Topology-Aware Navigation in Large Networks

Tomer Moscovich[1] **Fanny Chevalier**[1] **Nathalie Henry**[2] **Emmanuel Pietriga**[2,3] **Jean-Daniel Fekete**[2]
tomer@lri.fr chevalie@lri.fr nhenry@lri.fr pietriga@lri.fr fekete@lri.fr

[1]Microsoft Research-INRIA Joint Centre, Orsay, France [2]INRIA, Orsay, France [3]LRI - Univ. Paris-Sud & CNRS, Orsay, France

ABSTRACT

Applications supporting navigation in large networks are used every days by millions of people. They include road map navigators, ight route visualization systems, and network visualization systems using node-link diagrams. These applications currently provide generic interaction methods for navigation: pan-and-zoom and sometimes bird's eye views.

This article explores the idea of exploiting the connection information provided by the network to help navigate these large spaces. We visually augment two traditional navigation methods, and develop two special-purpose techniques. The first new technique, called "Link Sliding", provides guided panning when continuously dragging along a visible link. The second technique, called "Bring & Go", brings adjacent nodes nearby when pointing to a node. We compare the performance of these techniques in both an adjacency exploration task and a node revisiting task. This comparison illustrates the various advantages of content-aware network navigation techniques. A significant speed advantage is found for the Bring & Go technique over other methods.

ACM Classi cation Keywords
H. Information Systems H.5 Information Interfaces and Presentation H.5.2 User Interfaces (H.1.2, I.3.6)

Author Keywords
Interaction techniques, content-aware, graph visualization, document navigation

INTRODUCTION

Applications supporting navigation in large networks are used every day by millions of people. They include road map navigators such as Google maps [10], ight route visualization systems such as Delta Air Line Route Map [6] and network visualization systems using node-link diagrams [1, 13]. These applications currently provide generic interaction techniques for navigation: pan-and-zoom and sometimes bird's eye views.

However, for large networks, some important tasks related to the network's topology are not efficiently supported by existing techniques. For example, using Google maps, exploring a long route often involves panning over long portions of a highway with no exits. Zooming out or using a bird's eye view is possible, but some highway exits are difficult to distinguish from roads passing over or under the highway, so an exit can be missed. The same problem arises in network visualization systems where nodes are connected by links that can be long and cross many other links. Following a specific link can take a long time without zooming out, but zooming out makes it difficult to trace a link when other links cross it at a shallow angle.

The problem of panning and zooming, or using a bird's eye view, becomes even more difficult when using a small screen to view a network. On a PDA or smart phone, the input device may not have dedicated zooming controls, and only a small footprint is available for panning gestures, making panning over a long route slow and tedious.

In this article we explore several techniques to improve navigation in such network-related scenarios by using topological information in addition to geometric information. Two techniques are simply visual enhancements of common spatial navigation methods, while two are novel approaches that test different trade-offs between topological and spatial navigation. The first, Link Sliding, allows users to slide along a link to its destination, while the other, Bring & Go, brings all possible destinations within the users view, and automatically transports the user to the selected point.

We begin by introducing the Link Sliding and Bring & Go techniques. We then present a controlled experiment that compares them with visually augmented pan-and-zoom and bird's eye view navigation for three fundamental navigation tasks. We finally discuss implications to the design of systems for large-network visualization.

RELATED WORK

In their survey on Graph Visualization and Navigation, Herman et al. [14] cite four methods for navigating large networks: pan-and-zoom, space distortion techniques such as fisheye views, topological methods such as Furnas's "Generalized Fisheye Views" [8] and layout techniques to dynamically change the layout of the network according to the user's navigation.

Scrolling, Panning and Zooming

Scrolling consists of using a widget, such as a scroll-bar, to control the viewport. Panning uses direct manipulation of the viewport, usually coupled with zooming. A lot of work has been dedicated to improving navigation using scrolling, panning, and zooming, particularly in facilitating navigation in very large spaces, or the navigation to off-screen targets.

Navigation in Large Spaces

When the space is large compared to the viewport size—say more than ten times the size—navigation can take a substantial amount of time, particularly for exploratory tasks. Early on, scroll-bars were proposed as a means for traveling through large documents. However, they suffer from several limitations. They show only the size of the viewport relative to the size of the document, and give no information regarding off-screen content. When the size of the document is large compared to the viewport—say 1000 times the size—moving the thumb of the scroll-bar can produce jumps, and some positions may be unreachable. Therefore, scrolling alone is insufficient for navigating large spaces, hence the prevalence of the pan-and-zoom navigation method

Zooming and panning has been used since the beginning of interactive computer graphics to navigate in maps and other graphic scenes. Perlin introduced zoomable user interfaces in [24], while Furnas and Bederson have formalized the concept of space-scale diagrams to reason about these zoomable spaces.

Navigation involves not only viewing the space, but also moving the viewport. Several researchers have worked to optimize the coupling of zoom and pan to allow faster and more accurate navigation in zoomable interfaces [17, 11, 32, 27] with no assumptions regarding the contents of these spaces. Ishak and Feiner have proposed *Content-aware Scrolling* [19] to optimize navigation when paths are known in advance. For example, their system simplifies following the reading path of a 2-column document by interactively moving the viewport along the path, and dynamically adjusting the zoom to limit the rate of optical ow in a way similar to Speed-Dependent Automatic Zooming [17]. Our Link Sliding technique refines this idea for network navigation.

Navigation to Off-Screen Targets

More recent techniques have begun to address the problem of reaching known off-screen targets, both in terms of their visibility [12] and in navigating to them [18]. These techniques are closely related to our interest. Given n potential targets, some being off-screen, they provide visual indication of their location using a simple representation at the viewport's edge (either arcs or wedges). They then provide mechanisms to select items of interest, and to navigate to them quickly. These techniques are primarily aimed at small screens such as PDAs or smart phones, for reading maps, and reaching places of interest.

Other methods, such as the Vacuum [5], are designed for wall-sized displays, where targets may be too far away or too high to reach. These techniques rely on known targets that are attracted using a directional probe; once they have been attracted their selection becomes possible. Instead of navigating to the items, the items are brought close to the pointer for examination and selection. These techniques are closely related to our Bring & Go technique but do not use any topological information to attract objects, only geometric information.

Space Distortion Techniques

Another approach to navigating large spaces is to distort the space to shrink uninteresting areas or magnify interesting ones. Magnifying lenses are the simplest of these techniques. They have been improved with fisheye lenses [23] that can present an overview of the space while allowing local in-context zooms, featuring a smooth transition between the two regions. Latest developments include Sigma Lenses [26], which use different dimensions to transition between the overview and the local zoom. However, the maximum zoom level is about 50, still limited compared to the size of spaces such as the surface of the Earth.

Other approaches include rubber-sheet deformations [30] and folded spaces [7] where parts of the space is folded, or stretched, to provide faster navigation with context awareness. These techniques address representation, and can cope with a variety of interaction techniques for deforming the space.

Dynamic Layout for Navigation

The above-mentioned techniques are based on a stable space that users want to explore. Network visualization systems start from a graph structure and compute a layout that creates the space. Changing the layout algorithm, or parameters of these algorithms, can change the space dramatically. The Bring Neighbors Lens [31] dynamically adjusts the graph layout to show local connectivity, but is not designed as a navigation technique. Some network visualization methods do use this possibility to facilitate navigation [34]. However, if not carefully constrained, such transformations may break the user's mental map of the network [21].

To a lesser extent, the geometry of links can be changed to enhance their legibility. EdgeLens [33] interactively distorts links around the pointer, to separate close links that are hard to follow visually, or that are simply overlaid. Conversely, Hierarchical Edge Bundles [15] group links to better show aggregated trends in graph connections, and can be tuned interactively. These simple operations do not change the layout, but improve the readability of links and can help users in navigating the network.

Topological Navigation

When a spatial embedding is automatically created from a graph, the size and visibility of the graph features can be interactively controlled by the user through a "degree of interest" function, introduced by Furnas [8]. Furnas considers that items in any topology can be assigned an *a priori* importance and an importance related to a focus point. When a user selects an item, he conveys to the system the information that this item is important to him. Usually, items in the neighborhood of the focused item are also important, but not as much so, and the importance decreases with some notion of topological distance. The representation of the topology should show the focus item and its neighborhood, and then allocate less screen real-estate and visibility to items that are farther away and less important.

Again, representation can be separated from interaction. Furnas describes the application of his framework to a tree structure based on selection. This method has been used effectively on trees [28] and on networks [9]. Issues with this approach include visual discontinuity when updating the degree of interest, and inconsistency in the user's mental map when the geometry is recomputed. Topological navigation allows arbitrarily large data structures to be navigated, as only a small subset is visible at any time. Navigation consists of successive selection of neighbor items or of textual search as in SpaceTree [28].

Current navigation techniques either ignore topological information to optimize spatial navigation, or consider mainly topology while attempting to maintain a consistent geometry. Our approach considers aspects of the geometry and topology at the same time.

AUGMENTING STANDARD NAVIGATION TECHNIQUES

Pan-and-zoom navigation is a standard technique that is used in a large number of 2D navigation tasks. It is an essential element of map and graph visualization software, and many similar applications. However, in the presence of numerous crossing paths, it can be difficult to follow a single path using this technique alone. Furthermore, if the distance to the destination is unknown, users may fail to zoom-out, and require numerous clutching operations to follow the path. Bird's eye views have been shown to be effective in navigating large 2D information spaces [16, 20, 27], but they do not allow the user to clearly resolve individual paths due to the large scaling factor and the small amount of screen real estate alloted to the view (Figure 1-a).

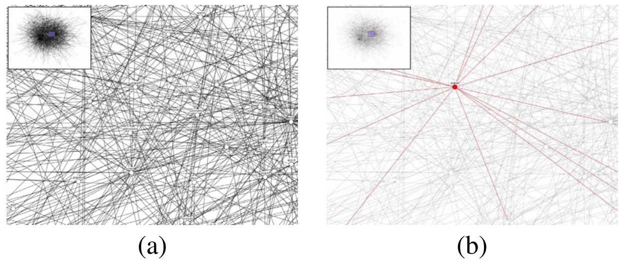


Figure 1. (a) Following a link in a dense graph can be difficult using standard Bird's Eye View navigation. (b) Highlighting a node's outgoing links make the task significantly easier.

We can simplify tasks that use connectivity information by visually distinguishing paths of interest from the background clutter. The techniques we implemented for the experiment below allow users to select a node of interest by clicking on it. The outgoing links are then colored in red, while the contrast of all other links, nodes, and labels (black by default) is reduced by rendering them a light-gray color (Figure 1(b)). Users may restore the default color-scheme by clicking again on the initial node, or may select a different node thereby highlighting its exiting links.

Our Pan & Zoom implementation supports panning by allowing the user to grab-and-drag the graph by clicking on any non-node location. The motion of the graph follows that of the mouse cursor. We use the mouse wheel to zoom by doubling or halving the view scale at each click of the wheel. As our graph labels are illegible at scales sufficient for context, we have selected this mapping to facilitate rapid switching between wide and focused views. Moving the wheel forward zooms in, and backward zooms out. Our Bird's Eye View implementation allows users to center the view over any part of the graph by simply clicking on the corresponding location in the Bird's Eye View. Users may also pan the view as in the Pan & Zoom technique, or by dragging the viewport indicator rectangle in the Bird's Eye View. For the purpose of the study, our Bird's Eye View technique does not support zooming.

LINK SLIDING

Following a route on a map, or a link in a graph visualization, is essentially a one-dimensional navigation task. However, traditional navigation techniques, such as Pan & Zoom, require the control of two or three degrees-of-freedom to accomplish the task effectively. The Link Sliding technique simplifies the control task by constraining motion to a single path. The user slides a link-cursor along the link towards the destination node, as though sliding a bead on a wire. Changes in the direction of mouse movements are only necessary if the path curves sharply. Otherwise, the user may slide between two nodes by simply moving the mouse along the direction tangent to the path. This motion does not require a high degree of precision, as any movement perpendicular to the path is ignored, and motion stops at the destination node. The view is automatically panned to follow the mouse cursor, keeping it in its initial screen location, and the zoom level is adjusted so as to provide the user with additional context while sliding along the link.

To activate Link Sliding, the user simply presses and holds the mouse button on the start node. A light-gray circle appears around the node, indicating the selection-radius (Figure 2-a). This radius specifies a region of unconstrained mouse movement, which allows the user to select an outgoing link. A link-cursor is projected onto the closest point on the nearest link to aid in selection. As the cursor passes the selection-radius, the mouse cursor is drawn towards the link-cursor. Constrained sliding proceeds as follows: at each mouse event, the system updates the mouse-cursor position $\mathbf{p}$, and calculates $\mathbf{c}$, the closest point on the path to $\mathbf{p}$, assigning it to the path-cursor position. The mouse position is pulled towards $\mathbf{c}$, by setting $\mathbf{p}' = \alpha\mathbf{p} + (\mathbf{1} - \alpha)\mathbf{c}$. We set $\alpha = 1$ within the selection-radius to allow free mouse movement, and smoothly blend it towards 0 beyond the radius by updating it at each mouse motion event: $\alpha' = \beta\alpha + \omega$. Our system sets $\beta = 0.5$ and $\omega = 0$ to rapidly pull the mouse cursor to the link.

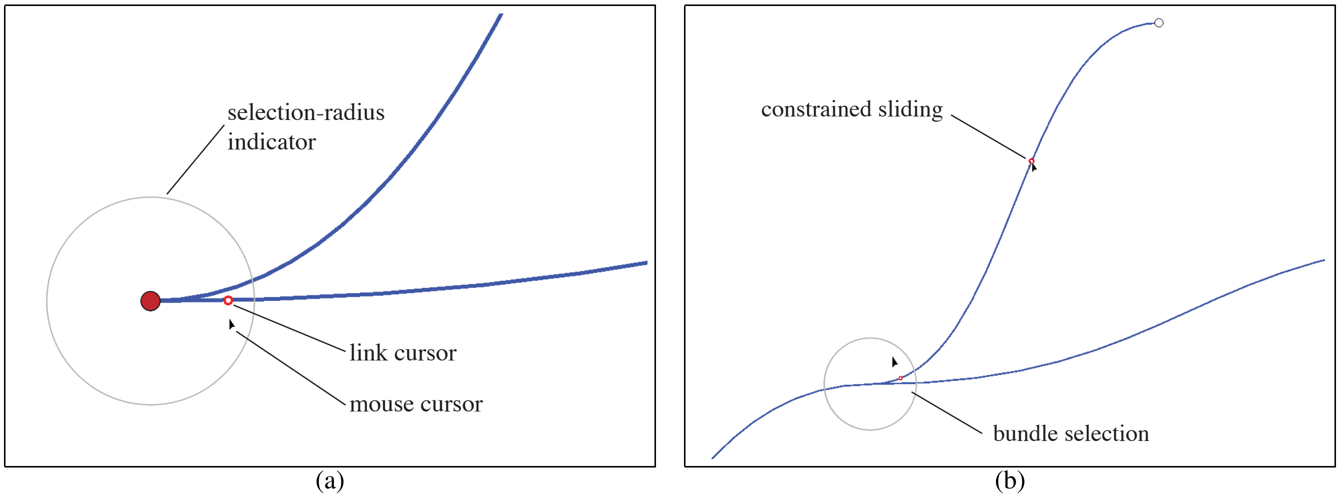


Figure 2. (a)The mouse cursor is free to move within a node's selection-radius. The link cursor shows the closest link, which will be selected upon passing the selection-radius. (b) Link selection can also be performed at junctions of edge-bundles, beyond a node or junction's selection-radius the mouse cursor is constrained to slide along the link.

Distance-dependent Automatic Zooming

Automatic zooming has been shown to reduce completion times in document navigation tasks [17]. However, previous work on speed-dependent automatic zooming [17] relied on rate-control panning and scrolling using a mouse. Pairing first-order control with an isotonic input device like the mouse is known to yield poor performance [35]. We introduce a zero-order control mapping that automatically zooms based on the user's position along a path. Distance-dependent Automatic Zooming (DDAZ) is possible, when both the start and end positions along a path are known (*e.g.* when following a route map). While a variety of useful zoom-level mappings are possible [19], for a graph navigation task we designed a mapping that sets the zoom level at the halfway point along the path so that the length of the path is equal to the viewport width. Thus, short links that traverse less than one screen-width will produce no zooming, while for long links, most of the remaining distance to the target is likely to be visible at the halfway point. Beyond providing the user with additional context, zooming the view adjusts the motor-space mapping to document space, allowing the user to move faster at the central part of the link. We use the following mapping (similar to a connes function): Given a distance d traveled by a user along a path of length l, the scale of the viewport is $z = (1-(1-(2d/l-1)^6)^4) \times (1-l/w)$, where w is the width of the viewport at $z = 1$. This mapping smoothly and rapidly reaches a wide view of the graph.

Sliding Along Edge Bundles

When a vertex in a graph has a large number of neighbors, it is helpful to gather the exiting links in *edge bundles* to reduce visual clutter, and aid in link selection [15]. Link Sliding has been designed to easily traverse edge bundles. Sliding along a bundle of links is identical to sliding along a single link until a junction in the bundle is reached. Around each bundle junction, a light-gray circle indicates a selection-radius where the mouse cursor is detached from the link cursor, allowing the user to select an exiting link in the same manner as is done at the start node. At each mouse event, the closest point on the entire bundle is computed, and its position assigned to the link cursor. This allows the user to jump between nearby links in a bundle by moving the mouse cursor rapidly away from the current link to which it is constrained. Isolated links strongly maintain the sliding constraint, as no other link can be reached within a single mouse motion event. The strength of the mouse cursor's attraction to the bundle can be modified by adjusting the smoothing parameters of α as described above.

Bundling

When a node has many outgoing links, selecting one specific link becomes difficult. To overcome that problem, we aggregate all the links that point in the same direction and construct edge bundles. We limit the number of links starting from a node to k (3 to 5 depending on the user's preference). Therefore, bundles have to split at special positions we call "junction nodes". Our algorithm constructs bundles and junction nodes until the number of outgoing links from n is less than or equal to k.

The bundling algorithm consists of two stages. It first selects the links that are to be bundled, i.e.: 1) the link to the most distant neighbor of n, called n_f, and 2) the links to the nodes closest to n_f among the neighbors of n. It then adds a single link (the bundle root) from n to a created junction node n_j. The junction node is positioned at the center of the wedge formed by the bundled links, at a distance from n that is a fraction of the distance to the closest node of the bundle. The second stage consists of changing the bundled links' sources from n to n_j. This process is repeated until all the long links have been bundled.

Bundling is only performed when the user starts the navigation by selecting the source node. It is considered a navigation mode and not a rendering style as in [15].

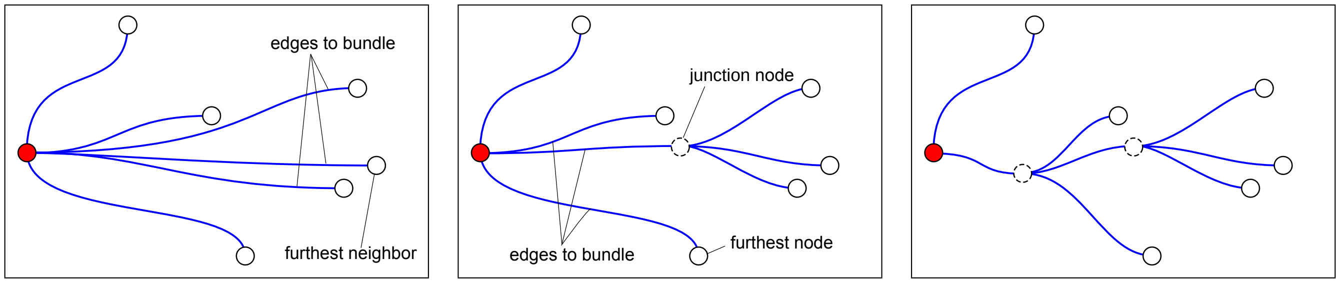
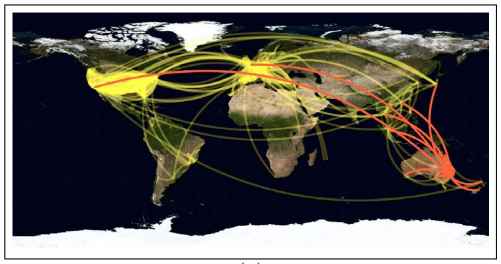
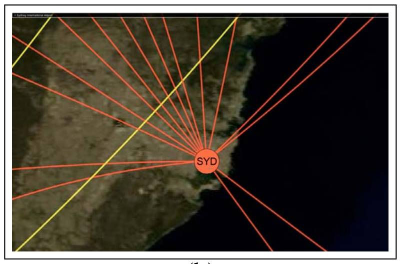
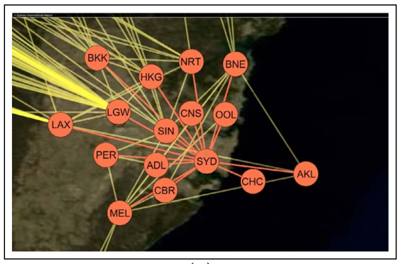


Figure 3. Multiple links leaving a node in the same compass direction (a) are collected into bundles by routing them through a dummy junction node (b). The process is repeated until the number of bundles leaving all nodes or junctions in the same compass direction fall below a given threshold (c).


(a)



(b)



(c)

Figure 4. (a) Highlighting all ights to/from Sydney, Australia. (b) Close-up on Sydney, with highlighting. (c) Bring & Go initiated on Sydney.

BRING & GO

Link Sliding makes it easy to navigate along a given path. However, it does not help in the decision process that leads to the selection of one path among many potential candidates. This decision might depend on the type of arc to be followed when there are different types of paths. It might also depend on attributes of the node at the other end of the path. Having to navigate to the other end, in order to decide whether this is the path of interest or not, quickly becomes tedious as the number of connected arcs increases.

Bring & Go addresses this problem by bringing adjacent nodes close to a node upon selection. Figure 4-a shows a map of about 700 commercial ights connecting 232 airports. Highlighting (in red) gives a general idea of the number and location of airports connected to the currently selected node: Sydney International. At this scale, the node is difficult to select, being only 1-pixel large on a 17” display. Moreover, some parts of the network are very crowded, making it difficult to visually follow the paths. One has to zoom-in to get detailed information such as airport names, thus losing context and moving all airports connected to Sydney out of the viewport (Figure 4-b). When selecting the node corresponding to Sydney, Bring & Go translates all airports connected to it inside the current viewport (Figure 4-c) using smooth animations to preserve perceptual continuity [29]. The spline curves that represent links are smoothly attened and brought inside the viewport, thus providing additional contextual information, such as the degree of connected nodes, that might help the user make his decision. For instance, the user might be looking firstly for an airport hub, which would be more likely to offer him a direct ight to his final destination.

Once the user has made a decision about what link to follow, Bring & Go makes it very easy to reach the corresponding node with a simple selection of that node. The viewport is smoothly animated, zooming out and then in to get some context, as when traveling along a path with Link Sliding (see section on distance-dependent automatic zooming for details about the computation of the trajectory in space and scale). In the meantime, all nodes and splines are animated back to their previous position and geometry, thus restoring the network to its original configuration and terminating the Bring & Go.

The concept of Bring & Go is similar to Hopping [18] and Drag-and-Pop[3], which facilitate selection by bringing proxies of potential targets closer to the cursor. Bring & Go, however, is designed for navigation, rather than selection, and can handle a much larger number of distant targets, as only connected nodes are brought close. The technique is also similar to the Bring Neighbors Lens [31], which adjusts the layout of the graph to bring connected nodes into view. However, we believe that Bring & Go’s use of proxies, rather than distortion, better preserves spatial constancy, and is critical for geographically embedded networks. Most importantly, Bring & Go works iteratively: a new Bring & Go can be initiated on a node that is currently inside the viewport as the result of a previous Bring & Go, bringing additional nodes into view, and so on. Looking for a ight from Sydney to Tel Aviv (which are not directly connected in our network), a user would easily identify London as a hub and, thanks to a second Bring & Go initiated on the London node brought inside the viewport, find that it is connected to Tel Aviv.

Bring & Go Layout

The layout algorithm for computing the position of nodes brought inside the viewport is relatively simple. As illustrated in Figure 5, nodes are placed in concentric circles centered on the selected node according to the following method.

Polar coordinates $(\rho \;\; \theta)$ are computed for all connected nodes. The list of nodes to be brought inside the viewport is sorted by distance ρ (shortest first). Nodes are then brought onto the rings following this order. For each node the algorithm checks each ring, starting from the innermost one, until it finds one where the node can be laid out, without overlapping any previously laid-out node, while keeping its θ coordinate constant. This method preserves the direction to the original location of brought nodes, and gives a sense of their relative distance to the selected node.

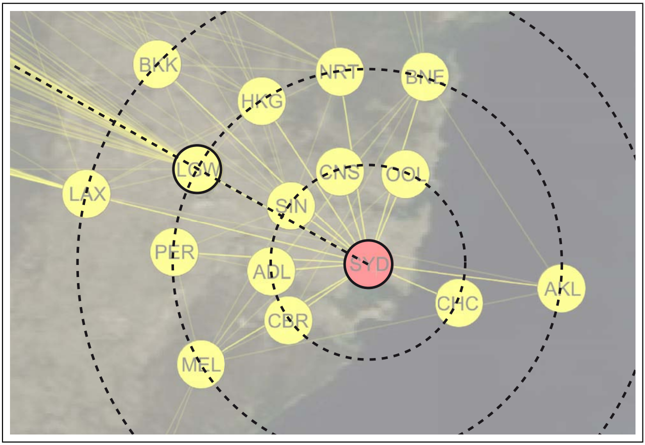


Figure 5. The layout of brought nodes preserves the direction to their actual location, and gives a sense of their relative distance to the selected node.

EXPERIMENT

We conducted an experiment to compare the performance and limits of Bring & Go (hereafter abbreviated B&G) and Link Sliding (LS) with simple visual augmentation of the methods currently available for navigating in node-link diagrams: Pan & Zoom (PZ), and interactive Bird's Eye View (BEV). Participants were asked to perform a compound navigation task on an abstract graph. We measured the performance time for each sub-task, and accuracy where relevant. Following the task participants responded to a questionnaire regarding their experience.

Task and Stimuli

All navigation tasks are set in one of two randomly generated scale-free graphs, one *sparse*, and one *dense*. The graphs were created using a Barábasi-Albert model [2]. In the graph generation process, each new node is connected to n distinct nodes, randomly chosen from the existing nodes. The *sparse graph* (1000 nodes, 1485 edges) was generated with a random connectivity $n \quad [1 \;\; 2]$, and the *dense graph* (1000 nodes, 2488 edges) using $n \quad [1 \;\; 5]$. We also provide a small graph (200 nodes, 477 edges) for training purposes.

The nodes on the graphs are labeled with the names of animals, and vegetables. Unlike a real-world task, where the node labels are meaningful to the user, random name labels can be difficult to remember. To reduce the cognitive load on our participants we consistently give the start node the label "rat", and give one of its neighbors the label "cat" (explained below).

The trials in each condition are assigned to four fixed patterns in the corresponding graph, each consisting of a starting node and its neighbors. A pattern has a starting node of degree d ($d = 5$ in the *sparse* graph, and $d = 10$ in the *dense* graph). For each technique, participants perform timed tasks for all patterns of the two graphs. As we have four techniques, we provided four versions of each graph: the initial one, and its rotations by 90, 180 and 260 degrees. The labels are changed for each graph pattern, but retain their animal/vegetable category.

As each technique we study may be best suited to a different graph navigation tasks, we select three representative task from Lee *et al.*'s task taxonomy for graph visualization [22]. The tasks include identifying all nodes connected to a given node, following a link, and returning to a previously visited link. Participants performed the three tasks in sequence to complete one trial. The first task (*neighbors* task) is to identify a node's immediate neighbors. Participants begin at the start node, which is highlighted in orange, and are asked to count how many of the node's neighbors are labeled with an animal name, and to remember the location of the cat node. Participants press the space-bar to start, and press it again when they are done counting. After typing in the number of animal nodes, the system informs the participant if they have counted correctly. In the second task (*follow* task), the system centers the view about the start node, and participants are asked to follow a link, highlighted in orange, from the start node to a selected "visit node". Pressing the space-bar begins the task and clicking on the visit node completes it. Participants are then instructed to begin the third task (*revisit* task). Participants press the space-bar to start, and must then locate the cat node that they have seen in the *neighbors* task, and click on it. They may do this either by retracing their steps back to the start node, or by moving directly to where they believe the cat node is located.

To control the tasks completion time, the sum of the distances between the starting node and its neighbors is constant in all trials of the sparse graph, and all trials of the dense graph. The distance between the starting node and the cat node is constant in all patterns, as is the distance between the starting node and the visit node. Participants were given a maximum of 40 seconds to perform each task.

Design

Our experiment follows a 4×2 within-subject design: each participant performs tasks using each of the four techniques ($Technique \quad$ PZ, BEV, B&G, LS) under two different graph conditions ($G \quad Sparse \;\; Dense$). We group trials into four blocks, one per technique, so as not to confuse participants with frequent changes of the technique. To avoid ordering effects, we balance the presentation of technique using a Latin square consisting of four presentation orders, and randomly assign three participants per order. Within a $Technique$ block, each participant perform eight measured trials, i.e., four trials with each of the two graphs. Trials within a block were presented in a random order after a training phase containing four trials, allowing participants to get familiar with a given technique before empirical measures were collected. Each navigation task (*neighbors*, *follow*, and *revisit*) within a trial had to be performed without any pause, but participants were allowed to rest between tasks. The experimenter first introduced the task, and then described each technique immediately before the corresponding block, to ensure that participants understood how each technique worked and how best to operate it.

Thirty-two measured trials per participant were analyzed, yielding a total of 384 trials:

$$\begin{array}{rl} & 12\ Participants \\ \times & 4\ Techniques \\ \times & 2\ Graphs \\ \times & 4\ Trials \\ \hline & 384\ Total\ Trials \end{array}$$

Apparatus

The experiment was ran on a HP Compaq 8710p equipped with a 2.4 GHz Intel Core 2 Duo processor, an Nvidia Quadro NVS 320M graphics card, a 1440 x 900 17" (43.2 cm) LCD monitor, and a Dell optical mouse. The mouse wheel produced 24 discrete clicks per revolution. The software was written in Java 1.6 using the Piccolo toolkit [4].

Participants

Twelve unpaid adult volunteers (8 male, 4 female), with ages ranging from 23 to 35 years, participated in the experiment. Participants were right-handed, with normal or corrected-to-normal vision.

Predictions

Neighbors Task

H0 - Time performance rank: B&G will be the fastest for both graph densities, as all of the neighbors can be seen on the screen at once with legible labels. B&G will be followed by LS as it should make navigating to neighbors easy, and by BEV, which allows fast motion through the graph. PZ is expected to be the worst performer.

H1 - Density in uence on LS*:* LS may be affected by graph density as participants must make more decisions while sliding along a bundle. This may cause its performance to drop below that of BEV for *dense graphs*.

Follow Task

H2 - Time performance rank: B&G will be the fastest for both graph densities, followed by LS; BEV and PZ will perform similarly, as following a single link requires a high degree of precision, which may be difficult to achieve using BEV.

H3 - Density in uence on LS*:* We expect LS to be affected by graph density, possibly performing slower than BEV for *dense graphs*.

Revisit Task

H4 - Time performance rank: B&G will be the fastest for both graph densities, followed closely by LS which may support greater use of spatial memory; BEV and PZ are expected to be slower due to more difficult control, but as both support use of spatial memory we do not expect the difference to be as great as in the previous tasks.

Results

In the analysis reported below the performance times of trials that were not completed successfully are replaced by the mean time of successful trials for each condition.

Neighbors Task

Error Rate: An error in this task indicates that the participant's count of neighboring nodes labeled with animal names was incorrect. Of all errors 19% were committed after the participant reached the time limit. The rest were committed within the alloted time for the task.

A Friedman's α^2 test showed a significant effect of *Techniques* (Figure 6-a). Wilcoxon's signed ranks test revealed a significant difference for *dense graphs*. BEV (42% error rate) and LS (33%) were both significantly more error-prone than B&G (10%), while PZ showed a 25% error rate.

Performance Time: Results for this task should be interpreted cautiously on account of the high rate of errors. An analysis of variance (ANOVA) revealed a significant effect of *Techniques* on Time ($F_{3,33}$ = 69.811, $p <$ 0001) (Figure 6-b). Post-hoc pairwise mean comparisons showed that B&G was significantly faster than the three other techniques and that BEV performed better than PZ. Mean times were: PZ 27.2 sec (SD=0.8), BEV 21 sec (SD=0.7), LS 24 sec (SD=0.8) and B&G 8.5 sec (SD=0.2). ANOVA also revealed a significant effect of *Graphs* ($F_{1,11}$ = 242.587, $p <$ 0001) and a significant interaction *Techniques* $\times$ *Graphs* ($F_{3,33}$ = 21.82, $p <$ 0001). The performances of all *Techniques* were degraded with *dense graphs*. As predicted in *H0*, B&G showed fastest performance at visiting the neighborhood. LS, however was slower than expected. Moreover, the density of the graphs had an in uence on time performance for all *Techniques*, whereas *H1* predicted that LS to be the most affected.

Follow Task

Error Rate: An error is reported when a participant was unable to complete the task within the time limit. Only two errors occurred, both for PZ in a dense graph. Both were due to subjects getting lost.

Performance Time: ANOVA revealed a significant effect of *Techniques* on Time ($F_{3,33}$ = 12.521, $p <$ 0001) (Figure 6(c)). Post-hoc pairwise mean comparisons showed that B&G was significantly faster than PZ and BEV and that LS was significantly faster than PZ. Mean times were: PZ 7 sec

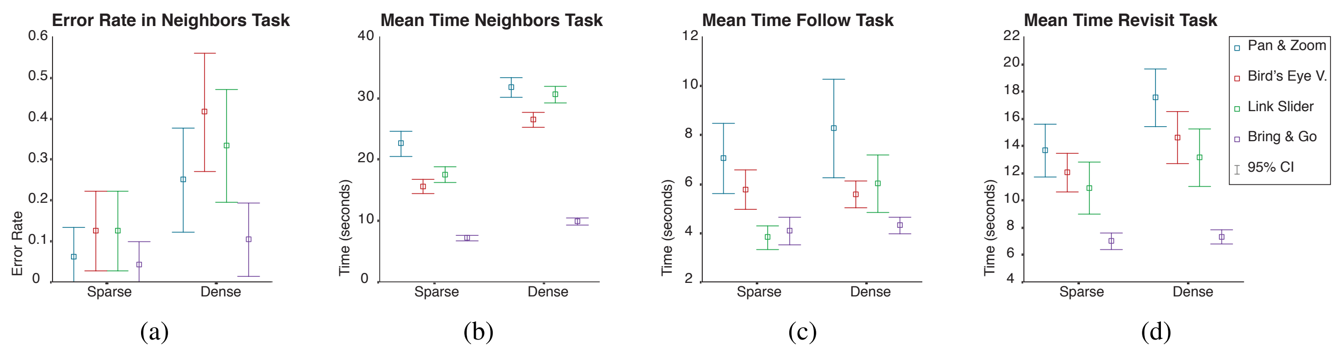


Figure 6. (a) Neighbours Task Error Rate and (b) Performance Time, (c) Follow Task Performance Time, (d) Revisit Task Performance Time.

(SD=0.4), BEV 5.6 sec (SD=0.2), LS 4.9 sec (SD=0.3) and B&G 4.2 sec (SD=0.2). ANOVA also revealed a significant effect of *Graphs* ($F_{1,11}$ = 5.81, $p <$ 05) and a significant interaction *Techniques* × *Graphs* ($F_{3,33}$ = 3.71, $p <$ 05). While the performances of PZ, BEV and B&G remains stable for both graphs densities, the performances of LS were degraded, with a mean time of 3.8 sec (SD=0.2) for *sparse graphs* and 6 sec (SD=0.6) for *dense graphs*. Here, our prediction for both time performance (*H2*) and graph density in uence on LS (*H3*) were verified.

Revisit Task

Error Rate: Nine errors occurred, 3 for PZ, BEV and LS, mainly for the dense graphs, where subjects did not find the cat node and time ran out.

Performance Time: ANOVA revealed a significant effect of *Techniques* on Time ($F_{3,33}$ = 35.453, $p <$ 0001) (Figure 6(d)). Post-hoc pairwise mean comparisons showed that B&G was significantly faster than the three other techniques and that LS was significantly faster than PZ. Mean times were: PZ 15.6 sec (SD=0.7), BEV 13.3 sec (SD=0.6), LS 12 sec (SD=0.7) and B&G 7.2 sec (SD=0.2). ANOVA also revealed a significant effect of *Graphs* ($F_{1,11}$ = 15.612, $p <$ 01). Performances on the *dense graphs* were significantly degraded. However, ANOVA did not show any significant interaction *Techniques* × *Graphs*. While our predictions on the time performance rank were verified, the difference between the spatial techniques and Bring & Go were greater then expected *(H4)*.

Qualitative Evaluation

Following the task, participants responded to a questionnaire regarding their experience. Questions were presented using a five point labeled Likert scale with labels ranging from "strongly disagree" to "strongly agree". For each technique, participants responded to statements stating that it was easy to learn, was tiring, and was pleasant to use, that they were able to accomplish the tasks quickly using the technique, and that they found it easy to accomplish the *neighbors* and *revisit* tasks using the technique. Response summaries are presented in Figure 7. Written and oral comments were also solicited.

Participants unanimously agreed that Bring & Go was quick, and made accomplishing the tasks easy. They also found it the least tiring, and most pleasant to use. Link Sliding and Bird's Eye View both received middling reactions regarding the ease and speed at accomplishing tasks. Pan & Zoom was generally rated as slow and difficult to use for the given tasks. A common complaint regarding Pan & Zoom was that the zoom control was too sensitive. We had set the zoom control to double or half the view scale at each click of the wheel in order to facilitate multi-scale navigation. This may be inconsistent with the wheel's more common use as a scrolling control, where a great amount of turning is needed to effect large changes.

Link Sliding elicited the most varied opinions regarding how pleasant it was to use. Some participants enjoyed the technique, one saying that it was "fun, and helped learning the graph layout," while others found it to be "tiring on the eyes." Several reported feeling dizzy or disoriented while sliding along links, attributing this feeling to the automatic zooming feature. The disorientation may also be due to rapid motion of the viewport while the participant's eyes were fixed on the cursor, which did not change screen location.

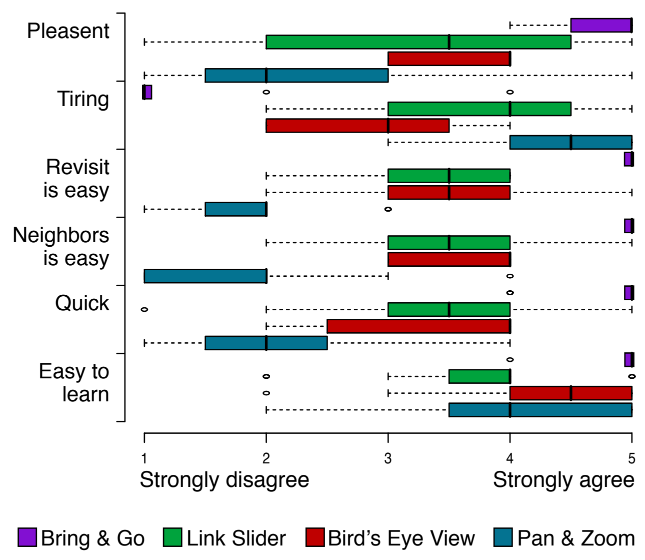


Figure 7. Questionnaire results for the four techniques.

DISCUSSION

Our experiment clearly illustrates that using connectivity information as a basis for graph navigation can significantly reduce task completion time, while improving the quality of the user experience. Specifically, the Bring & Go technique is faster and easier to use than the others we tested. While the other techniques do use connectivity information to some extent, they rely primarily on the spatial layout of the graph, and on the motion of the user's view port in this space. This is in sharp contrast with Bring & Go, which provides only slight spatial cues, and automatically controls the view port. The degree to which Bring & Go is faster is somewhat surprising, particularly for the *revisit* task, as one would expected its performance to improve with greater spatial awareness of the graph structure (*H4*). We believe that the ability to see all connected nodes quickly, and traverse links rapidly, greatly reduced our participants' need to rely on their memory.

One participant commented that Pan & Zoom "requires a lot of memory effort." This method does not allow a user to see all neighboring nodes at the same time while still being able to read their labels, consequently the user must remember the nodes' locations. Bring & Go makes remembering neighbor locations unnecessary, as they can all be seen at once. However this advantage may diminish in tasks that demand greater spatial awareness, such as navigating maps, or in networks where the location of a node is meaningful. Furthermore, the *revisit* task in our experiment only required users to traverse a single intermediate node with a known name. As the number of intermediate nodes increase, remembering a graphs connectivity may become more difficult than remembering its spatial embedding. Studying this trade-off is an interesting avenue for future research.

Beyond reducing memory requirements, Bring & Go also has a mechanical advantage over the other techniques, as the user does not need to pan the view. Panning is a closed-loop control task that must be performed almost continuously in the spatial navigation techniques, and which the user performs in parallel with a visual search. Bring & Go requires active control for only two pointing sub-tasks, and the visual search is performed separately. The difficulty of spatial navigation is re ected in participants comments on Pan & Zoom, as some note that they never used the zooming feature, finding it too difficult to control both position and scale at the same time. This difficulty may explain the high variance we observed in the Pan & Zoom technique, as some participants may have relied largely on zooming for navigation, while others would have relied more on panning.

Two key features of Link Sliding, that we expected would raise its performance over the other spatial techniques, are the simplification of the control task by reducing it to the control of a single degree of freedom, and the automatic control of the view port zoom level. Indeed, for the elementary task of following a single link, Link Sliding appeared to perform at least as well as Bring & Go This can be seen in the *follow* task in the sparse graph condition(Figure 6-c). However, this performance did not scale favorably with the complexity of the task (*H1*). Visiting all of the neighbors of a node requires repeated round-trips back to the start node. In contrast, using Bird's Eye View participants were able to move directly from neighbor to neighbor. A number of our participants commented that it was too difficult to jump between adjacent links, and one explicitly requested a mechanism for shortcuts between links. Another possible barrier to allowing Link Sliding to navigate easily from high-valence nodes is the effect of edge-bundling. While edge-bundling reduces clutter, and simplifies link selection when leaving a node, it also changes the visual orientation of links, and requires the user to follow a more complex path with multiple junctions that are not a part of the underlying graph's topology. The change in angle can be particularly problematic when the user attempts to return to the previous node after releasing the mouse button, as the new node's outgoing bundles do not match the original node's bundles, so the user must leave by following a bundle aimed in a slightly different direction then of the bundle on which she arrived. We believe that some of these issues may be addressed by optimizing the bundling algorithm to minimize changes in link angle, or by finding an optimal set of static edge-bundles for the entire graph.

CONCLUSION

This work has began the exploration of topology-aware navigation of large graphs, by examining several possible methods ranging from mostly spatial, to mostly topology-based navigation. We have found the technique Bring & Go, which relies more on topology than spatial location, to hold a clear advantage for several key navigation tasks. The Link Sliding technique, which attempts to combine the advantages of both spatial and topological navigation, did not perform as well as expected, performing the same as, or only slightly better than Bird's Eye View. However, we believe that Link Sliding is worthy of further investigation for navigating networks, such as route-maps, that are spatially embedded. While Bring & Go works well for finding labeled nodes, it provides very little geographical context. For example, finding all of the coastal towns of an unknown country would be a difficult task using Bring & Go, as the user cannot easily follow the shoreline. This task could be easily accomplished using either Link Sliding or Bird's Eye View navigation.

Our results suggest that any system for visualizing large networks will profit from some form of topology-aware navigation using one or more of the techniques described here. As both the visual augmentation, as well as the navigation techniques, are triggered only in the context of links and nodes, they do not interfere with existing spatial navigation. Indeed, a combination of methods may be required for high-level navigation tasks, as each method has its own unique strengths.

Bring & Go and Link Sliding have been implemented using both Piccolo [4] and ZVTM [25], and are available in ZGRViewer[1], a tool for navigating large graph layouts computed using the GraphViz package[2].

[1]`http://zvtm.sf.net/zgrviewer.html`
[2]`http://www.graphviz.org/`

REFERENCES

1. E. Adar. Guess: a language and interface for graph exploration. In *CHI 06*, 791–800, New York, NY, USA, 2006. ACM.
2. A. L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
3. P. Baudisch, E. Cutrell, D. Robbins, M. Czerwinski, P. Tandler, B. Bederson, and A. Zierlinger. Drag-and-pop and drag-and-pick: Techniques for accessing remote screen content on touch and pen-operated systems. In *Proc. Interact 03*, 57–64, 2003.
4. B. B. Bederson, J. Grosjean, and J. Meyer. Toolkit design for interactive structured graphics. *IEEE Trans. on Software Engineering*, 30(8):535–546, 2004.
5. A. Bezerianos and R. Balakrishnan. The vacuum: facilitating the manipulation of distant objects. In *CHI 05*, 361–370, New York, NY, USA, 2005. ACM.
6. Delta Air Lines. Delta air lines route map, 2008. http://delta.innosked.com/.
7. N. Elmqvist, N. Henry, Y. Riche, and J.-D. Fekete. Melange: space folding for multi-focus interaction. In *CHI 08*, 1333–1342, New York, NY, USA, 2008. ACM.
8. G. W. Furnas. Generalized fisheye views. *SIGCHI Bull.*, 17(4):16–23, 1986.
9. E. R. Gansner. Topological fisheye views for visualizing large graphs. *IEEE Trans. on Visualization and Computer Graphics*, 11(4):457–468, 2005.
10. Google. Google maps, 2008. http://maps.google.com.
11. Y. Guiard, F. Bourgeois, D. Mottet, and M. Beaudouin-Lafon. Beyond the 10-bit barrier: Fitts' law in multi-scale electronic worlds. In *HCI 01*, 573–588. Springer, 2001.
12. S. Gustafson, P. Baudisch, C. Gutwin, and P. Irani. Wedge: clutter-free visualization of off-screen locations. In *CHI 08*, 787–796, New York, NY, USA, 2008. ACM.
13. J. Heer, S. K. Card, and J. A. Landay. prefuse: a toolkit for interactive information visualization. In *CHI 05*, 421–430, New York, NY, USA, 2005. ACM.
14. I. Herman, G. Melancon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Trans. on Visualization and Computer Graphics*, 06(1):24–43, 2000.
15. D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Trans. on Visualization and Computer Graphics*, 12(5):741–748, 2006.
16. K. Hornbæk, B. B. Bederson, and C. Plaisant. Navigation patterns and usability of zoomable user interfaces with and without an overview. *ACM Trans. Comput.-Hum. Interact.*, 9(4):362–389, 2002.
17. T. Igarashi and K. Hinckley. Speed-dependent automatic zooming for browsing large documents. In *UIST 00*, 139–148, New York, NY, USA, 2000. ACM.
18. P. Irani, C. Gutwin, and X. D. Yang. Improving selection of off-screen targets with hopping. In *CHI 06*, 299–308, New York, NY, USA, 2006. ACM.
19. E. W. Ishak and S. K. Feiner. Content-aware scrolling. In *UIST 06*, 155–158, New York, NY, USA, 2006. ACM.
20. V. Kaptelinin. A comparison of four navigation techniques in a 2d browsing task. In *CHI 95*, 282–283, New York, NY, USA, 1995. ACM.
21. H. Lam, R. A. Rensink, and T. Munzner. Effects of 2d geometric transformations on visual memory. In *APGV 06*, 119–126, New York, NY, USA, 2006. ACM.
22. B. Lee, C. Plaisant, C. S. Parr, J.-D. Fekete, and N. Henry. Task taxonomy for graph visualization. In *BELIV 06*, 1–5, New York, NY, USA, 2006. ACM.
23. Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Trans. Comput.-Hum. Interact.*, 1(2):126–160, 1994.
24. K. Perlin and D. Fox. Pad: an alternative approach to the computer interface. In *SIGGRAPH 93*, 57–64, New York, NY, USA, 1993. ACM.
25. E. Pietriga. A toolkit for addressing hci issues in visual language environments. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 05)*, 145–152, Los Alamitos, CA, USA, 2005. IEEE Computer Society.
26. E. Pietriga and C. Appert. Sigma lenses: focus-context transitions combining space, time and translucence. In *CHI 08: Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems*, 1343–1352, New York, NY, USA, 2008. ACM.
27. E. Pietriga, C. Appert, and M. Beaudouin-Lafon. Pointing and beyond: an operationalization and preliminary evaluation of multi-scale searching. In *CHI 07*, 1215–1224, New York, NY, USA, 2007. ACM Press.
28. C. Plaisant, J. Grosjean, and B. B. Bederson. Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *INFOVIS 02*, 57, Washington, DC, USA, 2002. IEEE Computer Society.
29. G. G. Robertson, S. K.Card, and J. D.Mackinlay. Information visualization using 3d interactive animation. *Communication of the ACM*, 36(4):56–71, 1993.
30. M. Sarkar, S. S. Snibbe, O. J. Tversky, and S. P. Reiss. Stretching the rubber sheet: a metaphor for viewing large layouts on small screens. In *UIST 93*, 81–91, New York, NY, USA, 1993. ACM.
31. C. Tominski, J. Abello, F. van Ham, and H. Schumann. Fisheye tree views and lenses for graph visualization. In *IV 06: Proceedings of the conference on Information Visualization*, 17–24, Washington, DC, USA, 2006. IEEE Computer Society.
32. J. J. van Wijk and W. A. Nuij. A model for smooth viewing and navigation of large 2d information spaces. *IEEE Trans. on Visualization and Computer Graphics*, 10(4):447–458, 2004.
33. N. Wong, S. Carpendale, and S. Greenberg. Edgelens: An interactive method for managing edge congestion in graphs. In *InfoVis 03*, 51–58. IEEE Computer Society, 2003.
34. K.-P. Yee, D. Fisher, R. Dhamija, and M. Hearst. Animated exploration of dynamic graphs with radial layout. In *INFOVIS 01*, 43, Washington, DC, USA, 2001. IEEE Computer Society.
35. S. Zhai. User performance in relation to 3d input device design. *SIGGRAPH Comput. Graph.*, 32(4):50–54, 1998.