DOI: 10.1111/cgf.12394
Eurographics Conference on Visualization (EuroVis) 2014
H. Carr, P. Rheingans, and H. Schumann
(Guest Editors)

Volume 33 (2014), Number 3

Semi-Automatic Editing of Graphs with Customized Layouts

S. Gladisch, H. Schumann, M. Ernst, G. Füllen & C. Tominski

University of Rostock, Germany

Abstract

Usually visualization is applied to gain insight into data. Yet consuming the data in form of visual representation is not always enough. Instead, users need to edit the data, preferably through the same means used to visualize them. In this work, we present a semi-automatic approach to visual editing of graphs. The key idea is to use an interactive EditLens *that defines where an edit operation affects an already customized and established graph layout. Locally optimal node positions within the lens and edge routes to connected nodes are calculated according to different criteria. This spares the user much manual work, but still provides sufficient freedom to accommodate application-dependent layout constraints. Our approach utilizes the advantages of multi-touch gestures, and is also compatible with classic mouse and keyboard interaction. Preliminary user tests have been conducted with researchers from bio-informatics who need to manually maintain a slowly, but constantly growing molecular network. As the user feedback indicates, our solution significantly improves the editing procedure applied so far.*

Categories and Subject Descriptors (according to ACM CCS): I.3.4 [Computer Graphics]: Graphics Utilities—Graphics editors

1. Introduction

Visualization has become a widely accepted means to facilitate data intensive work. Many excellent approaches exist to support exploratory or confirmatory analysis as well as the visual communication of analysis results. In these scenarios, the data on the machine are transferred via a visual representation to insight or understanding on the side of the user [vW05].

Often there is also the need to go the opposite direction, meaning that user knowledge needs to be fed back to the data by manipulating or editing them. This can be necessary, for instance, to correct obviously erroneous data entries, to add missing values, or to incorporate new knowledge into an existing database. Hence, there is a need to develop solutions, where methods to visualize data work hand in hand with methods to edit data [Bau06].

As we will see in Section 2, the literature is rich of interaction techniques allowing users to control the visualization. Computer science in general provides powerful means to manipulate data. Yet there are two serious problems. First, manual data manipulation is often time-consuming business and lacks user support [KHP*11]. Second, and more importantly, visualization and editing are usually carried out separately, which hinders a smooth data analysis and data manipulation cycle. Therefore, our work advocates an integrated data visualization and editing process.

We aim to support the visual editing of graphs with established customized layouts. Users working with such graphs over a longer period of time develop a mental map of the data. Care has to be taken to maintain the integrity of the mental map on incremental updates to the graph. Moreover, customized graph layouts often obey application-dependent constraints (e.g, certain nodes may appear only in specific regions). Such constraints are usually incompatible with existing automatic graph layout methods. Hence, users have to resort to diagram editors to carry out manipulations manually, which is cumbersome especially when working with an already complex layout of nodes and edges.

Our approach to editing graphs with customized layouts is *semi-automatic*, combining the strengths of interaction and computation. We address the editing of the graph's structure, more specifically, the insertion of new nodes and edges as well as the update of existing elements to improve the layout. The key idea is to use an interactive lens, the *EditLens*, to define a local region where edit operations are to take effect. Node positions and edge routes of the entities being edited are automatically computed based on different optimization criteria. As we do so within the confines of the *EditLens*, the

computational costs are kept at a manageable level and the mental map is maintained. By moving the lens or switching between optimization criteria, the user can on-the-fly explore different solutions and customize the suggestions made by the lens before an edit operation is finally committed. In Section 3, we introduce our approach in more detail.

An interactive tool for visualizing and editing graphs has been implemented, combining established visualization concepts with the advantages of the EditLens and standard manual editing facilities. As described further in Section 4, we take advantage of novel multi-touch gestures, but still support classic mouse and keyboard interaction. In Section 5, we apply our solution to a concrete problem from bio-informatics where researchers collect and maintain a network of molecular interactions in mouse, the so-called *PluriNetWork* [SHG*10] with several hundred nodes and edges. With our tool, the maintainers of the PluriNetWork can now conduct the necessary steps more easily as can be seen from the user feedback described in Section 6. Discussion and conclusion will be given in Sections 7 and 8.

2. Related Work

We structure the review of related work into two parts: aspects of interaction and algorithmic aspects.

2.1. Aspects of Interaction

Interactive manipulation is a key theme of visualization. While interaction is mostly applied to alter the visualization, there are also approaches to edit the underlying data. Next we take a brief look at both aspects.

Several techniques have been proposed for interacting with graphs in particular. Examples for adjusting graph layouts are interactive sticks and arcs [RRK04], magnet-based attraction [SF08], or radial menus and hotboxes [MJ09]. There are also interactive lenses to reduce node and edge clutter and to create local overviews of subgraphs [TAvHS06]. Interactive adjustment of edge curvature via bundling, fanning, magnets, and legends is addressed in [HRDLC12]. Individually, the aforementioned approaches provide excellent means for interaction with graphs, but editing of graphs, in particular of graphs with customized layouts, has received only little attention.

On the other hand, Baudel [Bau06] argues convincingly for manipulation of the data through means of visual representation and direct interaction. Off-the-shelf graph editors provide the functionality to interactively manipulate graph data represented as node-link diagrams. Yet, much work has to be done manually by the user, with only little or no assistance from the computer. Sketching and multi-touch approaches strive to ease graph editing by employing more natural drawing gestures. Examples are works on sketching of UML diagrams [CGH03] and on beautifying sketches of node-link diagrams on-the-fly [AN06]. Combining overview+detail with constraint-based layout has been demonstrated to be beneficial [DMS*08]. Multi-touch and pen interaction can help to support the editing of graphs [FHD09].

Usefulness and ease-of-use of existing graph editing approaches have been demonstrated for moderately-sized graphs being constructed from scratch. However, the existing approaches fall short of addressing larger graphs with hundreds of nodes and edges forming a customized and established layout. In such scenarios, accuracy and efficient use of screen space become important aspects. Addressing them by manual positioning of nodes and edges is no longer sufficient. Our semi-automatic editing approach tackles these problems by integrating interactive and algorithmic methods. Related work on algorithmic aspects of graph drawing are reviewed next.

2.2. Algorithmic Aspects

In our work, we consider evolving graphs. Although changes do not occur automatically, but are the result of editing operations by the user, there is still a certain analogy to time-varying graphs. Therefore, we shall indicate some related algorithms for these kinds of graphs.

In general, visualization of time-varying graphs follows two approaches: either time steps are visualized individually in sequence or a super graph including all time steps is constructed. Visualizing time steps individually usually means showing small multiples or an animation of the evolving graph (e.g., [CSP*06] or [TA08]). Approaches based on a super graph can optimize the layout according to a holistic view of the graph (e.g., [TS07] or [FWSL12]). None of these approaches can be used in our scenario, because they require a finite set of time steps, which is not given for an open-ended editing procedure.

Such scenarios are dealt with by online graph drawing algorithms. Examples include online drawing of directed acyclic graphs [Nor96], approaches based on Baysian networks [BW97], solutions for clustered graphs and corresponding quality metrics [FT04], or approaches based on simulated annealing [LLY11]. These algorithms aim to achieve stability and coherence between subsequent drawings, which is also needed for editing. However, these algorithms are all fully automatic and do not provide the facilities to adjust the layout according to application-dependent customizations.

In summary, according to our review of related work, editing customized graph layouts remains an under-researched problem to whose solution we contribute.

3. Semi-Automatic Graph Editing

We propose to combine interactive and computing methods to create a semi-automatic approach to graph editing. Before going into detail, we take a look at the problem to be solved.

3.1. Problem Analysis

In general, data manipulation is one of the following operations: insert, update, or delete. In our specific case of editing the structure of graphs, nodes and edges are the entities to be manipulated. Editing data attributes or data values associated with nodes and edges is beyond the scope of this paper and a concern for future investigation.

Editing graphs with an established customized layout requires maintaining the mental map that users have already developed of the data. This is especially true when the layout obeys certain semantic properties, for example, when specific nodes may only appear in certain regions of the layout. Therefore, edit operations must not result in a global change of the graph layout, but the effects must remain local. Otherwise, we would risk that the graph can no longer be recognized as the one that it had been before.

When editing graphs with customized layouts, users should be allowed to input their ideas or constraints of the layout into the system to control the final outcome. Placing single nodes or routing individual edges by hand results in local changes and clearly allows for full customization of the graph layout. However, such manual editing is infeasible when editing larger graphs. Imagine yourself using a diagram editor for inserting a node with say a dozen of edges into an existing graph with hundreds of nodes. Positioning this node and routing each connected edge by hand is tedious work and would take a lot of time, especially when certain quality criteria (e.g., shortest overall length of all connected edges) have to be evaluated mentally.

On the other hand, we could let automatic graph layout algorithms do the math and compute high quality layouts with respect to general quality criteria. But these algorithms are usually incompatible with customized layouts. Furthermore, most graph layout algorithms work globally, which contradicts our need for only local changes to preserve the mental map. A third difficulty is that computational costs rise significantly when considering multiple layout criteria for larger graphs, which hinders interactivity of edit operations.

In summary, fully manual editing techniques are not sufficient for larger graphs with customized layouts due to the immense effort required from the user and fully automatic layout algorithms dismiss custom application-dependent layout constraints and lack mechanisms for controlling the desired editing outcome. Our idea is to combine the power of interaction (yet without intensive labor) with the power of computation (yet with local effect and customizable).

3.2. General Approach

In the first place, we need to think about what tasks are to be interactive and what can be done automatically, and how visual feedback ties interaction and computation together.

User interaction Interaction gives the user control of what to edit (e.g., insert node) and of what the editing outcome will be, including the possibility for customization. However, we do not want to burden the user with placing nodes at exact positions or defining exact routes of edges. Therefore, we relax *point-wise* editing to *region-oriented* editing. Instead of specifying precise points, the user interactively defines a local region where an edit operation is to take effect. This local region acts as a *coarse solution*, and finding a *precise solution* will be the task of automatic computations, effectively reducing the users workload.

Automatic computation Algorithmic calculations enhance the interactive editing by providing suggestions for exact node positions and edge routes. The underlying algorithms consider a set of layout criteria to compute "good" results with respect to the local region as defined by the user. Different heuristic layout strategies leave space for customization via prioritizing specific layout criteria. The computational part relieves the user from evaluating the layout quality mentally and finding optimized exact solutions by hand.

Visual feedback The interplay of interaction and computation also demands suitable visual feedback. Of primary concern is to accentuate elements being considered for an edit operation and to dim those that are not affected. Moreover, the local region acting as a coarse solution plays a special role, while the rest of the graph layout fades into the background. To ensure a smooth editing cycle while interactively exploring the space of "good" solutions, visual feedback about the suggestions of the automatic computations is continuously provided to the user.

To support the aforementioned functionalities, we developed the novel concept of an interactive lens for edit operations, which we describe in detail next.

3.3. The EditLens

Magic lenses as introduced by Bier et al. [BSP*93] naturally lend themselves to our purposes. Magic lenses affect a local region, which is defined by the lens' shape, size, orientation, and position. The actual effect is defined by a lens function. The lens functions known so far mostly alter the visual content within the lens, for instance, to magnify focus regions or to de-clutter overcrowded parts of the display.

We propose the EditLens whose function is to edit the underlying data and the visual representation. According to our research, this is a novel type of lens function, not yet considered in the literature. With the EditLens, we incorporate (1) interactive control by adjusting the lens, (2) locally confined automatic computation of the lens function, and (3) the presentation of corresponding visual feedback in a single editing tool.

The EditLens has a rectangular shape, which is suitable for most edit operations on customized graph layouts. Alternative shapes might be useful for special applications (which

we are still unaware of). The rectangular lens can be resized and repositioned interactively to define the local region where the edit operation is to take effect. By using a larger lens, the user widens the space of possible node positions and edge routes. Using a rather small lens can be useful when the user has a strong idea of a suitable solution in mind or to adhere to application-dependent constraints. As such, the EditLens utilizes the coarse-positioning skills of humans (as any magic lens does [BSF*94]).

Precise positioning is left to exact calculations by the computer. These calculations suggest "good" node positions and edge routes within the lens. The calculations are kept local in order to maintain the integrity of the already existing layout and to keep computational costs low. Naturally, computations with a local scope rarely lead to a global optimum. Therefore, the user is free to explore alternative solutions by adjusting the EditLens or prioritizing a different layout criterion. Eventually, the user may accept a suggested solution and commit the edit operation. Finally, there is still the option to manually refine a suggested solution via standard means of editing altogether.

The EditLens has been designed to support the following edit operations as depicted in Figure 1:

- Insertion of nodes and edges into a graph
- Update of node positions and edge routes of the layout
- Deletion of nodes and edges from a graph

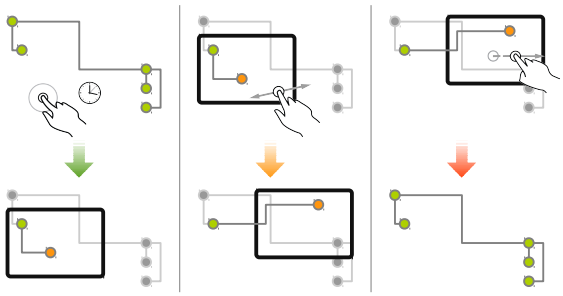
We start with describing the insertion of a node, as it best illustrates how the EditLens works in general. Additional details for the remaining operations are given later on.

3.3.1. Inserting a node

Once node insertion has been triggered for the local region defined by the EditLens, two computational steps need to be carried out. First, the node position must be computed and based on that, as the second step, the edges connected to the node must be routed. For this, aesthetics and quality criteria have to be considered [BETT98]. General goals are to avoid clutter and ambiguities, which usually means utilizing the available screen space efficiently.

Since it is impossible to handle all possible criteria simultaneously [Pur02], we resort to a plausible set of four criteria. In the first place, nodes should be easily discernible, which can be achieved by leaving sufficient space to other nodes. Second, neighbors should be easily detectable, which demands edges to be short. Third, edges should circumvent existing nodes, to prevent misinterpretation of connectivity. A well-accepted way to achieve this is to use orthogonal edge routing. Finally, edges should be easily traceable, which is the case for edges with as few bends as possible. This set of criteria is certainly not exhaustive, but we assume that alternatives can be taken into account on demand.

For finding a "good" position p_v for the node v being inserted we follow a three step procedure:



(a) Insert (b) Update (c) Delete

Figure 1: *Schematic depiction of insert, update, and deletion with the EditLens. The node being edited is highlighted in orange, regular nodes are shown in green. Nodes that are not considered for the editing operation are dimmed.*

1. Determine a set $\mathfrak{R}$ of areas that are potential candidates for node placement. The areas must be within the EditLens and must be free of existing nodes or edges.
2. Select a suitable area $R \in \mathfrak{R}$ within which p_v is to be computed. Necessary preconditions are that R be large enough to contain the node's shape and that application-dependent semantic regions be obeyed.
3. Compute a "good" position p_v within R taking into account the layout criteria indicated before.

A sufficiently fast method to determine $\mathfrak{R}$ is the *dynamic space management* by Bell and Feiner [BF00]. Their method is based on *full* and *empty space rectangles*, which describe occupied and empty space, respectively. The dynamic space management is initialized by creating full space rectangles for the bounding boxes of all existing nodes and edge segments. Note that the bounding boxes are enlarged by a certain δ to preserve a minimal gap between existing and newly placed graph elements. The dynamic space management can then be queried for *empty space rectangles* inside the EditLens to compile the set $\mathfrak{R}$ as illustrated in Figure 2.

The next steps are to select an $R \in \mathfrak{R}$ and to compute the exact position p_v in R. For these steps, we have to consider the necessary conditions formulated in step (2) and the sufficient conditions as imposed by the addressed layout criteria. However, it is hard or even not possible to obtain node positions and edge routes that fulfill all criteria optimally. For example, orthogonal edge routing depends on the node position. But the node position for the optimal orthogonal edge route could be too close or even overlap with an existing node. To support the search for good positions, the EditLens can prioritize certain criteria. For this purpose, we have developed three placement strategies:

Node space first A node is easily discernible when there is enough free space around it. Following this thinking, we select R as the largest empty space rectangle available that

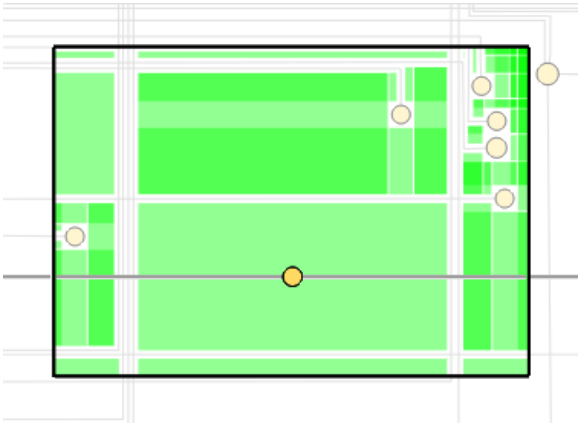


Figure 2: *Empty space rectangles (green) under the EditLens (dark frame) with the node being edited (yellow). Different shades of green indicate where rectangles overlap.*

falls into the semantic region associated with the node being inserted. Then we compute p_v as the center of R. Figure 2 illustrates the available empty space rectangles, with the largest one selected and the node positioned in its center. After placing the inserted node this way, the edge routing is invoked to establish links to the node's neighbors.

Edge length first To easily identify neighbors in a graph, it makes sense to prioritize short edge lengths. According to this thinking, the sum of the lengths of the edges connected to the node being inserted should be minimized. As this sum depends on the node's position, we have to find the position within the EditLens where the sum of edge lengths is minimal. However, the computation costs for edge routing prevent testing every possible position within the lens. Therefore, we developed the following heuristic.

First, select R as the empty space rectangle that falls into the semantic region associated with the node being inserted and whose center c_R has the minimal sum of Manhattan-distances between c_R and every neighbor of v. Then we define a linear optimization problem. Let (x,y) be a position in R, and $(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)$ be the positions of the n neighbors of v. Further, (l,t) denotes the top-left corner of R and (r,b) is R's bottom-right corner, and d_v is the diameter of the shape of v. Our goal is to optimize:

$$f(x,y) \;=\; \sum_{i=1}^{n} |x-x_i| + |y-y_i| \;\rightarrow\; min$$

such that

$$\begin{aligned} x &> l + d_v/2 \\ x &< r - d_v/2 \\ y &> t + d_v/2 \\ y &< b - d_v/2 \end{aligned}$$

By solving this linear optimization problem with the simplex algorithm, we derive the optimal position $p_v = (x,y)$ for v in R. This heuristic, however, does not account for obstacles along the shortest routes (according to Manhattan distance) to neighbors of v. The exact routes around obstacles are finally computed by an orthogonal edge router.

Number of edge bends first When tracing edges, bends along the way can be disruptive. Therefore, it makes sense to keep the number of edge bends to a minimum. Again the position at which edge routes have a minimal number of bends could be determined by invoking the edge routing for all possible positions and selecting the best one. As this is prohibitively expensive, we came up with a simple heuristic to prioritize the reduction of edge bends.

A key observation is that if the node v being inserted is aligned horizontally or vertically with one of its neighbors u, the edge (v,u) does not bend. Further, if two neighbors u_1 and u_2 are already on a horizontal or vertical line, inserting v on that line in between u_1 and u_2 generates edges (v,u_1) and (v,u_2) that do not bend either. Therefore, our heuristic searches exactly for such situations. In a first step, we determine existing straight horizontal and vertical lines between all neighbors of v. These lines are intersected with all empty space rectangles in $\mathfrak{R}$. Then we select an $R \in \mathfrak{R}$ as follows.

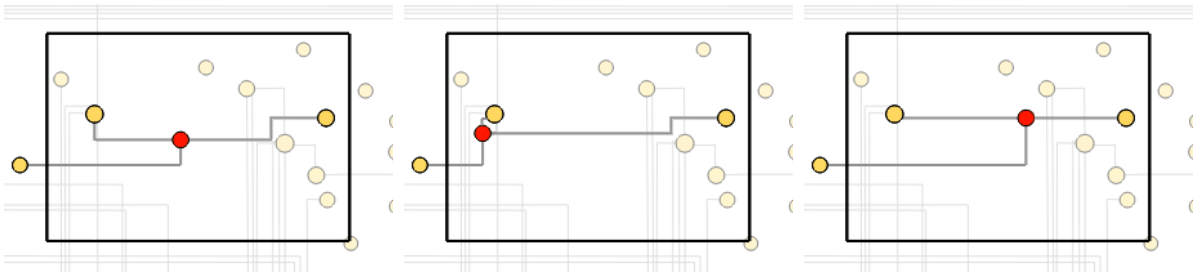
If an empty space rectangle is intersected by both a horizontal line and a vertical line, select that rectangle as R, and the intersection of both lines will be the position p_v of the node v. If no such rectangle exists, we look for a rectangle R that is intersected by at least one horizontal or one vertical line, and place p_v on that line and centered in R. If we encounter multiple possible solutions during our search, we can either favor larger rectangles or shorter intersection lines to drive the final decision for R and p_v.

Figure 3 compares the results of the three placement strategies. These strategies together resizing and relocating the EditLens offer sufficient room for controlling and customizing the outcome of the node insertion. At the same time, the automatic calculations reduce the user's mental load and manual work significantly.

If no suitable node position can be found by the automatic means due to insufficient space or due to unsatisfiable constraints (e.g., EditLens placed in wrong semantic region), the color of the lens border changes to a signaling color such as red. The user can then either adjust the lens parameters or place the node manually, overriding any quality criteria.

3.3.2. Inserting an edge

When inserting an edge (u,v) the idea is to use the EditLens as a constraint for the edge routing, more precisely as a region through which the edge has to be routed automatically. Consider two nodes u and v of the same type (i.e., both nodes are placed in the same dedicated semantic region) that have



(a) Node space first (b) Edge length first (c) Number of edge bends first

Figure 3: *Three placement strategies determine a node's position based on prioritizing different layout criteria, including favoring much free space around nodes, shortening edges to neighbors, and reducing the number of edge bends. The inserted node is shown in red, its neighbors are yellow, unaffected elements are dimmed.*

to be connected by an edge. It makes sense that the route of this edge be within the confines of the dedicated region. However, when using a routing algorithm that is unaware of the dedicated region and simply calculates a route from the source u to the target node v, one cannot guarantee that the result satisfies this specific constraint.

With the EditLens, the user first defines the local area through which the edge has to be routed. Second, the EditLens automatically calculates a single point p in the local region through which the edge has to be routed precisely. For determining p, we again use the aforementioned placement strategies. Third, the automatic edge routing algorithm calculates a route from the source node u to p and from p to the target node v. This guarantees that the route will connect u and v passing through p. As edge routing algorithms usually aim to minimize edge length, it is also almost certain that the edge overall remains within the bounding box of u, v, and p.

3.3.3. Deleting and updating nodes and edges

Delete operations are straight-forward. When a node is deleted, all connected edges are deleted as well. If the deleted node was an obstacle for other edges, the edge routes of these edges can now be improved by update operations. When deleting an edge, nodes that have been placed near the route of that edge might need further layout improvement to better utilize the screen space made available.

Updating a node or an edge can be reduced to virtual delete and insert operations in the following way. When updating an existing node, delete this node first and then re-insert it to a new position. When updating an edge in the diagram, delete this edge first and the re-insert it with a new edge route.

3.4. Approach Summary

The EditLens supports insertion, update, delete operations on graphs with customized layouts. By interacting with the lens, users can explore different placement or routing suggestions for different regions and different layout strategies. We see this as a key advantage of the EditLens. Such an exploration of solutions is not possible with either manual editing techniques or automatic layout algorithms alone.

Yet, as the EditLens computes heuristic-driven suggestions only, minor refinements might be necessary to fully satisfy the user. Such refinements of node positions or edge routes can be done using classic means of editing (e.g., dragging a node or the bends of an edge to slightly adjusted positions). As the coarse solution is already "good", the effort for manual refinement will remain at a reasonable level.

4. Implementation and Interaction

We implemented a prototype that combines classic means for exploring graphs with our novel techniques for editing graphs. The graph is shown as a node-link representation. Nodes are represented as circles whose size and color are set according to data attributes. Textual labels for nodes are placed via a dedicated labeling algorithm [LSC08]. Edges are represented by orthogonal polylines as computed by the edge routing algorithm. The node-link diagram is embedded into a zoomable space [Bed11].

While the visualization part follows well-known ideas, the editing part brings in our new ideas. We implemented a dual-interaction prototype to take advantage of the efficiency of familiar mouse+keyboard interaction and the potential of novel multi-touch gestures. This gives users the chance to choose the modality they deem best for the task at hand.

Standard GUI elements such as buttons, scroll bars, and menus are operated identically with mouse+keyboard and touch interaction. Interaction differs, however, when editing in the zoomable node-link diagram. Our design has been inspired by previous work on touch interaction for graphs [FHD09, SNDC10]. Figure 4 summarizes the gestures we employ. Next we indicate how users actually carry out navigation and selection as well as manipulation tasks.

Figure 4: *Touch gestures used in the EditLens prototype.*

Navigation and Selection Pan-navigation works via mouse drag in empty space or by two finger touch-slide in empty space. Zoom-navigation is activated via the mouse wheel or by using a pinch gesture, which is very common for this task. Individual nodes or edges can be either selected by single mouse clicks or by a single tap with a finger. Selected elements are highlighted and detailed textual information about them is shown. For selecting multiple graph elements, the user can also use a lasso-selection gesture.

Manipulation The lens is our main interaction element. It can be spawned for node insertion with a long tap or press in empty space or for node editing with a long tap or press on an existing node. The lens can be enlarged in x-direction, y-direction, or both combined by using pinch gestures. Resizing is also possible via the keyboard or the mouse. Translation of the lens is done via single finger slide or regular mouse drag. To move the lens over large distances, the user can use a double tap or click as a shortcut, which triggers a smooth transition of the lens to the desired location. Adjusting the lens initiates an automatic re-computation of the suggested position and routes of the elements being edited. Users can accept the placement suggestion by a single finger tap or mouse click. Elements can be deleted using a flick gesture or by pressing the DEL key.

As fall-back solutions, we also support fully manual editing of node positions and edge bends. This is done through classic drag and drop gestures.

With these interaction techniques and the underlying algorithmic solutions for node positioning and edge routing, our prototype is capable of dealing with graphs with several hundreds of nodes and edges. Next we describe an application of our solution to a real-world problem from bio-informatics.

5. Application Example: Editing the *PluriNetWork*

A use case for our approach is the manual maintenance of networks discovered in complex systems. An example is the *PluriNetWork* [SHG*10], a literature-curated network of 365 nodes and 631 edges, where nodes represent genes and edges describe molecular interactions that are important for the cellular state of pluripotency in mouse. Bio-informatics scientists have developed the network from scratch using the editor of the Cytoscape platform [SMO*03]. The established layout follows a circuit-like design motivated by the analogy of the flow of gene regulation and the flow of electricity. As illustrated in Figure 5, the layout is divided into three distinct regions of semantically related nodes: signaling on the top (blue), epigenetics on the left (green), and transcriptional gene regulation in the center (orange). Once new verified knowledge becomes available in the literature, the network is edited manually. This manual editing is still carried out using the Cytoscape editor. However, with the increased size and complexity of the PluriNetWork, preserving the already existing semantic map is a major difficulty. A description of the original editing procedure makes clear why this is so.

Original editing procedure Given a newly reported interaction between two genes $(gene_1, gene_2)$ as input, the editing workflow is as follows. First, it has to be determined if $gene_1$ and $gene_2$ are already present in the graph. If either of the genes does not exist, the curator has to find a good spot for placing a node for the missing gene according to its associated semantic region. If one gene already exists, the new node should be as close as possible to the existing node, but node overlap has to be avoided.

Then the curator heuristically determines a path from $gene_1$ to $gene_2$ that (1) is short, (2) has a low number of bends, and (3) is as far away as possible from existing edges. Finding such a path could turn into a major challenge, as the optimal solution satisfying the three listed criteria is very often not at all obvious or even does not exist. Often, a suboptimal solution was attempted, yielding a displeasing layout and triggering further follow-up adjustments of nodes and edges that were not affected in the first place.

If both genes exist already, and only their interaction is new, the genes are usually already close in the network layout, because the layout reflects biological relatedness, and new interactions tend to connect related genes. In some cases, however, the nodes of $gene_1$ and/or $gene_2$ need to be moved to accommodate a visually more pleasing layout according to the mentioned criteria. Manually finding a path between $gene_1$ and $gene_2$ would then pose the same difficulties as described before. Further complications arise if a node in question has so many edges that there is insufficient space to start a new one. In such cases, existing edges are rearranged, and if needed, the node is enlarged to be able to attach a new edge to it.

The bio-informatics researchers transferred the graph (connectivity as well as layout) to the *WikiPathways* platform [KvIH*12] to enable community contributions. However, the built-in editor of WikiPathways turned out to be very cumbersome to use, effectively inhibiting any public editing of the PluriNetWork.

Applying the EditLens To help the curators of the PluriNetWork in editing their data, we integrated the EditLens into the original editing workflow. In the first part, nothing changes. The first step is to classify a newly reported interaction of two genes $(gene_1, gene_2)$ according to whether none, one, or both of the genes already exist in the layout. If none of the genes exist we first apply the EditLens to insert $gene_1$ (operation: insert node). Then $gene_2$ with the connection to $gene_1$ is inserted (operation: insert node and edge).

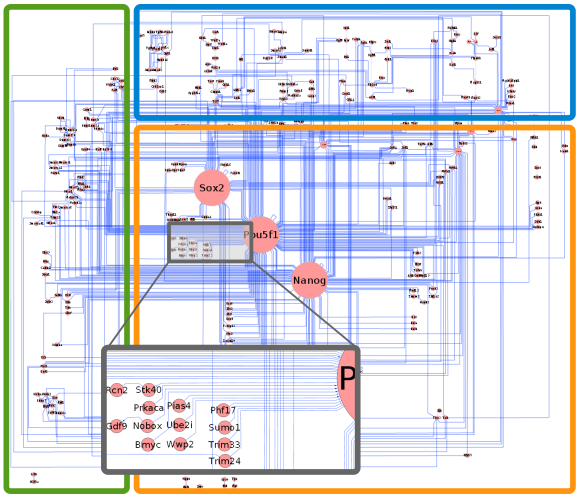


Figure 5: *The PluriNetWork. Marked are the three semantic regions (blue, green, and orange). The detail view (gray) indicates how cumbersome manual editing must be.*

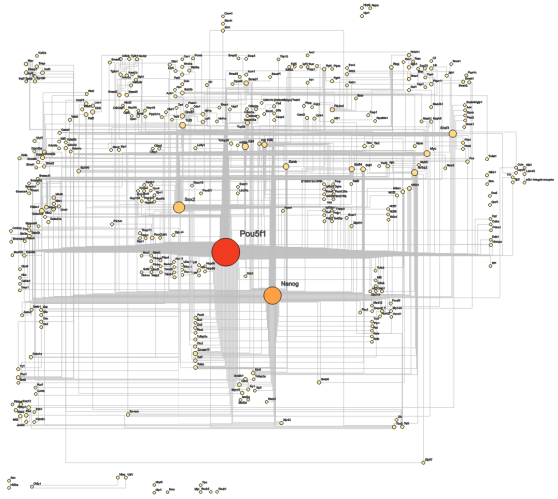


Figure 6: *The improved PluriNetWork as visualized by our prototype. Node size and color visualize node degree. Labels were placed using a dedicated labeling algorithm.*

The latter operation also applies if only one gene is already present in the layout. If both genes already exist, we use the EditLens to insert a new edge between $gene_1$ and $gene_2$ (operation: insert edge).

When genes need to be repositioned or gene interactions need to be rearranged for layout improvement in general, the EditLens' update functionality is employed. Deletion of genes or interactions among them is not applicable in the PluriNetWork as all existing elements are backed by scientific literature. In the unlikely event that a scientific discovery should be refuted, the EditLens would support carrying out the necessary correction in the PluriNetWork.

We expected that integrating the EditLens into the PluriNetWork's editing workflow will significantly help the maintainers of the network in carrying out their editing tasks. A small user evaluation has been set up to verify this.

6. Preliminary User Feedback

For the evaluation, the aim was to gather feedback regarding the potential usefulness of the EditLens. Prior to the user evaluation, a visualization specialist was recruited for pilot testing. Then we performed a small qualitative study using a flexible interview format.

Participants Our primary interest was to receive feedback from domain experts who edit networks on a regular basis. Therefore, we invited two of the researchers maintaining the PluriNetWork. In order to contrast expert feedback against feedback from more casual users, we recruited two additional non-experts, who had never worked with the PluriNetWork before. All four participants were male and employees or students at a university. They all were familiar with using interactive software, but none had used our EditLens prior to the study. The participants judged themselves as experienced in using touch gestures.

Tasks and devices For the first part of the study, the participants' task was to familiarize themselves with our prototype, especially with navigation techniques, manual editing techniques, and with the EditLens. The experimenter first demonstrated the available functionality and then the participants were prompted to try them out on their own. The network presented throughout the study was an updated and improved PluriNetWork as depicted in Figure 6. The introduction lasted approximately 15 minutes.

In the second part, the participants had to solve three tasks on the PluriNetWork. The first task was to insert a new node along with two edges to existing nodes. For the second task, the participants were asked to insert an edge between two already existing nodes. Finally, the third task was to improve the network layout by updating an existing node. All three tasks were motivated by real-world tasks from the context of the PluriNetWork.

The participants had to carry out all tasks using both the original manual editing procedure and our novel EditLens. Manual editing was done with mouse+keyboard interaction, whereas the EditLens was operable via touch-gestures. Two participants (one expert and one non-expert) used the EditLens first, the other two participants used the manual editing procedure first. On every task, the participants were instructed to "think aloud" meaning that they should give any feedback they have in mind. The second part of the study took between 30 and 40 minutes.

The study was performed using a regular desktop computer with a 23" touch-enabled monitor, which was arranged in a horizontal position. A reference sheet with the available multi-touch and mouse gestures as well as key mappings was presented on another monitor so that the participants could look them up quickly when needed.

Results All participants were able to finish all tasks using the original editing procedure and the EditLens. Questions regarding the usefulness of the EditLens were answered positively by all participants. The different layout strategies were also positively received by all participants. Ease-of-use of the EditLens was generally acknowledged. When asked which technique they would prefer, both experts and both non-experts answered in favor of the EditLens, as we expected. One expert said that "the EditLens is very useful and can clearly reduce the editing effort". A non-expert user said that "the automatic suggestion of node positions and edge routes is obviously beneficial".

Additional comments suggest that there is space for improving our approach. One participant of the study suggested routing inserted edges along existing edges to create bundles of edges. Another user would have liked to experiment with alternative lens shapes.

Note that our study did not formally control all influencing factors (e.g., mouse+keyboard vs. multi-touch, horizontal vs. vertical display, simple vs. complex editing tasks). Therefore, the results obtained are to be understood as qualitative indicators to be confirmed in further formal studies.

7. Discussion

This section is to discuss the EditLens as one piece of the larger editing puzzle and to indicate limitations and open questions yet to be addressed.

An important point for discussion is the scalability of the EditLens, in terms of both computation and interaction. Our solution works well with the hundreds of nodes and edges of the PluriNetWork. As the computational costs are usually bounded by the size of the EditLens and the limited number of entities affected by an edit operation, we expect that even larger graphs are possible to work with. Critical are update operations that affect high-degree nodes with many edges. Elaborate connector routing algorithms (e.g., [MHT93] or [WMS09]) are required to cope with such cases.

In terms of interaction, we have focused on edit operations for single graph elements. Scaling to multi-element edit operations is a sensible next step. Such operations could work on groups of elements, where certainly additional constraints would need to be introduced, for example, to maintain intra-group ordering and alignment.

An interesting question to be investigated is the long-term effect of locally optimal edit operations. It is yet to be observed, if the global layout quality deteriorates after many local edit operations. The PluriNetWork application would be an ideal candidate to further study this behavior.

The EditLens has been designed with orthogonal graph layouts in mind. To overcome this limitation and to be applicable more broadly, alternative aesthetic criteria (e.g., uniform edge-length, edges in bundles) and corresponding algorithmic solutions need to be considered. One can also imagine lenses that automatically apply different constraints to different regions of a graph layout according to available meta information or data characteristics. Investigating non-rectangular or even adaptive lens shapes is another interesting direction for future work.

Balancing all influencing factors will be a formidable challenge, which we think requires collaboration of interaction, visualization, and algorithm experts.

8. Conclusion

In this work, we presented a semi-automatic approach to editing graphs with customized layouts. Our novel EditLens combines interactive means with automatic computation. In contrast to existing point-wise editing, the EditLens follows a region-oriented paradigm. This significantly eases editing operations, because the user only needs to define a coarse region interactively, rather than a precise position. The algorithmic part of the EditLens computes suggestions for precise solutions, which the user can customize on-the-fly by adjusting the lens or selecting different heuristic placement strategies. As far as we know, no such lens has been considered in the literature.

A prototype implementation of our approach has been applied to a real-world problem in the context of manually curating a larger biomedical network. In a small user evaluation we tested the utility and usefulness of the EditLens. Overall our approach received quite positive feedback from domain experts as well as casual users. We are convinced that the EditLens is a valuable addition to the existing tools for editing data through interactive visual means.

References

[AN06] ARVO J., NOVINS K.: Fluid Sketching of Directed Graphs. In *Proc. of Austr. User Interface Conf. (AUIC)* (2006), Australian Computer Society, pp. 81–86. 2

[Bau06] BAUDEL T.: From Information Visualization to Direct Manipulation: Extending a Generic Visualization Framework for the Interactive Editing of Large Datasets. In *Proc. of ACM Symp. User Interface Softwa. Techn. (UIST)* (2006), ACM Press, pp. 67–76. doi:10.1145/1166253.1166265. 1, 2

[Bed11] BEDERSON B. B.: The Promise of Zoomable User Interfaces. *Behaviour & Information Technology 30*, 6 (2011), 853–866. doi:10.1080/0144929X.2011.586724. 6

[BETT98] BATTISTA G. D., EADES P., TAMASSIA R., TOLLIS I. G.: *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1998. 4

[BF00] BELL B. A., FEINER S. K.: Dynamic Space Management for User Interfaces. In *Proc. of ACM Symp. User Interface Softwa. Techn. (UIST)* (2000), ACM Press, pp. 239–248. doi:10.1145/354401.354790. 4

[BSF*94] BIER E. A., STONE M. C., FISHKIN K. P., BUXTON W., BAUDEL T.: A Taxonomy of See-Through Tools. In *Proc. of SIGCHI Conf. Hum. Fact. Comput. Syst. (CHI)* (1994), ACM Press, pp. 358–364. doi:10.1145/191666.191786. 4

[BSP*93] BIER E. A., STONE M. C., PIER K., BUXTON W., DEROSE T. D.: Toolglass and Magic Lenses: The See-Through Interface. In *Proc. of Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)* (1993), ACM Press, pp. 73–80. doi:10.1145/166117.166126. 3

[BW97] BRANDES U., WAGNER D.: A Bayesian Paradigm for Dynamic Graph Layout. In *Proc. of Intl. Symp. Graph Drawing (GD)*. Springer, 1997, pp. 236–247. doi:10.1007/3-540-63938-1_66. 2

[CGH03] CHEN Q., GRUNDY J., HOSKING J.: An E-Whiteboard Application to Support Early Design-Stage Sketching of UML Diagrams. In *Proc. of Symp. Human Centric Comput. Lang. Envir. (HCC)* (2003), IEEE Comput. Soc., pp. 219–226. doi:10.1109/HCC.2003.1260232. 2

[CSP*06] CARD S. K., SUH B., PENDLETON B. A., HEER B., BODNAR J. W.: Time Tree: Exploring Time Changing Hierarchies. In *Proc. of IEEE Symp. Visual Analytics Scien. Techn. (VAST)* (2006), IEEE Comput. Soc., pp. 3–10. doi:10.1109/VAST.2006.261450. 2

[DMS*08] DWYER T., MARRIOTT K., SCHREIBER F., STUCKEY P., WOODWARD M., WYBROW M.: Exploration of Networks Using Overview+Detail with Constraint-Based Cooperative Layout. *IEEE Trans. Vis. Comput. Graph. 14*, 6 (2008), 1293–1300. doi:10.1109/TVCG.2008.130. 2

[FHD09] FRISCH M., HEYDEKORN J., DACHSELT R.: Investigating Multi-Touch and Pen Gestures for Diagram Editing on Interactive Surfaces. In *Proc. of Intl. Conf. Interactive Tabletops Surfaces (ITS)* (2009), ACM Press, pp. 149–156. doi:10.1145/1731903.1731933. 2, 6

[FT04] FRISHMAN Y., TAL A.: Dynamic Drawing of Clustered Graphs. In *Proc. of IEEE Symp. Information Visualization (InfoVis)* (2004), IEEE Comput. Soc., pp. 191–198. doi:10.1109/INFOVIS.2004.18. 2

[FWSL12] FENG K.-C., WANG C., SHEN H.-W., LEE T.-Y.: Coherent Time-Varying Graph Drawing with Multifocus+Context Interaction. *IEEE Trans. Vis. Comput. Graph. 18*, 8 (2012), 1330–1342. doi:10.1109/TVCG.2011.128. 2

[HRDLC12] HENRY RICHE N., DWYER T., LEE B., CARPENDALE S.: Exploring the Design Space of Interactive Link Curvature in Network Diagrams. In *Proc. of Intl. Conf. Advanced Visual Interf. (AVI)* (2012), ACM Press, pp. 506–513. doi:10.1145/2254556.2254652. 2

[KHP*11] KANDEL S., HEER J., PLAISANT C., KENNEDY J., VAN HAM F., RICHE N. H., WEAVER C., LEE B., BRODBECK D., BUONO P.: Research Directions in Data Wrangling: Visualizations and Transformationsfor Usable and Credible Data. *Information Visualization 10*, 4 (2011), 271–288. doi:10.1177/1473871611415994. 1

[KvIH*12] KELDER T., VAN IERSEL M. P., HANSPERS K., KUTMON M., CONKLIN B. R., EVELO C. T. A., PICO A. R.: WikiPathways: Building Research Communities on Biological Pathways. *Nucleic Acids Research 40*, D1 (2012), 1301–1307. doi:10.1093/nar/gkr1074. 7

[LLY11] LIN C.-C., LEE Y.-Y., YEN H.-C.: Mental Map Preserving Graph Drawing Using Simulated Annealing. *Information Sciences 181*, 19 (2011), 4253–4272. doi:10.1016/j.ins.2011.06.005. 2

[LSC08] LUBOSCHIK M., SCHUMANN H., CORDS H.: Particle-Based Labeling: Fast Point-Feature Labeling Without Obscuring Other Visual Features. *IEEE Trans. Vis. Comput. Graph. 14*, 6 (2008), 1237–1244. doi:10.1109/TVCG.2008.152. 6

[MHT93] MIRIYALA K., HORNICK S., TAMASSIA R.: An Incremental Approach to Aesthetic Graph Layout. In *Proc. of Intl. Worksh. Computed-Aided Software Engineering (CASE)* (1993), IEEE Comput. Soc., pp. 297–308. doi:10.1109/CASE.1993.634832. 9

[MJ09] MCGUFFIN M. J., JURISICA I.: Interaction Techniques for Selecting and Manipulating Subgraphs in Network Visualizations. *IEEE Trans. Vis. Comput. Graph. 15*, 6 (2009), 937–944. doi:10.1109/TVCG.2009.151. 2

[Nor96] NORTH S. C.: Incremental Layout in DynaDAG. In *Proc. of Intl. Symp. Graph Drawing (GD)*. Springer, 1996, pp. 409–418. doi:10.1007/BFb0021824. 2

[Pur02] PURCHASE H. C.: Metrics for Graph Drawing Aesthetics. *Journal of Visual Languages & Computing 13*, 5 (2002), 501–516. doi:10.1006/jvlc.2002.0232. 4

[RRK04] RAISAMO J., RAISAMO R., KARKKAINNEN P.: A Method for Interactive Graph Manipulation. In *Proc. of Intl. Conf. Inform. Visualis. (IV)* (2004), IEEE Comput. Soc., pp. 581–587. doi:10.1109/IV.2004.1320202. 2

[SF08] SPRITZER A. S., FREITAS C. M. D. S.: A Physics-Based Approach for Interactive Manipulation of Graph Visualizations. In *Proc. of Intl. Conf. Advanced Visual Interf. (AVI)* (2008), ACM Press, pp. 271–278. doi:10.1145/1385569.1385613. 2

[SHG*10] SOM A., HARDER C., GREBER B., SIATKOWSKI M., PAUDEL Y., WARSOW G., CAP C., SCHÖLER H., FUELLEN G.: The PluriNetWork: An Electronic Representation of the Network Underlying Pluripotency in Mouse, and Its Applications. *PLoS One 5*, 12 (2010). doi:10.1371/journal.pone.0015165. 2, 7

[SMO*03] SHANNON P., MARKIEL A., OZIER O., BALIGA N. S., WANG J. T., RAMAGE D., AMIN N., SCHWIKOWSKI B., IDEKER T.: Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks. *Gen. Res. 13*, 11 (2003), 2498–2504. doi:10.1101/gr.1239303. 7

[SNDC10] SCHMIDT S., NACENTA M. A., DACHSELT R., CARPENDALE M. S. T.: A Set of Multi-Touch Graph Interaction Techniques. In *Proc. of Intl. Conf. Interactive Tabletops Surfaces (ITS)* (2010), ACM Press, pp. 113–116. doi:10.1145/1936652.1936673. 6

[TA08] TELEA A., AUBER D.: Code Flows: Visualizing Structural Evolution of Source Code. *Computer Graphics Forum 27*, 3 (2008), 831–838. doi:10.1111/j.1467-8659.2008.01214.x. 2

[TAvHS06] TOMINSKI C., ABELLO J., VAN HAM F., SCHUMANN H.: Fisheye Tree Views and Lenses for Graph Visualization. In *Proc. of Intl. Conf. Inform. Visualis. (IV)* (2006), IEEE Comput. Soc., pp. 17–24. doi:10.1109/IV.2006.54. 2

[TS07] TU Y., SHEN H.-W.: Visualizing Changes of Hierarchical Data Using Treemaps. *IEEE Trans. Vis. Comput. Graph. 13*, 6 (2007), 1286–1293. doi:10.1109/TVCG.2007.70529. 2

[vW05] VAN WIJK J. J.: The Value of Visualization. In *Proc. of IEEE Visualization Conf. (Vis)* (2005), IEEE Comput. Soc., pp. 79–86. doi:10.1109/VIS.2005.102. 1

[WMS09] WYBROW M., MARRIOTT K., STUCKEY P.: Orthogonal Connector Routing. In *Proc. of Intl. Symp. Graph Drawing (GD)*. Springer, 2009, pp. 219–231. doi:10.1007/978-3-642-11805-0_22. 9