IADIS International Journal on Computer Science and Information Systems
Vol. 8, No. 2, pp. 12-26
ISSN: 1646-3692

STABLE INCREMENTAL LAYOUTS FOR DYNAMIC GRAPH VISUALIZATIONS

Martin Steiger. *Fraunhofer IGD, Fraunhoferstr. 5, 64283 Darmstadt*

Thorsten May. *Fraunhofer IGD, Fraunhoferstr. 5, 64283 Darmstadt*

Jörn Kohlhammer. *Fraunhofer IGD, Fraunhoferstr. 5, 64283 Darmstadt*

ABSTRACT

In this paper we present a set of extension techniques to stabilize interactive dynamic graph layout algorithms. It works with different existing Focus & Context methods. We first deal with the initial placement of newly inserted nodes to mitigate acting forces in the layout algorithm. Then, their influence on the existing layout is gradually increased to create a smooth transition between the old and the new layout. To complement this approach we use a look-ahead strategy that integrates additional nodes in the layout to stabilize the layout even more. Our approach is validated using both quantitative and qualitative measures.

KEYWORDS

Graph drawing, dynamic graph views, stable layout, mental map

1. INTRODUCTION

The tasks in the 2012 VAST Challenge[1] are about finding suspicious events that happened during just two days. The given data set describes states and connections of all computers in a fictive company network. The major problem here as well as in most real world scenarios is the sheer amount of data. In total, 160 million nodes are available in the database.

A visual representation of the data is vital to gain insight and decide what is suspicious and what is not. Obviously, displaying all entities and all interconnections at the same time cannot produce meaningful results. Screen space is always limited, especially compared to the ever-increasing amount of graph data, so massive overdraw is the consequence for such

[1] http://www.vacommunity.org/VAST+Challenge+2012

visualizations. But also visual perception has its limits. Even if it would be possible to draw millions of nodes, the data analyst would not be able to deal with that information. Typically, the user is interested in a small part of the graph, maybe even a single node. The challenge in the visualization is to preserve a context that is large enough for the user to orientate and to navigate towards potentially interesting nodes and to solve a given task.

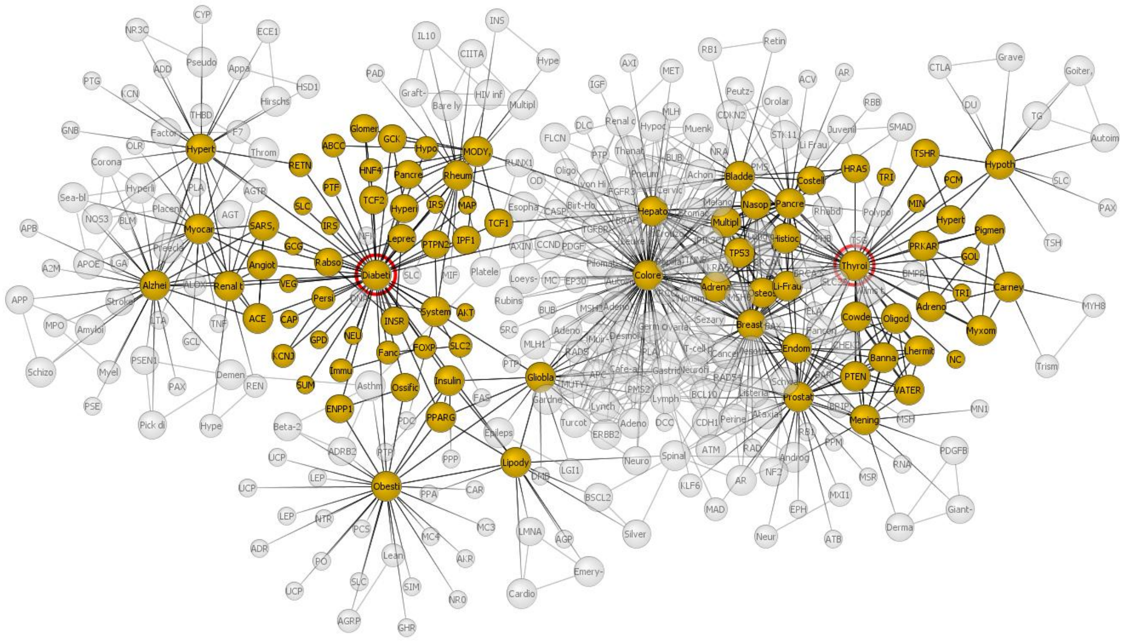

Figure 1. A screenshot from the running system showing a partial view of a graph including a set of focal nodes (marked with circles) and a set of visible context nodes. Also, a number of additional, potentially interesting ghost nodes (semi-transparent) are integrated to stabilize the layout, but not shown to the user.

Therefore, the layout of the displayed subgraph that is used in these techniques plays a key role. For small to medium-sized graph a pre-computed global layout with fixed node positions has its advantages. Above all, the visualization of a specified graph will always look the same which helps to build a mental map of the data set. In the navigation process, the nodes are simply toggled between visible and invisible state while their positions remain the same. But this also means that nodes that do not exist in the partial view influence the position of the visible nodes. This effect can be seen in Figure 1.

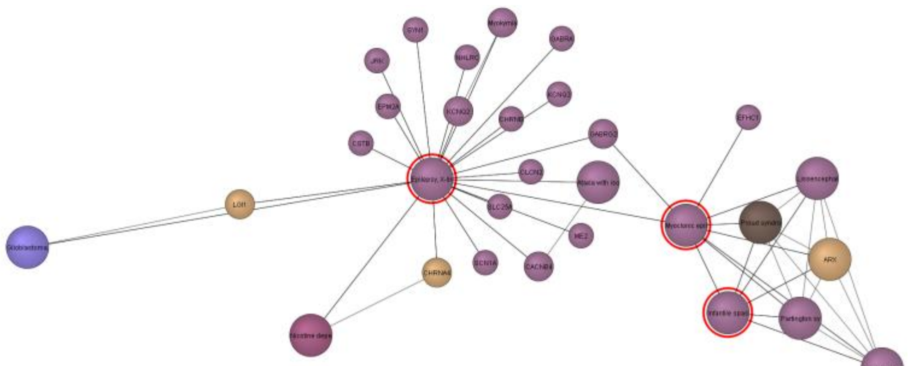

Figure 2. Some nodes are strongly pulled outwards without any visible explanation. They have to fulfill all constraints that are imposed by the full graph (adapted from May et al. with permission)

The time and memory limitations render global layout impractical for large graphs. Also, a fair amount of the computed layout is never used in the navigation process later on. One way to handle this is to perform the layout computation on-the-fly for the area that is currently visible. This process works quite well, if the displayed subgraph does not change over time. Otherwise, these topological changes can have a strong and sudden impact on the visual representation if they are not carefully dealt with. In order to communicate the performed changes it is important to create smooth transitions, for example through animation, so that the user can better understand which nodes are appearing or disappearing.

We claim to contribute a combination of features that improve the user experience with focus-based navigation concepts:

- A new placement algorithm that puts new nodes close to their neighbors so that acting forces are kept minimal.
- A weight function that integrates new nodes in the existing layout in a smooth transition by gradually increasing their mass.
- A layout looking ahead for nodes that potentially visible after the next focus node change.

These features can be used as separate improvements or in an integrated system. As a result, nodes are added and removed smoothly, keeping the visual changes in the existing layout to a minimum. We think that their effect on the layout reduces the cognitive effort for the user and thus support the preservation of the mental map.

The rest of the paper is organized as follows: In the next section we give a short overview on related work in the area. In Section 3 we present the data structures and algorithms we use before we go into implementation details in Section 4. Section 5 covers the test results and describes limitations. Finally, Section 6 concludes and discusses future work.

2. RELATED WORK

A survey on visual graph analysis in general has been compiled by von Landesberger (Von Landesberger et al, 2011). It covers a variety of analysis techniques for node-link diagrams as well as matrix representations for partial graph views or dynamic graphs. A typical problem of some graph layout techniques is the at least partially random behavior caused by a random initialization of node positions.

Eades noted that the *visual order* of screen elements should not change for animated diagrams in order to prevent user irritation (Eades et al, 1991), but he original definition of the term "mental map" has been coined by Misue et al. (Misue et al, 1995). They discuss three concepts on mental models: Orthogonality, proximity and topology.

Based on these ideas, Purchase presented an empirical study that strengthens the assumptions on the relevance for the understanding of node-link diagrams (Purchase et al, 2007). Depending on the task, preserving the mental map seems to be more complicated than earlier works indicated (Purchase and Samra, 2008). In a recent evaluation (Archambault and Purchase, 2013), the authors showed the effects of map stability to navigation tasks for animated and transitions and for small multiples.

While most of these concepts have been developed for the layout of dynamic graphs, we conclude that they also can be applied to extracted, dynamic subgraphs of large static graphs. For both cases the same requirement applies: Asignificant portion of the visible graph is topologically stable between any two changes. Different metrics on how to measure and optimize these changes has been presented earlier (Branke, 2001), but most of them compare only key frames of the full graphs. In this work, we will apply such measures to the animated transitions between key frames.

Mathematically speaking, navigation in partial graph views creates a series of subgraphs that are derived from a larger graph. Diehl and Görg presented a strategy for the transition between these subgraphs, but it requires that the navigation path must be known in advance, and the subgraphs required for the animation are derived from that (Diehl, Görg and Kerren, 2001). Because user interaction cannot be predicted, this approach cannot be adopted “as-is” for interactive browsing. Instead, interaction is supported by morphing (Diehl and Görg, 2002). Our approach solved this problem in a different way. Instead of only reacting to the user interaction, we prepare the graph layout for different possible scenarios.

Lee et al. apply quality measurements to an optimization framework based on simulated annealing (Lee, Lin and Yen, 2006). While this approach is generic, the given performance suggests that simulated annealing might not be suitable to support interactive browsing and real-time feedback, however.

Osawa combines traditional mass-spring layout with heat models (Osawa, 2001) to create stable layouts. The user distributes virtual heat energy to one or more nodes. Thermal radiation then distributes the energy to neighbors. Depending on the amount of inherent energy, nodes become larger or smaller, edges shorter or longer. While this approach seems to be promising, it requires the user to manually tweak the visualization.

For graphs with inherent hierarchical structures, a clustering can be built up and used for the layout. This also works for dynamic graphs, as Frishman and Tal show with their work (Frishman and Tal, 2004). Visual changes in the layout are reduced by adding “spacer” vertices that reserve space for future nodes. In their later work (Frishman and Tal, 2008) they present a drawing algorithm that is tailored for rendering online graphs (i.e. only previous frames are known) efficiently on GPUs. Similar to our work, the authors also define placement strategies for new nodes and an approach to reduce unnecessary node movement. However, the realization of these ideas and their evaluation method differ.

Some approaches make use of an evaluation function to extract interesting parts of the graph with respect to one or more selected vertices. This idea of a Degree of Interest function was brought up by Furnas who proposed to derive the set of most interesting points based on user interaction (Furnas, 1986). Heer and Card applied this concept to create a tree layout algorithm that works with multiple focus points (Heer and Card, 2004). Van Ham and Perer also showed that dynamic graph visualizations benefit from this approach (Van Ham and Perer, 2009). While their DOI function is based on a single focus point, May et al. extended the function to work with multiple focus points (May et al, 2012). However, their approach does not yet consider smooth transitions between the different extracted subgraphs.

3. DEFINITION OF GRAPH, CONTEXT AND FOCAL NODES

In mathematical terms, the data we work with has the form of a connected graph. It is defined as G(V, E) where V is a set of nodes or vertices and a set of edges E that connect some pairs of vertices. Our approach works with both directed and undirected graphs without limitations.

The technique we present is based on the browsing paradigm. This means that new nodes are reached through their neighbors. If the graph contains parts that are not connected to the rest of the graph, they cannot be reached. For the rest of the paper we will assume that the graph is connected, i.e. a path exists for every pair of vertices.

Based on the graph G we will extract the focus $F \subset G$, a subgraph that is displayed to the user as the visible part of the full graph. It is constructed from the focal nodes $Z \subset V$, a small set of vertices in G and a degree-of-interest function that defines the visible neighborhood of F. The focus nodes are initially selected by the user (e.g. through a textual search query) and thus considered to be the most interesting points for the analysis. The subgraph F surrounds the focal nodes and thus provides a context. The definition used here is analogous to the definition in the work of van Ham and Perer (Van Ham and Perer, 2009), but we decided to use the multiple focus points approach as described by May et al. (May et al, 2012) for two reasons.

First, changing a single focus point keeps most of the context as it is induced by the other nodes. Typically, the set of focal nodes is implemented as a first-in-first-out (FIFO) queue with limited size. Every time the analyst puts a new node in focus, it is added to the queue of focus points. It also adds its interesting neighbors to the visible graph. When the queue is full, the least-recently used focus node is dropped from the queue and its neighborhood is removed from the view. Using more than one focus node also keeps changes in the visual structure to a minimum which preserves the context better than the single-focus-version.

The second reason is that it also gives a sense of history which is particularly useful for browsing tasks where the path to the solution is not known beforehand or part of the solution. After the focal nodes have been picked the next step is to derive the context. To achieve that, a degree-of-interest function is used to evaluate the relevance of a node with respect to the current focus.

3.1 Initial Node Selection

The initial pick of focus nodes in a large-scale graph is a non-trivial task. Following the "Overview first, zoom and filter, then details-on-demand" paradigm by Shneiderman we notice that the overview is not available in partial graph visualizations (Shneiderman, 1996). In order to find potentially interesting regions of the graph, we cannot drill down from a high-level view to a specific region that demands our interest. We have to explicitly perform a search query that iterates over the full graph and select one or more of its results as a starting point. For example, a text search could provide all nodes whose attributes that match the expression. Alternatively, picking a keyword from a predefined list could select all vertices that are associated with that keyword.

3.2 The Degree of Interest Function

The concept of a function that evaluates the interest of a data element with the respect to the observer was first presented by Furnas (Furnas, 1986). The larger the distance from the current viewport the smaller seems to be the importance for the current task. Such a DOI function typically contains the following parts:

- An a-priori interest *(API)* that reflects the user-independent, general interest of a graph node *x*.
- The user interest *(UI)* function that represents the matching score to the user's current task *t*.
- It is complemented by a distance function *D* that measures the distance to the focus nodes in *Z*.

Denoting the set of search parameters as *t*, the resulting function can be expressed as:

$$DOI(x,y,z) = \alpha \cdot API(x) + \beta \cdot UI(x,t) + \gamma \cdot D(x,Z)$$

Furnas presented several possible mappings for the parts of this function. For example, the API function can be generated from inherent attributes of the node or its relevance in the structure. The UI function could match a node to a user defined text search query and the distance function can be mapped directly to the minimal weighted or un-weighted distance in a graph.

We will briefly sketch how the contextual subgraph can be constructed. A modified version of the Dijkstra algorithm can be used to derive the visible nodes around the focus points. Instead of using a single starting point, this variation works with multiple starting points, namely the focal nodes. The DOI of neighboring nodes is used as edge weight. In every iteration, the node with the highest interest is added to *F*. Typical stopping criteria are the number of total nodes in the set or a predefined threshold that filters out nodes with a DOI lower that a certain value. As it is, this function leads to disconnected graphs which are undesirable. A connected subgraph can be created by adding the shortest path between all focal nodes to the visible graph.

4. COMPUTING THE INCREMENTAL LAYOUT

We employ a classic force-directed layout algorithm based on the mass-spring-model that is described by Fruchterman and Reingold (Fruchterman and Reingold, 1991). Its main actors are spring tension and repulsion forces. For the initial set of nodes no placement constraints exist. Also, for subgraphs that are not connected to the rest, random placement can be used. All other nodes are at least constrained by edge forces that act between neighbors.

4.1 Placing New Nodes

The key idea here is to avoid sudden changes in the visualization through reduction of acting forces. Selecting a visible node as focus point creates a new, different set of context nodes. In order to keep the layout stable, new nodes should be placed with a distance equal to the rest length of the edge to satisfy spring constraints. However, it also desirable to place nodes as far away from all other nodes as possible to make use of free space and avoid repulsion forces.

The first aspect can be easily satisfied if the new node has only one edge. It can be placed anywhere on a circle with a radius equal to the desired edge length around the existing node. If two distance constraints exist the range of valid positions is reduced from a full circle to two points on that circle (see Figure2a).

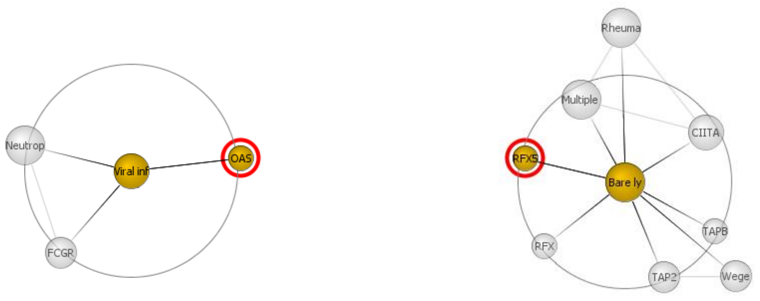

Figure 3. a) Up to two connected nodes (semi-transparent) can be placed on the circle around the existing vertex
b) Additional nodes are placed on a circle around the center of gravity of already inserted nodes

For nodes that have links to more than two neighbors, no general solutions exist that fulfills all criteria. We thus have to approximate the ideal position. First, the center of gravity of all neighbor nodes that are already in the layout is computed. Then, the final node position is a position on the circle around the center of gravity with a radius of the desired edge length (see Figure 2b). This node potentially lies outside the previously mentioned circle and does not fulfill any of the imposed constraints, but it is pushed away from the areas with high node density.

The placement on the circle is performed with respect to the number of nodes in its proximity. This area-based density measure is also required for the computation of repulsion forces of the Fruchterman-Reingold algorithm. Often-used methods to find empty space and perform distance tests quickly are spatial hashing algorithms. They define a grid-like structure of buckets that contain the graph vertices based on their position. Nodes are added and removed from the buckets as the move over the virtual grid. Given a point location the algorithm retrieves its corresponding bucket (based on hash keys). All other nodes in the bucket are considered to be close enough for further distance testing. Depending on the search distance, adjacent buckets are inspected, too.

We can make use of this data structure and query different positions on the circle. We count the number of nodes in the search area of every query location and choose the one with the lowest hit count. This ensures that new nodes are placed where most space is available.

Vertices that fall out of scope are removed from the visible graph. Typically, these nodes have been in the layout for a while and thus have "settled". The forces that act on them are rather small. Consequently, removing such a node hardly affects acting forces on other nodes and can thus be removed without having a strong visual impact.

4.2 Look-ahead for Invisible Neighbors

When new nodes are inserted in the layout and displayed immediately, they typically move a lot in order to satisfy all repulsion and edge forces. This can be mitigated by meaningful placement of new nodes but not cured entirely.

This is why we propose a look-ahead technique that also adds nodes to the layout that might become visible, if one of the currently visible nodes was selected as a focus point. Every time a node is added to the visible graph, the DOI function is evaluated for its neighbors as if the node was in the queue of focal nodes.

All nodes that would then become visible have already been added to the layout subgraph before. However, this subgraph can quickly become quite large, especially for highly connected graphs. A restrictive DOI function that filters out the majority of neighboring nodes mitigates the problem.

4.3 Gradually Increasing Node Weights

When many new nodes are placed around an existing node, the existing node will jitter heavily until the newly inserted spring forces have relaxed. We thus modify the algorithm by adjusting the mass of vertices over time. The goal of this modification is to adjust the inertia of nodes, depending on the time spent in the user's focus. Following the "separation of concerns"-pattern we first create a function that provides the number of layout iterations for every node. We track this by counting the write operations in the layout on a per node basis. Removing a node sets its counter back to zero.

We then derive the node weight from the normalized values of the counting function. It is desirable that invisible nodes reach the mass of visible nodes smoothly as the mass of a node is in direct relation with the acting forces that are used in the layout algorithm. For two vertices with mass *m_1* and *m_2* the distribution of forces can be computed as:

$$f_1 = \frac{m_2}{m_1+m_2} \qquad f_2 = \frac{m_1}{m_1+m_2}$$

The larger the mass of a particle is, the weaker is the influence of the force and the stronger is the influence on its counterpart. The two functions are plotted in Figure 3.

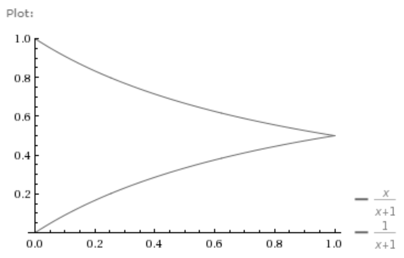

Figure 4. The weight function starts with full effect on the newly inserted node. The influence is shared as the number of layout steps increases until equilibrium is reached and both nodes are equally affected.

A similar idea has been presented by Huang et al. who introduce friction forces to stabilize dynamic layouts. The longer a node is visible the large its friction coefficient gets (Huang et al, 1998).

5. TEST RESULTS

The graph data that is used for the tests has been taken from the Diseasome[2] project. We extracted the largest connected subgraph (disconnected parts cannot be reached with exploratory search) which contained about 1500 nodes and 5500 edges. Although the edges per node ratio has an influence, our approach works on partial views only and is thus independent of the graph size per se.

Starting at a randomly chosen focus node, we explored its neighborhood over six focus changes, keeping all explored nodes visible. This created a sequence of six keyframe layouts with an increasing number of visible nodes (from 4 to 69). A transition between any two keyframes is calculated with 100 iterations of the simulation.

To test our methods we measured the node velocity of all visible nodes in three different settings. Few nodes with high velocities should implicate a larger penalty than many slow movements. This is why we measure the average of squared velocities.

[2] http://diseasome.eu/map.html

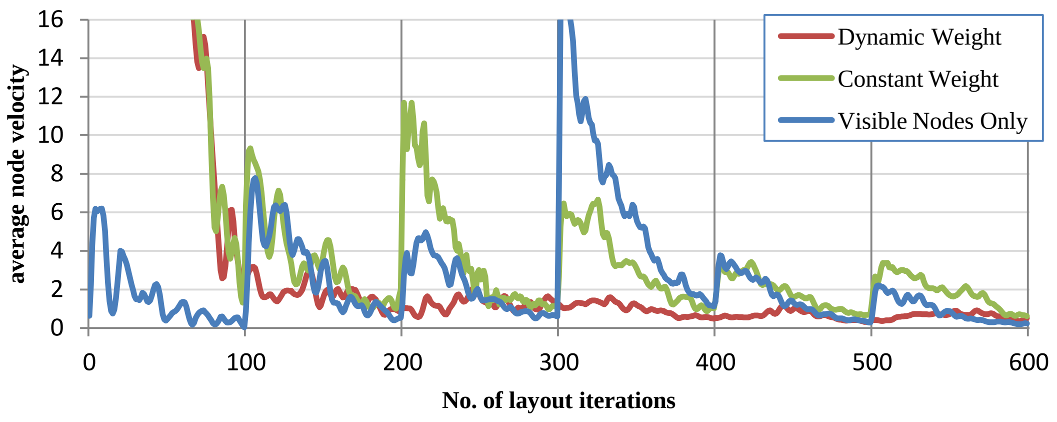

Figure 5 Node velocities for three different setups measured over six transisions with 100 layout steps each. Using only visible nodes has the highest avage node velocity (blue). Including invisible neighbors reduces velocity peaks (green). Using dynamic node weights reduces the peaks even more as the influence of invisible nodes is still very small after their insertion (red).

The first measurement (blue) was performed with constant node weight and without the invisible neighbor layout. It performs best during the first frame as only 4 nodes need to be laid out. However, the fourth subgraph contains a star node that introduces a large number of new nodes which causes the peak at iteration 300.

The second test run (green) also included the layout of the invisible neighborhood. The first subgraph contained additional 15 nodes that exert forces on the visible 4 nodes causing a rather poor performance for the first 100 iterations. At the start of frame 3 (iteration 200) a fair amount of nodes that could become visible in frame 4 has already been added. This causes a similar peak as in the first run, but less high and shifted by one keyframe. The peak at the begin of the forth frame is thus significantly smaller.

The gradual increase of node weights has also been activated for the third test run (red). As before, the average velocity for the 4 nodes in the initial frame is very high, but once they have settled down, it outperforms the other methods, especially when a large number of nodes is added simultaneously.

To validate the generality of our presented approach we conducted a second run of tests with a different dataset. Using the public API of a large online retailer we were able to access its enormous product database. The network that was used for our test contains roughly 550’000 entities linked by a binary ‘is-similar’ relationship. Roughly 1.2 million of those links were used to form the edges of the graph. Most of the listed products are associated with at least one category they belong to. This information was used to annotate nodes that are deemed to be important links towards interesting parts of the graph. Details on these visual cues can be found in the work of May et al. (May et al., 2011). To demonstrate the usefulness of the proposed approach we explored the graph using a dynamic view and recorded the generated animation. A set of selected still images was extracted and is presented in Figure 6-9.

The first series of images illustrates the expansion of the gray “Be” node at the upper right corner. Before being selected, all directly connected neighbors are already layed out properly (Figura 6a, transparent nodes). After the selection event the previously invisible nodes appear. Now, using the look-ahead strategy the neighbors of the newly appearing nodes are inserted into the layout (Figure 6b, transparent nodes). While still being invisible, the layout computes an optimal layout for these nodes in case the user explores in this direction which would make them visible.

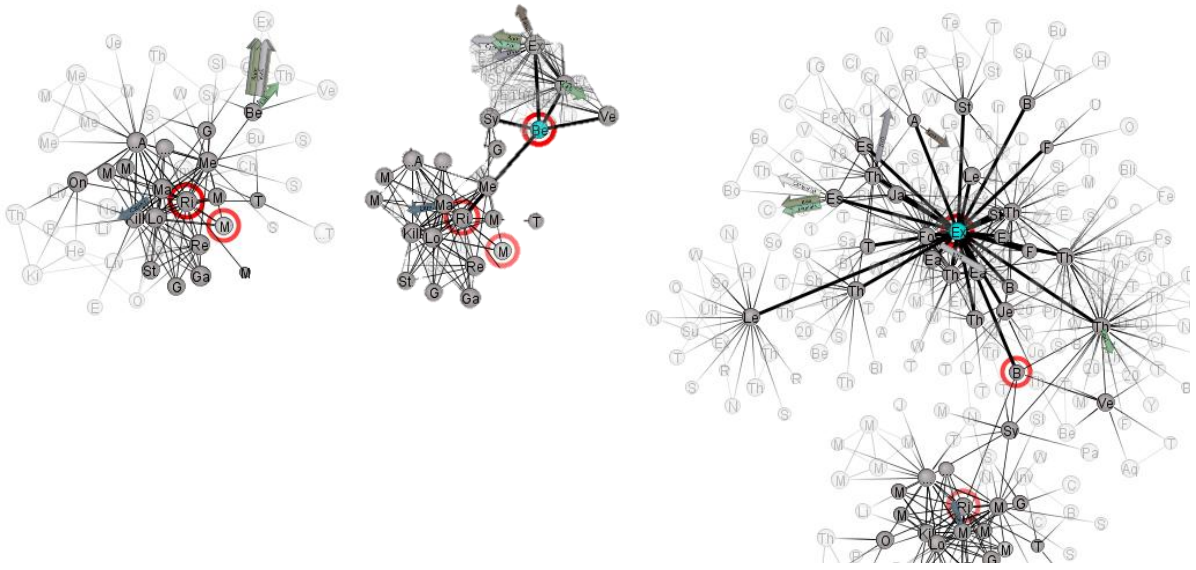

Figure 6. Exploration sequence in the product similarity graph. Note that the semi-transparent ghost nodes are not visible in the visualization, but exert forces. Left: original layout. Center: the node “Be” is selected for exploration. Its neighbors become visible and neighbors of neighbors are inserted into the layout without being visible for the user. Right: over time, the layout settles when a locally optimal solution has been found for all nodes.

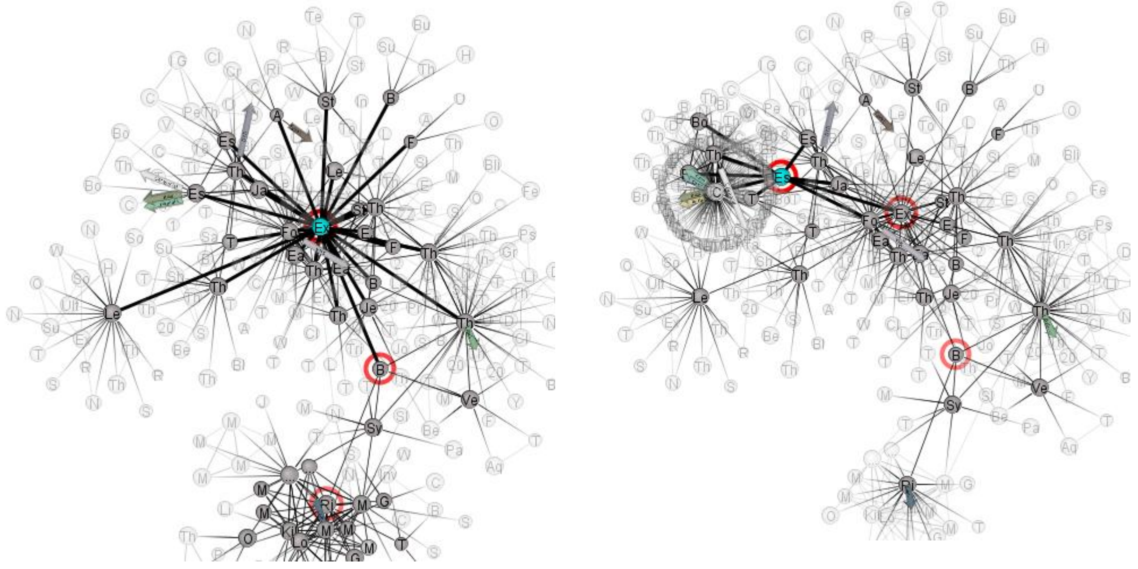

Figure 7. Exploring denser region of the graph shows how the node placement algorithm works. At the left side of the visible subgraph, the "Es" node is selected for exploration. This triggers the visibility of the star node "C" (right image). Adding all neighbors leads to a prominent ring around the node, indicating that most neighbors are only linked with "C". Those who also have other neighbors are put in the center of gravity which is also close-by.

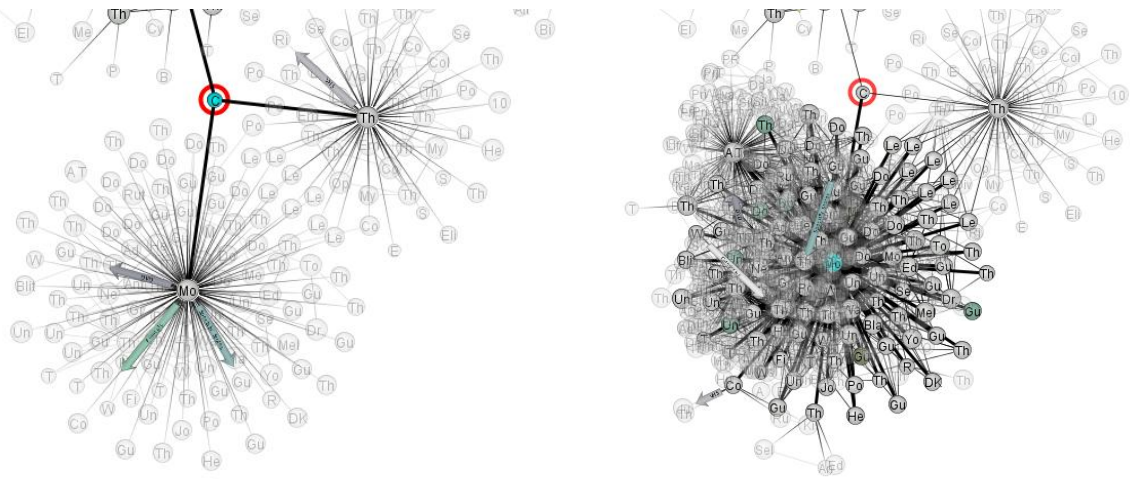

Figure 8. Again, a dense star node is being explored. Selecting the node triggers the insertion of a large amount of new nodes that are placed in an already dense area. This does not lead to visual clutter as the new nodes are invisible, but their repulsion forces act on the visible nodes.

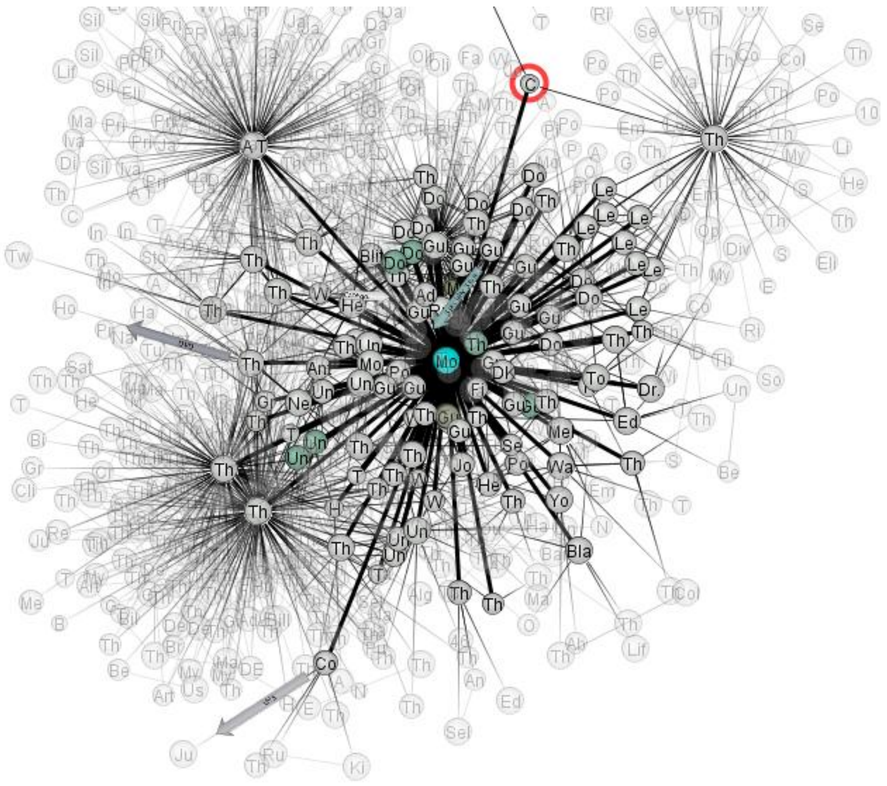

Figure 9. The newly inserted nodes pull some of the nodes away from the star node, especially those who are star nodes themselves (e.g. the “AT” and three “Th” nodes). The length of the visible edge here is no longer optimal, but the layout is already stabilized for future exploration. Pulling stars out also makes them more prominent on the screen which can be useful, as they are probably more important than nodes with only one link.

6. CONCLUSION & OUTLOOK

In this paper we describe contributions that increase the quality of incremental layout algorithms. A set of nodes that will possibly become visible in the foreseeable future is already integrated in the layout computation to keep the visual representation stable when they are shown. After they have been placed close to their neighbors, their influence on the layout is gradually increased from almost none to full effect. This keeps the transition between different focal node sets smooth.

The look-ahead strategy we propose significantly improves the layout. Star nodes that are visible have many invisible neighbors that pull the node outwards. This indicates that the vertex is highly connected and possibly worth exploring, even if the connection cannot be seen. On the downside, a high edge to node ratio introduces a large number of nodes on the layout which can cause slowdowns.

Currently, navigating from one part to another and back does not necessarily produce the same visual structure. Whenever a part of the graph is removed from the visible context, the positions of the nodes become invalid. When the nodes are then again added to the context they are attached in a best-fit manner to the existing layout. Reusing the old positions would not help, because the position of the current visual context is different. In future research, we plan to develop a layout that produces repeatable structures and paths to these structures.

REFERENCES

D. Archambault and H. C. Purchase, 2013, Mental Map Preservation Helps User Orientation in Dynamic Graphs. *Proceedings of the 20th international Conference on Graph Drawing*, pp. 475–486.

J. Branke, 2001. Dynamic Graph Drawing. *Methods and Models, Drawing Graphs. Lecture Notes in Computer Science(2025)*, pp. 228-246

S. Diehl, C. Görg, A. Kerren, 2001. Preserving the Mental Map using Foresighted Graphlayout. *Proceedings of the Joint Eurographics IEEE TVGC Symposium on Visualization (VisSym)*, Wien, New York, Springer Verlag. pp. 175-184

S. Diehl, C. Görg, 2002. Graphs, They Are Changing, *Revised Papers from the 10th International Symposium on Graph Drawing*, London, UK, pp. 23-31

P. Eades, 1991. *Preserving the Mental Map of a Diagram*. International Institute for Advanced Study of Social Information Science

Y. Frishman, A. Tal, 2004, Dynamic Drawing of Clustered Graphs. *IEEE Symposium on Information Visualization, INFOVIS,* pp.191-198

Y. Frishman, A. Tal, 2008, Online dynamic graph drawing. *IEEE Transactions on Visualization and Computer Graphics, 14*(4), 727-740.

T. Fruchterman, E. Reingold, 1991. Graph Drawing by Force-directed Placement. *Software: Practice and experience,* vol. 21(11), New York, USA, pp. 1129-1164

G. W. Furnas, 1986, Generalized Fisheye Views. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* Boston, USA, pp. 16-23

F. van Ham, A. Perer, 2009. "Search, Show Context, Expand on Demand": Supporting Large Graph Exploration with Degree-of-Interest, *IEEE Transactions on Visualization and Computer Graphics*, vol.15, no.6, pp.953-960.

J. Heer, S.K. Card, 2004. DOITrees revisited: Scalable, Space-constrained Visualization of Hierarchical Data. *Proceedings of the working Conference on Advanced visual interfaces*. New York, USA, pp. 421-424

M. L. Huang, P. Eades, 1998. A Fully Animated Interactive System for Clustering and Navigating Huge Graphs, *Graph Drawing 6th International Symposium, GD' 98,* Montréal, Canada, pp. 374-383

M. L. Huang, P. Eades, J. Wang et al, 1998. Online Animated Graph Drawing using a Modified Spring Algorithm, *Journal of Visual languages and Computing*, vol. 9(6), pp.623-645

T. Kamada, S. Kawai, 1989. An Algorithm for Drawing General Undirected Graphs. *Information Processing Letters*, vol. 31(1), pp. 7-15

J.B. Kruskal, J.B. Seery, 1980. Designing Network Diagrams, *Proceedings of the First General Conference on Social Graphics,* Washington DC, USA, pp.22-50

T. May, M. Steiger, J. Davey and J. Kohlhammer, 2012. Using signposts for navigation in large graphs. *Computer Graphics Forum.* vol.31(3/2), pp. 985-994

K. Misue, P. Eades, W. Lai, K. Sugiyama, 1995. Layout Adjustment and the Mental Map. *Journal of visual languages and computing*, 6(2) pp.183-210

T. von Landesberger, A. Kuijper, T. Schreck et al., 2011. Visual Analysis of Large Graphs: State-of-the-Art and Future Research Challenges. *Computer Graphics Forum* (2011), vol. 30(6), pp. 1719-1749

Y.Y. Lee, C.C. Lin, H.C. Yen, 2006. Mental Map Preserving Graph Drawing using Simulated Annealing. *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualization,* vol. 60, Darlinghurst, Australia, pp. 179-188

N. Osawa, 2001. A Multiple-Focus Graph Browsing Technique using Heat Models and Force-directed Layout. *Proceedings of the Fifth International Conference on Information Visualisation*, pp. 277-283

H. C. Purchase, E. Hoggan, C. Görg, 2007. How Important is the "Mental Map"? – an Empirical Investigation of a Dynamic Graph Layout Algorithm. *Proceedings of the 14th International Conference on Graph Drawing*. Berlin, pp.184-195.

H. C. Purchase and A. Samra, 2008. Extremes are better: Investigating Mental Map Preservation in Dynamic Graphs. *Proceedings of the 5th international Conference on Diagrammatic Representation and Inference*, pp. 60–73.

B. Shneiderman, 1996. The eyes have it: A task by data type taxonomy for information visualizations. *Proceedings of the IEEE Symposium on Visual Languages*, Washington DC, USA, pp.336-343