Eurographics / IEEE Symposium on Visualization 2011 (EuroVis 2011)
H. Hauser, H. Pfister, and J. J. van Wijk
(Guest Editors)

Volume 30 (2011), Number 3

Dynamic Insets for Context-Aware Graph Navigation

S. Ghani[1], N. Henry Riche[2], and N. Elmqvist[1]

[1]Purdue University, USA
[2]Microsoft Research, USA

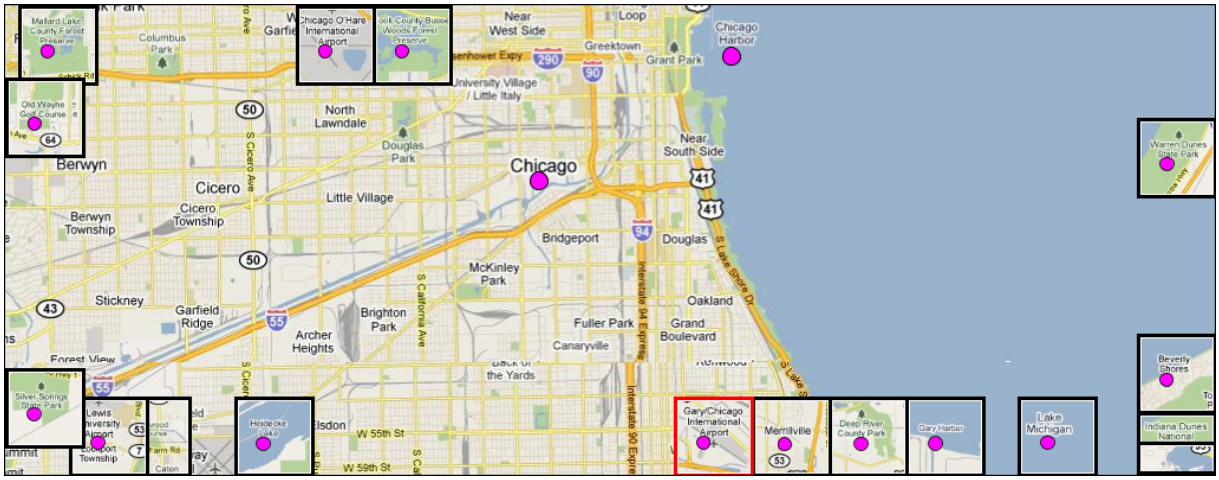


Figure 1: *The Dynamic Insets technique for a map of the Chicago area showing insets for off-screen nodes with their context.*

Abstract

Maintaining both overview and detail while navigating in graphs, such as road networks, airline route maps, or social networks, is difficult, especially when targets of interest are located far apart. We present a navigation technique called Dynamic Insets that provides context awareness for graph navigation. Dynamic insets utilize the topological structure of the network to draw a visual inset for off-screen nodes that shows a portion of the surrounding area for links leaving the edge of the screen. We implement dynamic insets for general graph navigation as well as geographical maps. We also present results from a set of user studies that show that our technique is more efficient than most of the existing techniques for graph navigation in different networks.

Categories and Subject Descriptors (according to ACM CCS): H.5.2 [Information Interfaces and Presentation]: User Interfaces—Interaction styles I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

1. Introduction

Interacting with road networks in a handheld GPS or on map websites such as Google Maps is just one example of large-scale networks that lots of people use in their everyday life, and there exist many other graph applications for social networks (such as on Facebook), large hierarchies, and network topologies. Navigation in these networks is usually done using a sequence of pan and zoom operations [FB95], sometimes with a bird's eye view. For example, to move from one position in a large network to another position far away, the common approach is to zoom out, pan to that position, and finally zoom in [vWN03]. However, this can be tedious, ineffective, and even disorienting [Fur86, FB95].

A recent trend is to utilize the topology of a graph to en-

hance navigation; examples include Link Sliding and Bring & Go [MCH*09]. However, these techniques do not provide awareness of the context around destination nodes, which is useful information when deciding where to go next.

In this paper, we present a technique called *Dynamic Insets* that supports multi-focus interaction during navigation in graphs. Drawing inspiration from cartography, we add dynamic map insets that show a small part of the surrounding area for the destinations of links leaving the boundary of the screen (Figure 1). Insets can be clicked to automatically animate the user's viewpoint to that destination. We also introduce interest functions that control which off-screen nodes should be made visible through insets. By choosing between different interest functions, the user can opt to show **all** neighboring nodes, or filter to show only nearby gas stations (for a map application), central actors (for a social network), or wireless access points (for network topology design).

Effective graph navigation techniques are essential for managing the large graphs that are common to many graphical applications [HMM00]. To validate the new technique, we also perform a controlled experiment involving human subjects where we compare it to the Bring & Go technique of Moscovich et al. [MCH*09]. Our results show that the technique significantly helps users in maintaining an awareness of both overview and details of the network. We follow up with two qualitative studies where we apply dynamic insets to social networks and geographic maps. Our findings suggest that the technique also scales to larger graphs, and that it does not interfere with mental map creation.

2. Related Work

Herman et al. [HMM00] emphasize navigation and interaction techniques as essential for understanding large graphs. Here we review relevant work on navigation, off-screen awareness, and specific graph navigation techniques.

2.1. Common Navigation Techniques

Pan and zoom are common navigation techniques which control the position and dimension of the graphical viewport on the visual space. Since the original infinitely zoomable canvas proposed in Pad [PF93], there has been a large amount of work in this area. Furnas and Bederson [FB95] designed a space-scale formalism for describing and understanding pan and zoom interactions. However, pan and zoom can be tedious when accessing distant objects [Fur86].

Splitting the screen into smaller viewports is another standard method for large visual spaces, and gives the user awareness of multiple regions of the space. A special type of split-screen technique is called overview and detail, where one of the viewports shows a bird's eye view of the area surrounding the other viewport (the detail). The overview often includes a visual cue indicating the current viewport, and can be used to navigate. Hornbæk and Frøkjær [HF01] showed that overview and detail techniques can outperform panning and fisheye views for some tasks. Because overviews often occupy small screen space, the large scale factor becomes a hindrance in finding specific landmarks or following a link.

A third approach is to distort the visual space nonlinearly to ease navigation. For example, fisheye views [Fur86] magnify the focus region while showing the surrounding context in less detail. Elmqvist et al. [EHRF08] propose a space-folding technique that combines different parts of the visual space so that multiple focus points and their surrounding context are visible. However, the transition between focus and context needs to be carefully designed [PA08] and fisheyes may cause problems for targeting [Gut02].

2.2. Navigation to Off-Screen Targets

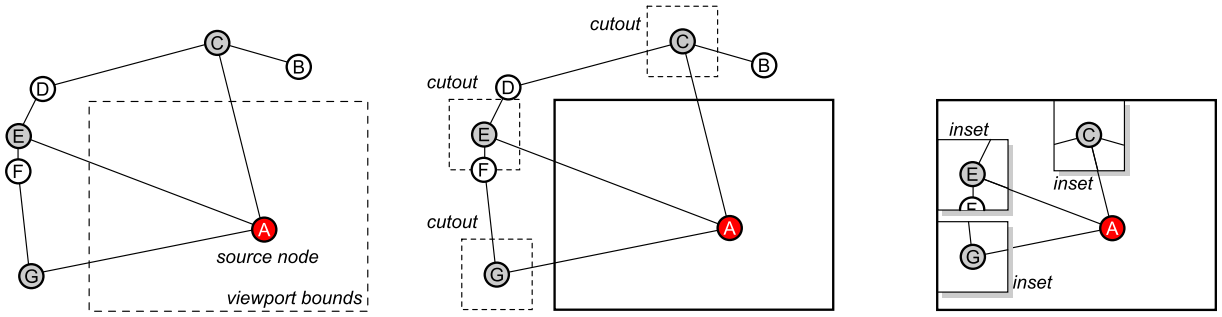
Visual Cues. Some techniques have been specifically designed to provide awareness of off-screen targets. Halo [BR03], and its descendants Wedge [GBGI08] and EdgeRadar [GI07], indicate off-screen targets by adding visual cues to the border of the viewport. While these techniques give users awareness of off-screen targets, they do not provide a mechanism to reach them.

Proxies. Proxy-based techniques provide local copies of distant or off-screen targets to ease their selection; examples include drag-and-pop [BCR*03] and the Vacuum [BB05]. Because these techniques are primarily designed for selection rather than navigation, they do not provide the target context, forcing users to visit each target to see the context.

Proxies+Context. One of the most relevant techniques to ours is WinHop [PNIG07], which introduces an inset to let users explore distant regions without leaving the current location. However, WinHop does not take advantage of the network topology, so the user must first select the node to expand. This affects its ability to scale to large spaces. Frisch and Dachselt [FD10] recently proposed a proxy+context technique for UML class diagrams where connected classes outside the current view are projected onto an interactive border region. However, like WinHop, their technique requires extra interaction to see a preview of off-screen targets.

2.3. Navigation in Networks

Distortion Lenses. Some distortion techniques are specifically designed for graphs. EdgeLens [WCG03] interactively distorts links around the focus point to remove clutter. Most relevant to us is the Bring-Neighbor Lens [TAvHS06], an interactive lens that temporarily modifies the graph layout to show all neighbors of the node in focus. However, this particular lens was not designed for navigation: users can gain awareness of the neighbors by temporarily bringing them into the viewport, but are still required to use navigation techniques such as panning and zooming to reach them.



(a) Compute interest for off-screen nodes. (b) Generate cutouts. (c) Add insets to viewport.

Figure 2: *Dynamic construction of insets for a simple graph consisting of 7 nodes and 8 edges. A is the source node (n_s).*

Degree-of-Interest Techniques. Furnas [Fur86] showed that the visibility of graph features can be controlled by the user through a degree of interest (DOI) function. Nodes are ranked according to their topological distance to the focus node in the graph. The graph may then be distorted to show the most interesting information to the user. This principle has been employed for hierarchies [PGB02] and in large networks [GKN05, vHP09]. However, these techniques deform the graph, potentially compromising the user's mental map.

Topology-Aware Navigation. Recent work [MCH*09] presented two navigation techniques taking advantage of the topology of the network to ease navigation. The first technique, Link Sliding, provides guided panning and zooming when continuously dragging along a link. The second technique, Bring & Go, gathers the neighbors of a particular node and enables the user to navigate to one of these. These techniques were compared to overview+detail and pan and zoom techniques. Bring & Go outperformed both of these, while Link Sliding had mixed results. Regardless, from all the techniques presented in this section, these are most relevant to ours. We designed dynamic insets to overcome the principal drawback of Bring & Go: the lack of context for off-screen targets, requiring the user to visit each node.

3. Dynamic Insets

Dynamic Insets is a topology-aware navigation technique for providing context awareness while traversing large-scale networks. It uses the connectivity of the graph to bring off-screen neighbors of on-screen nodes—and their context—into the viewport as insets (Figure 1). In this way, multiple destinations for visual links leaving the edge of the screen can be explored without actually leaving the current location; this is more generally known as *multi-focus interaction* [EHRF08]. The insets are created by cutting out a region of the visual space surrounding the off-screen neighboring nodes and bringing these regions into the main viewport as small, nested viewports. Figure 2 shows the basic idea, where a simple network with three off-screen nodes are shown on the screen using dynamically created insets.

While the above description captures the essence of the dynamic insets technique, there exist several details concerning *which* off-screen nodes to include, how insets are created, how they are laid out, and how to display their distance from the current focus. We discuss these issues below.

3.1. Degree-of-Interest Function

Instead of trying to visualize *all* off-screen targets, such as existing off-screen navigation techniques [PNIG07], dynamic insets uses the connectivity of the graph itself to determine *which* off-screen targets to include. This means that the technique can handle even large graphs.

To give additional expressive power to the technique, we use the concept of a *degree-of-interest (DOI) function* [Fur86] to rank all off-screen nodes in terms of their interest for a particular task. The simplest imaginable DOI function is a binary one that uses a neighbor relation between the set of visible nodes V and an off-screen node n:

$$DOI(n,V) = \begin{cases} 1 & \exists v \in V : \text{neighbor}(n,v) \\ 0 & \text{otherwise} \end{cases}$$

In other words, this function will assign 1 to all neighbors of currently visible nodes, and 0 to all others. An important special case for this situation is where the user has selected a particular source node n_s to use as a focus point (instead of all currently visible nodes). The above DOI function can be trivially adapted for this purpose by passing $V = \{n_s\}$.

Given any DOI function of the above format, we rank all off-screen nodes according to their current DOI values. A certain number of the highest ranked off-screen nodes will be selected depending on the number of dynamic insets to create. Ties may be arbitrarily broken, for example using distance, so that nearby nodes are given precedence over more distant ones. However, a better approach is to devise a mod-

ified DOI function that takes distance into account:

$$DOI_{dist}(n,V) = DOI(n,V)/\text{dist}(n)$$

where $\text{dist}(n)$ is the distance from n to the edge of the screen.

The default setting for our technique is to use the standard DOI function defined above, but to allow the user to select a focused source node n_s by clicking on it. Clicking outside any node on the visual space will clear the current source node and return to the default setting.

In addition, more advanced interest functions are also possible, including interest functions composed of several simpler interest functions, or which make use of domain-specific information. Here follows examples of such DOI functions:

- For a GPS, interests could be assigned for nearby gas stations, shopping malls, or hospitals, weighted by distance;
- For airline route selection (such as the Delta Air Lines Route Map at `http://delta.innosked.com/`), destinations could be ranked according to their ticket price, travel duration, or number of stops; and
- For social network analysis, the interest of off-screen neighbors could be based on the connection strength.

3.2. Insets

Given that we have derived *which* off-screen nodes to make visible on the screen, the next step is to actually create the insets. Insets are small rectangular regions containing the target node and its surrounding context in the visual space. They have a visual border to differentiate them from the main viewport, and, if possible, they are laid out on the edge of the screen to coincide where the link they are associated with leaves the main viewport (see below). Initially, the zoom factor in the inset is the same as the main view.

Figure 2(c) shows the inset for three off-screen nodes in a simple graph. Insets are not static, but the viewport inside each inset area can be interacted with, such as through panning or zooming in and out (left-dragging and using the mouse wheel in our implementation, respectively). Clicking an inset will animate the position of the user's viewport to the location of the node in the inset. This both provides a quick way of navigating in the graph, and also makes the path from source to destination clearly visible to the user, thereby increasing their awareness of the whole visual space.

3.3. Inset Layout

We have designed a simple layout algorithm for positioning insets along the edge of the screen. It has two constraints:

- **Align insets with their links.** Insets should be placed so that their location on the edge of the screen coincides with the outgoing visual link that it is associated with. Ideally, the visual link on the main viewport and on the inset viewport should align perfectly to maximize the visual connection (all three insets in Figure 2(c) exhibit this property).
- **Avoid total occlusion.** For situations with many outgoing links, insets may start to overlap. This is acceptable as long as no single inset is fully occluded by another.

For overlapped insets, the algorithm accordingly stacks the insets as shown in Figure 3. We also implement a paging interaction technique similar to that in the Mélange technique [EHRF08], where the user can hover the mouse cursor over an inset to bring it to the top.

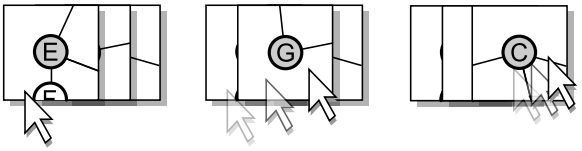


Figure 3: *Insets that overlap are stacked so that all insets are visible. This allows them to be paged through by hovering the mouse over the visible part, bringing it to the top.*

3.4. Drag-to-Fan

Sometimes paging between insets is not sufficient if the number of stacked insets is very high, such as for a central actor in a social network with a degree in the hundreds or thousands. We propose the *drag-to-fan* technique (Figure 4) to support this situation, where the user can separate (fan out) stacked insets by dragging the mouse on an inset stack. The mouse cursor then controls the radius of a semi-circle whose perimeter will be used to space out the stacked insets. The inset stack essentially becomes a pie menu, and users can themselves control how far they have to displace insets to be able to see the desired destination.

Accordingly, selecting an inset using drag-to-fan is done by "dialing in" the inset, i.e., through the angle between the pointer and the original click. What happens when releasing the button is configurable: the straightforward approach is that this means travel to the inset, just like clicking on an inset. However, because this does not allow users to cancel the travel operation if they do not find the inset they are looking for in the stack, our implementation just changes the stacking order to bring the selected node to the top of the stack.

Drag-to-fan can also be used for iteratively exploring inset neighbors (i.e., neighbors of neighbors) by dragging on an inset with the right mouse button pressed. This would cause the neighbors of that inset to expand along the perimeter of the circle, thereby matching the iterative navigation functionality of the Bring & Go technique.

3.5. Distance Awareness

Our new technique does not intrinsically support distance awareness in its visual representation such as, for instance, Bring & Go [MCH*09] does. Nevertheless, we can easily extend the representation of individual insets to incorporate

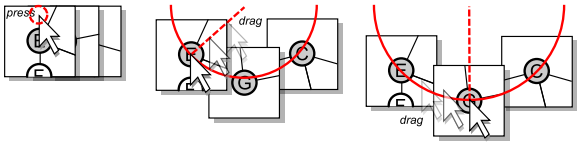


Figure 4: *Interaction sequence for the drag-to-fan technique. The user presses and holds the mouse button on an inset, then drags to fan the insets along the perimeter of a semi-circle. Releasing the button over an inset will execute the configured action (travel to, bring to top, etc).*

this information. Here are a few different alternatives (not implemented or evaluated in this paper):

- **Number:** The simplest approach is to add a number for the distance from the edge of the screen to the destination node shown in the inset (Figure 5(a)).
- **Border Color:** A color scale (i.e., a heatmap) can be used for the inset border to convey distance (Figure 5(b)).
- **Transparency:** Changing the inset transparency: distant nodes are translucent, nearby ones opaque (Figure 5(c)).
- **Size:** Scaling the size of the inset so that distant destinations are smaller than nearby ones (Figure 5(d)).
- **Overview map:** This could show both the extents of the main viewport as well as the actual position of each inset.

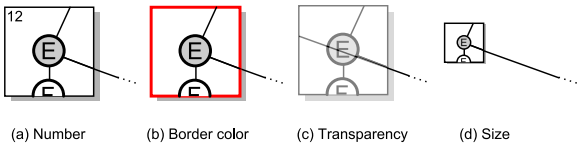


Figure 5: *Four distance visualizations: (a) actual number, (b) border color, (c) inset transparency, and (d) inset size.*

3.6. Implementation Notes

We have implemented dynamic insets in a graph visualization built using Java and the Piccolo [BGM04] toolkit. Our algorithm first calculates DOI values for all off-screen nodes and ranks them using a priority queue. Given a specific budget N of the maximum number of insets to display, the N most highly-ranked off-screen nodes are selected and cameras are placed at their locations. A layout algorithm tries to optimize the location of the insets associated with each camera on the screen border. This algorithm will first strive to preserve link alignment, but, failing that, will stack overlapping insets to ensure that no inset is fully occluded. If there are two insets for the same node but for different links, the layout algorithm will combine these insets if they are located within a specific distance of each other on the canvas.

4. Controlled Experiment

We conducted a study to validate that the dynamic insets technique does provide efficient graph navigation. Here we describe our methods, present the results, and discuss them.

4.1. Method

4.1.1. Participants and Apparatus

We recruited 12 participants balanced for age and gender, and screened to not be color-blind. All participants indicated that they used computers more than 16 hours/week. The participant pool consisted of 6 males and 6 females, with ages ranging from 22 to 47. Each participant used a 3.00 GHz dual-core PC with 4 GB of memory, running Microsoft Windows Vista, and equipped with a 21" flat-screen monitor set to a resolution of 1600×1200 (1000×1000 window size).

4.1.2. Techniques

We used our Dynamic Insets (DI) technique as described in this paper. Drag-to-fan was disabled to minimize extra interaction. Moscovich et al. [MCH*09] showed that navigation techniques taking advantage of graph topology outperforms traditional techniques such as pan and zoom, and bird's eye views. In particular, the Bring & Go (BG) technique presented in that paper outperformed all other techniques. Thus, we decided to compare our technique to Bring & Go.

We reimplemented Bring & Go, using standard values for the animation speed for bringing neighbors (500ms) and for traveling to destinations (600ms).

4.1.3. Tasks

We used the first task used by Moscovich et al. [MCH*09] since it captures awareness of direct neighbors. We also included two additional tasks to capture context awareness. The three tasks tested were the following:

CN CountNeighbors. How many neighbors of the node named "cat" are vegetables?

CC CloseContext. Which neighbor of the "cat" node is enclosed by a **red** circle? (The circle diameter is small enough to be fully visible in the insets.)

DC DistantContext. Which neighbor of the "cat" node is enclosed by a **red** circle? (The circle diameter is larger than the size of the insets.)

We included two context tasks to force participants to have to zoom and pan the insets in the DI condition. We wanted to investigate whether this extra interaction would cause the technique to exhibit worse performance than BG.

4.1.4. Datasets

To measure the performance of the techniques in realistic conditions, we selected two datasets with two densities: sparse vs. dense. To allow us to strictly control the network

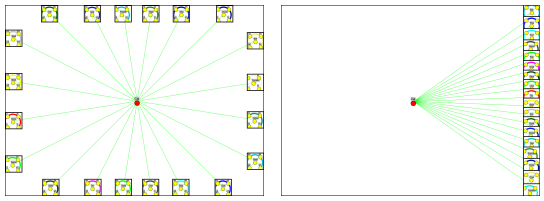
topology, we did not generate complete graphs, but only a subset of those nodes and edges involved in a particular task. This also allows us to keep the experiment to a reasonable length and limit user fatigue and frustration (potentially induced by visiting a very large number of neighbors).

To achieve this, our graphs had a star structure with the source node, labeled "cat", at the center of the star. We then created 20 potential destination nodes as neighbors to "cat" (level 1), each in turn having an additional four local neighbors (level 2). Thus, we had a total of 101 nodes and 100 edges. All level 1 neighbors were located at random distances from the source node, but sufficiently distant to be outside of the viewport when viewing the source "cat" node at default magnification level (pan and zoom was disabled). Level 2 neighbors were arranged equidistantly in a circle around their level 1 parent—the size of this circle was small enough to fit inside the insets for the CC task, whereas it was larger than the insets for the DC task (requiring zooming).

To achieve different graph densities, we varied only the concentration of the destination nodes (level 1) in space:

- **Sparse.** Neighbors are equally distributed in a circle around the source node (Figure 6(a)).
- **Dense.** Neighbors are placed in a 90° arc centered around the horizontal to the right of the source (Figure 6(b)).

This strategy lets us keep the number of neighbors to visit stable while reproducing the artifacts induced by both sparse and dense graphs. More specifically, the dense case causes high overlap for insets that mimics a large graph without penalizing the BG technique. For the DI technique, however, users will be forced to page between insets to find the correct target. Again, we wanted to investigate whether this extra interaction would impact the performance of the DI technique.



(a) Sparse - CloseContext. (b) Dense - CloseContext.

Figure 6: *Social network with the dynamic insets technique.*

4.1.5. Experimental Design

Given the above factors, we used a within-subject full-factorial design: 2 navigation techniques (N) $\times$ 2 densities (D) $\times$ 3 tasks (T) $\times$ 3 repetitions = 36 unique conditions. We counterbalanced the order of the techniques. The order of tasks was fixed whereas density was randomized and graphs were randomly generated for each trial, including the number of targets (for CN), and the position of the red circle (for CC and DC). With 12 participants, we collected data for a total of 432 individual trials.

4.1.6. Procedure

Participants received training before each technique and each task. We ensured they answered correctly before performing the timed tasks. For each trial, participants clicked on a button to indicate that they had finished reading the description of the task and were ready to begin, and pressed the space bar when they were done. The application recorded accuracy and completion time. The completion timer started only after each navigation technique had been first activated (i.e., after the initial BG animation had ended). We did not enforce any time limit. After the experiment, we collected user preferences and comments using a questionnaire. The study lasted approximately one hour, including the initial training session and the post-experimental questionnaire.

4.1.7. Hypotheses

H1 For CountNeighbors, DI will perform as well as BG for both correctness and completion time.

H2 For CloseContext, DI will be faster than BG because there is no need to travel to a neighbor to see its context.

H3 For DistantContext, DI will be faster than BG in the sparse graph because panning and zooming in each inset for DI is easy. However, for dense graphs, BG will be faster because the insets in DI overlap with each other, requiring additional interaction.

4.2. Results

We averaged measurements for all repetitions for each condition. Below we discuss these results in more detail.

4.2.1. Correctness

Correctness was high: 97 % across all tasks. Table 1 summarizes the main effects on correctness for all factors, analyzed using logistic regression. As the table shows, only Task T had a significant effect. We studied this using a Tukey HSD, and found that the only significant difference was that the CN task was less accurate than the CC task ($|t| = 2.05, p = .04$). For the CN task, the mean correctness was 93 % for both navigation types with no significant difference between them ($F(1,11) = .08, p = 0.78$).

Task	Factors	df, den	F	p
All	Navigation type (N)	1, 11	0.08	
	Density (D)	1, 11	0.08	
	Task (T)	2, 22	3.36	*

* = $p \leq 0.05$, ** = $p \leq 0.001$.

Table 1: *Effects of factors on errors (logistic regression).*

4.2.2. Completion Time

Table 2 summarizes the significant effects on completion time using a repeated-measures analysis of variance (RM-ANOVA). We found that the completion time violated the normality assumption of the ANOVA, so we analyzed the logarithm of the times. Other assumptions were met. Figure 7 shows boxplots for the completion time as a function of the navigation type N and other factors. In particular, we analyzed the task factor T using a Tukey HSD and found that all pairwise differences were significant ($p < .001$) in the order DC > CN > CC (decreasing time, CC was fastest).

Task	Factors	df, den	F	p
All	Navigation type (N)	1, 11	205.37	**
	Density (D)	1, 11	2.31	
	Task (T)	2, 22	85.97	**
	V * T	2, 22	229.75	**
	D * T	2, 22	9.82	**
	V * D * T	2, 22	3.46	*
CN	Navigation type (N)	1, 11	6.26	**
	Density (D)	1, 11	1.92	
CC	Navigation type (N)	1, 11	559.58	**
	Density (D)	1, 11	7.59	*
DC	Navigation type (N)	1, 11	34.93	**
	Density (D)	1, 11	3.89	

* = $p \leq 0.05$, ** = $p \leq 0.001$.

Table 2: *Effects of factors on time (RM-ANOVA).*

4.3. Discussion

We can summarize our findings as follows:

- Bring & Go is faster (by 15% in completion time) than dynamic insets for CountNeighbors, but not more accurate (partially confirming H1);
- Dynamic insets are faster (89%) than Bring & Go for the CloseContext task (confirming H2); and
- Dynamic insets are faster (44%) than Bring & Go for the DistantContext task (partially confirming H3).

4.3.1. Explaining the Results

The results from our study obey our intuition, but there are some surprises as well. Collectively, as we correctly hypothesized, dynamic insets are faster than the Bring & Go technique (Figure 7(a)). This is clearly an effect of the user being able to see the off-screen nodes along with their context without the need to travel to each node.

However, the fact that dynamic insets were significantly slower than Bring & Go for the CountNeighbor task is a surprise (Figure 7(b)); we had expected the techniques to be comparable in time. We think the reason for this effect is that users had more screen space to cover for DI than for BG: for Bring & Go, all neighbors are grouped together close to the center node, whereas for dynamic insets, the neighbors are spread out in insets placed around the edge of the screen.

As hypothesized, dynamic insets are significantly faster than Bring & Go for the CC task (Figure 7(b)). According to the same figure, they are also faster for the DC task, which we had not anticipated. It seems that the benefit of being able to see the context surrounding a node without having to travel there outweighs the extra interaction needed to page between stacked insets in the dense case. In fact, density had no significant impact on completion time (Figure 7(c)). While surprising, this shows that both Bring & Go and dynamic insets are robust against locally high node densities.

The post-questionnaire results also showed that all participants preferred DI to BG. Most of them explained that BG required them to travel to the neighbors' locations to assess the context, whereas the context was easily accessible on the viewport using DI. Several participants stated that even with insets overlapping and the need to use the zoom inside the inset, DI was still far less tedious to use than BG.

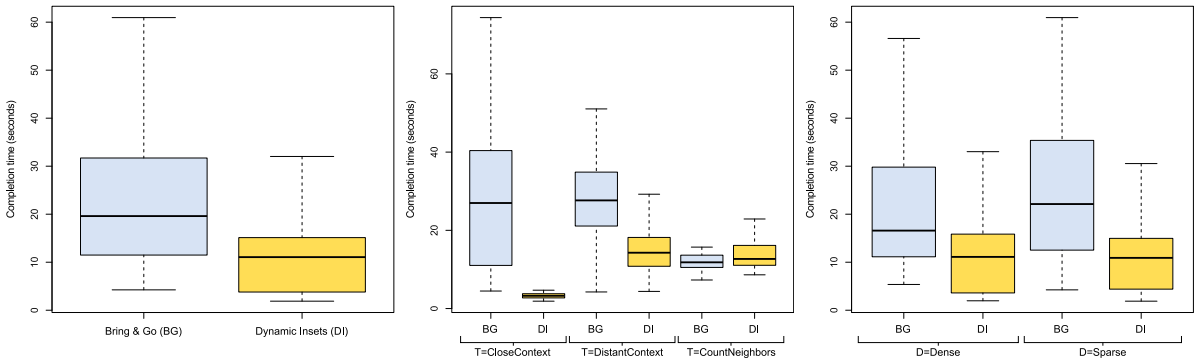
4.3.2. Limitations and Generalizations

To keep the experiment at a reasonable length, we did not include standard techniques such as pan and zoom, and bird's eye views. We based this decision on Moscovich et al.'s [MCH*09] study that demonstrated that the Bring & Go technique outperformed these. From our study results, we can conclude that dynamic insets outperform Bring & Go for all context-related tasks. Therefore, we can reasonably argue that dynamic insets outperform standard techniques such as pan and zoom or bird's eye views for these tasks.

However, further research is required to study how these techniques help users maintain a mental map of the network. For example, Bring & Go distorts the network (potentially breaking the users' mental map) but uses the layout of the neighbors brought to maintain distance awareness. Dynamic insets do not distort the network but introduce visual encodings for indicating the distance. It is difficult to evaluate the impact of these compromises, and techniques such as pan and zoom may perform better here for mental map building.

We opted to disable drag-to-fan and as well as iterative neighbor navigation for both techniques to keep the complexity of the experiment and the dataset low. Future studies are needed to investigate navigation performance for iterative invocations of these two techniques.

Another potential limitation is that our controlled experiment uses a tree instead of a full graph. Our motivation was that we wanted to study the canonical graph navigation task, i.e., navigating from one node to another, and in such situations, there is no need to model the full graph. Also, we performed our experiment using a small network consisting of only 101 nodes and 100 edges. We argue that dynamic insets and Bring & Go only act on local sub-graphs



(a) All tasks combined. (b) Task T. (c) Density D.

Figure 7: *Completion time as function of navigation type N and the other experimental factors.*

and thus that the results found in our experiment are generalizable to larger networks, particularly for small-world networks [WS98] (such as social networks) that are locally dense but globally sparse. Finally, in the dense case we do not increase the number of neighbors but only their spatial arrangement (all to one side of the starting node). This is so that the task is equally difficult for BG, but potentially more difficult for DI due to overlapping insets.

Nevertheless, to address all of these issues, we study dynamic inset performance in follow-up experiments involving larger graphs of a more realistic nature, described next.

5. Examples and Follow-Up User Studies

To showcase the applications of dynamic insets, we designed examples for two general applications: social networks and maps. For each application, we conducted follow-up user studies with 6 new participants recruited from the general population. We aimed at getting feedback on the usability and effectiveness of dynamic insets to navigate large visual spaces. We were also interested in studying if our navigation technique interfered with the mental map of the participants.

Metric (endpoints)	GeoMap	SocNet
Efficiency (low/high)	4.00 (0.89)	2.83 (1.17)
Enjoyability (low/high)	4.00 (0.63)	2.83 (0.98)
Ease of use (low/high)	4.50 (0.55)	3.83 (0.98)
Visual clutter (low/high)	3.50 (0.84)	4.00 (0.63)
Aids mental map (no/yes)	4.50 (0.55)	2.67 (1.63)
Context utility (low/high)	4.67 (0.52)	4.00 (0.82)
Use in daily work? (no/yes)	3.83 (0.75)	2.67 (1.03)

Table 3: *Subjective ratings for follow-up studies (Likert scale 1-5 averages, standard deviations in parentheses).*

5.1. Geographic Maps

The inspiration for the dynamic insets technique comes from cartography, and, not surprisingly, there is excellent potential for using the technique for cartographic applications. To begin realizing this potential, we implemented a map scenario that uses static (i.e., not live) data from Google Maps. Figure 1 shows a map of Chicago and its environs. We have modeled a connectivity graph (invisible) for a few locations outside of the viewport; insets are automatically created for these and laid out on the edge of the screen. This example clearly shows the benefit of contextual information from the geographical features visible in the insets.

Because we are currently only modeling connectivity, and not the actual geographical road network, insets are placed in the direction of the destination, and not on the road. This is a limitation of our example and not the technique itself.

Our map application is similar to existing work in cartography; for example, Karnick et al. [KCJ*10] present a method for visualizing route maps with multiple focus points by showing overview and detail views of the route within a single visual frame. However, their goal is to provide a static route map rather than to facilitate navigation.

5.1.1. Follow-up User Study

We engaged 6 new participants (4 male, 2 female, all used computers more than 16 hours/week) in a follow-up user study to perform tasks using dynamic insets on a geographic map of the Chicago area. This scenario lasted around 30 minutes. All 6 participants were excited with the technique and commented in the first few minutes how useful it would be in their everyday tasks when navigating geographical maps. Three of them mentioned that training was not even needed to use dynamic insets and that navigation was "so much easier than with traditional zooming and scrolling."

All participants successfully performed some 20 graph-related tasks such as finding nodes, counting neighbors,

identifying clusters, following paths, etc. We also asked them to revisit previously visited locations at different stages of the exploration. Geographical features on the map—labels, roads, cities, parks, and water—provide very rich context and most participants explained that after 5 minutes of exploration, they could quickly assess which part of the map the insets were from. We were surprised to observe almost 100% performance in revisitation tasks. One participant could exactly remember the inset configurations and found previously visited locations using long series of inset revisitation. This participant was also able to revisit different places using panning when explicitly asked to do so. Overall, these findings indicate that the technique did not interfere with participants building mental maps of the visual space.

The center column in Table 3 shows subjective ratings for the map scenario. As the numbers show, participants liked this scenario and thought the technique was easy-to-use and enjoyable. They did remark on the high visual clutter that arises in some situations, and asked for a way to control the creation or filtering of insets. However, all six noted this as an additional desirable feature, and not a scalability issue.

5.2. Social Networks

Social networks are the canonical application of graph visualization. Using our Java framework for dynamic insets, we implemented a simple social network visualization and utilized our technique to showcase how navigation can be improved in such networks. The tool reads GraphML files and renders the graph with icons for each individual node and colored clusters in the viewport background (Figure 8).

5.2.1. Follow-up User Study

The same 6 participants as in the study above used our visualization tool to explore a large social network. Our dataset was the ACM AVI conference co-authorship network: a large graph with 450 authors and approximately 1,000 edges (communicating co-authorship), laid out using a lin-log graph layout. We added 16 node clusters to provide some context to the network. To ease the training and tasks, we told our participants that this network represented friends grouped by their music preferences. Participants were briefly instructed in how to use the technique and then followed a task sheet to perform about 20 graph-related tasks. The scenario lasted approximately 30 minutes.

The rightmost column in Table 3 shows subjective ratings for the social network study. Ratings are lower than for the geographical map. We believe this is due to the fact most of our users were not familiar with social networks; some stated they did not see why they would perform such tasks at all. In other words, their motivation to use our technique in this scenario was low. In addition, the network proposed was much larger and denser than in the map scenario, making tasks more difficult to complete, and, despite getting a break, participants reported fatigue from their previous exploration.

Despite these lower ratings, our participants performed very well with dynamic insets, indicating that the technique scales to larger graphs. All participants were able to accurately perform all tasks, including spatial memory tasks (navigating to specific communities or nodes). We were surprised that users were able to memorize the location of specific communities after such a short exploration time and the lack of distinctive spatial features compared to the map scenario. This observation further indicates that dynamic insets do not interfere with the creation of a mental map. While all participants were able to perform tasks in very cluttered areas (up to 25 insets on the screen edge), they suggested again that filters would allow them to better control which insets to show. Nevertheless, all participants used either the fanning, panning or the paging technique in solving tasks.

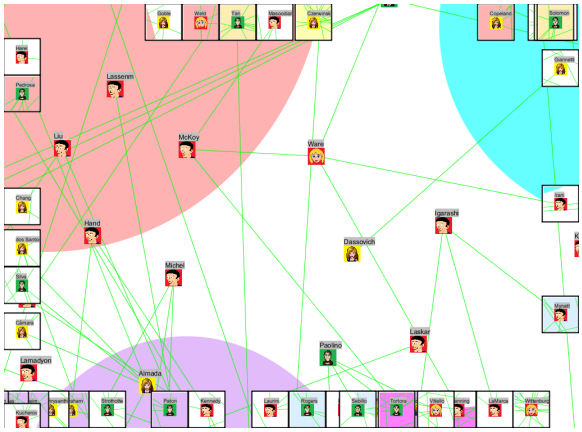


Figure 8: *Social network for the AVI co-authorship dataset. This application was used in one of the follow-up studies.*

6. Conclusion and Future Work

We have presented a context-aware graph navigation technique that utilizes the topology of a graph to dynamically create insets for off-screen neighbors of visible nodes. We give two examples showcasing the utility of the technique for social networks and geographical maps. We also present results from a controlled experiment that shows that our technique outperforms current state-of-the-art graph navigation techniques, as well as more qualitative findings from two usability studies with larger and more realistic tasks.

In our future endeavors we want to study the performance of different distance and degree-of-interest mechanisms. We also plan to deploy the technique in a real-world online map website such as Google Maps or Bing Maps.

Acknowledgments

This work was partly supported by the U.S. Department of Homeland Security's VACCINE Center under award no. 2009-ST-061-CI0001.

References

[BB05] BEZERIANOS A., BALAKRISHNAN R.: The vacuum: facilitating the manipulation of distant objects. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems* (2005), pp. 361–370. 2

[BCR*03] BAUDISCH P., CUTRELL E., ROBBINS D., CZERWINSKI M., TANDLER P., BEDERSON B., ZIERLINGER A.: Drag-and-pop and drag-and-pick: Techniques for accessing remote screen content on touch- and pen-operated systems. In *Proceedings of INTERACT* (2003), pp. 57–63. 2

[BGM04] BEDERSON B. B., GROSJEAN J., MEYER J.: Toolkit design for interactive structured graphics. *IEEE Transactions on Software Engineering 30*, 8 (2004), 535–546. 5

[BR03] BAUDISCH P., ROSENHOLTZ R.: Halo: a technique for visualizing off-screen objects. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems* (2003), pp. 481–488. 2

[EHRF08] ELMQVIST N., HENRY N., RICHE Y., FEKETE J.-D.: Mélange: Space folding for multi-focus interaction. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems* (2008), pp. 1333–1342. 2, 3, 4

[FB95] FURNAS G. W., BEDERSON B. B.: Space-scale diagrams: Understanding multiscale interfaces. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems* (1995), pp. 234–241. 1, 2

[FD10] FRISCH M., DACHSELT R.: Off-screen visualization techniques for class diagrams. In *Proceedings of the ACM Symposium on Software Visualization* (2010), pp. 163–172. 2

[Fur86] FURNAS G. W.: Generalized fisheye views. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems* (1986), pp. 16–23. 1, 2, 3

[GBGI08] GUSTAFSON S., BAUDISCH P., GUTWIN C., IRANI P.: Wedge: clutter-free visualization of off-screen locations. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems* (2008), pp. 787–796. 2

[GI07] GUSTAFSON S. G., IRANI P. P.: Comparing visualizations for tracking off-screen moving targets. In *Extended Abstracts of the ACM CHI Conference on Human Factors in Computing Systems* (2007), pp. 2399–2404. 2

[GKN05] GANSNER E. R., KOREN Y., NORTH S. C.: Topological fisheye views for visualizing large graphs. *IEEE Transactions on Visualization and Computer Graphics 11*, 4 (2005), 457–468. 3

[Gut02] GUTWIN C.: Improving focus targeting in interactive fisheye views. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems* (2002), pp. 267–274. 2

[HF01] HORNBÆK K., FRØKJÆR E.: Reading of electronic documents: The usability of linear, fisheye, and overview+detail interfaces. In *Proceedings of ACM CHI Conference on Human Factors in Computing Systems* (2001), pp. 293–300. 2

[HMM00] HERMAN I., MELANÇON G., MARSHALL M. S.: Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics 6*, 1 (Jan./Mar. 2000), 24–43. 2

[KCJ*10] KARNICK P., CLINE D., JESCHKE S., RAZDAN A., WONKA P.: Route visualization using detail lenses. *IEEE Transactions on Visualization and Computer Graphics 16*, 2 (2010), 235–247. 8

[MCH*09] MOSCOVICH T., CHEVALIER F., HENRY N., PIETRIGA E., FEKETE J.-D.: Topology-aware navigation in large networks. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems* (2009), pp. 2319–2328. 2, 3, 4, 5, 7

[PA08] PIETRIGA E., APPERT C.: Sigma Lenses: Focus-context transitions combining space, time and translucence. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems* (2008), pp. 1343–1352. 2

[PF93] PERLIN K., FOX D.: Pad: An alternative approach to the computer interface. In *Computer Graphics (ACM SIGGRAPH '93 Proceedings)* (1993), pp. 57–64. 2

[PGB02] PLAISANT C., GROSJEAN J., BEDERSON B. B.: SpaceTree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *Proceedings of the IEEE Symposium on Information Visualization* (2002), pp. 57–64. 3

[PNIG07] PARTRIDGE G., NEZHADASL M., IRANI P., GUTWIN C.: A comparison of navigation techniques across different types of off-screen navigation tasks. In *Proceedings of INTERACT* (2007), vol. 4663 of *Lecture Notes in Computer Science*, Springer, pp. 716–721. 2, 3

[TAvHS06] TOMINSKI C., ABELLO J., VAN HAM F., SCHUMANN H.: Fisheye tree views and lenses for graph visualization. In *Proceedings of the International Conference on Information Visualization* (2006), pp. 17–24. 2

[vHP09] VAN HAM F., PERER A.: "Search, show context, expand on demand": Supporting large graph exploration with degree-of-interest. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 953–960. 3

[vWN03] VAN WIJK J. J., NUIJ W. A. A.: Smooth and efficient zooming and panning. In *Proceedings of IEEE Symposium on Information Visualization* (2003), pp. 15–22. 1

[WCG03] WONG N., CARPENDALE M. S. T., GREENBERG S.: EdgeLens: An interactive method for managing edge congestion in graphs. In *Proceedings of the IEEE Symposium on Information Visualization* (2003), pp. 51–58. 2

[WS98] WATTS D. J., STROGATZ S. H.: Collective dynamics of 'small-world' networks. *Nature 393* (1998), 440–442. 8