, E.M., Tollis, I.G.: Weak Dominance Drawings for Directed Acyclic Graphs. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 566–568. Springer, Heidelberg (2013)
15. Kornaropoulos, E.M.: Dominance Drawing of Non-Planar Graphs, Masters Thesis, Department of Computer Science, University of Crete (2012)
16. Papakostas, A., Tollis, I.G.: Efficient Orthogonal Drawings of High Degree Graphs. Algorithmica 26(1), 100–125 (2000)
17. Papakostas, A., Tollis, I.G.: Algorithms for Area-Efficient Orthogonal Drawings. Computational Geometry Theory and Applications 9(1-2), 83–110 (1998)
18. Papamanthou, C., Tollis, I.G.: Algorithms for computing a parameterized st-orientation. Theoretical Computer Science 408(2-3), 224–240 (2008)
19. Purchase, H.C., Hoggan, E., Görg, C.: How Important Is the "Mental Map"? – An Empirical Investigation of a Dynamic Graph Layout Algorithm. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 184–195. Springer, Heidelberg (2007)
20. Purchase, H.C., Pilcher, C., Plimmer, B.: Graph Drawing Aesthetics-Created by Users, Not Algorithms. IEEE Trans. Vis. Comput. Graph. 18(1), 81–92 (2012)
21. Purchase, H.C.: Which Aesthetic Has the Greatest Effect on Human Understanding? In: Di Battista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 248–261. Springer, Heidelberg (1997)
22. Purchase, H.C., Cohen, R.F., James, M.I.: An experimental study of the basis for graph drawing algorithms. J. Exp. Algorithmics (JEA) 2(4) (1997)
23. Tzitzikas, Y., Hainaut, J.L.: On the visualization of large-sized ontologies. In: Proc. of the Workshop on Advanced Visual Interfaces, pp. 99–102 (2006)