Interaction Techniques for Selecting and Manipulating Subgraphs in Network Visualizations

Michael J. McGuffin, *Member, IEEE*, and Igor Jurisica

Abstract—We present a novel and extensible set of interaction techniques for manipulating visualizations of networks by selecting subgraphs and then applying various commands to modify their layout or graphical properties. Our techniques integrate traditional rectangle and lasso selection, and also support selecting a node's neighbourhood by dragging out its radius (in edges) using a novel kind of radial menu. Commands for translation, rotation, scaling, or modifying graphical properties (such as opacity) and layout patterns can be performed by using a hotbox (a transiently popped-up, semi-transparent set of widgets) that has been extended in novel ways to integrate specification of commands with 1D or 2D arguments. Our techniques require only one mouse button and one keyboard key, and are designed for fast, gestural, in-place interaction. We present the design and integration of these interaction techniques, and illustrate their use in interactive graph visualization. Our techniques are implemented in NAViGaTOR, a software package for visualizing and analyzing biological networks. An initial usability study is also reported.

Index Terms—Interactive graph drawing, network layout, radial menus, marking menus, hotbox, biological networks.

1 Introduction

Despite a growing number of algorithms (e.g. [5]) available for computing the layout of graphs, there remains a significant need for users to be able to interactively manipulate such graphs, to manually adjust their layout or to change the display options (colours, labels, node shapes, etc.) associated with subsets of nodes. For example, in bioinformatics, different researchers looking at the same biological network are often interested in emphasizing different nodes and pathways within the network, in applying different layout constraints to subsets of nodes, and in using different mappings for colours and other graphical attributes. Users may also want to adjust the layout of a network opportunistically as they explore it. Thus, fully automated algorithms for graph layout cannot offer a complete solution for the visualization needs of users.

In addition, there is a trend toward larger networks, as well as increasingly feature-rich, complex software packages [18]. As networks become larger, manual adjustment of their layout becomes more difficult, and as software becomes more complex, there tend to be more menu items, dialog boxes, widgets, and modes that the user must navigate, ultimately slowing down the user, consuming screen space, and creating opportunities for mode errors [17].

Within the human-computer interaction (HCI) community, several advanced popup or gestural interaction techniques have been proposed (e.g. marking menus [11], hotbox [12], Control Menus [15], Flow Menus [7], FaST Sliders [13]) to enable rapid access to a large number of functions, reducing screen space usage, and eliminating mode errors. However, these techniques have so far seen little application within the visualization community, particularly to the problem of interactive graph layout and manipulation. This paper presents an integrated set of advanced interaction techniques inspired by previous approaches, designed for selecting subgraphs within a network and invoking commands on the selected subgraph to change layout or display options. Our work introduces novel extensions to radial menus as well as extensions to Kurtenbach et al.'s hotbox [12]. Our interaction techniques support multiple selection methods, including rectangle and lasso selection, and a novel selection method based on the network's edge-connectivity. Our hotbox allows commands to be selected with simultaneous specification of 1D or 2D arguments, as required by commands such as "move" or "adjust opacity" (of the selected nodes). Our techniques have been prototyped in NAViGaTOR [9], a software package for visualizing biological networks[1]. Although our work is motivated by its application within bioinformatics, our techniques are general and could be applied in many situations involving interactive manipulation of graphs.

The contributions of our work are (1) novel variants on marking menus and hotbox widgets that could be applied to domains other than graph visualization; (2) a new technique for quickly selecting neighbourhoods of nodes of arbitrary radius based on edge-connectivity, with a single drag; (3) a tight integration of interaction techniques allowing the user to quickly select subgraphs (using rectangle, lasso, neighbourhood-radius, and set operations) and invoke multiple commands (some of which may involve 1D or 2D arguments) using just 1 keyboard key and 1 mouse button, while avoiding persistent modes; (4) a demonstration of how simple layout patterns coupled with translate, rotate, and scale operations can be applied to subgraphs to quickly customize the layout of a graph; and (5) the results of an initial usability study.

2 Required Functionality

This section gives an overview of the kinds of functions that we require our interaction techniques to support. The details of how these functions will be *invoked* (i.e. with a particular button, gesture, or widget) is the topic of subsequent sections.

Our work was motivated by a desire to improve the user interface in NAViGaTOR [9], a freely-downloadable software package for visualizing and analyzing biological networks, such as protein-protein interaction networks. The data sets visualized in NAViGaTOR usually comprise a few dozen to tens of thousands of nodes, where each node usually represents a protein, and edges between them represent interactions. Typically, a force-directed layout algorithm is used to establish an initial layout of the network. The user may also adjust the layout of the network by selecting one or more nodes and changing their position, optionally *fixing* them in place so the force-directed layout will not subsequently change their positions. If desired, such manual adjustment of the layout can be done with the force-directed layout running in the background on the other nodes, incrementally updating their positions over time, as the user repositions fixed nodes. Nodes and edges have several graphical attributes including their label, shape, colour, and opacity, and these can be manually modified or

- *Michael J. McGuffin is with École de technologie supérieure, Montreal, Canada, E-mail: michael.mcguffin@etsmtl.ca.*
- *Igor Jurisica is with Ontario Cancer Institute, PMH/UHN, Toronto, Canada, E-mail: juris@ai.utoronto.ca.*

Manuscript received 31 March 2009; accepted 27 July 2009; posted online 11 October 2009; mailed on 5 October 2009.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org .

[1] Readers may also experience a small Java applet demonstration of our interface at http://profs.logti.etsmtl.ca/mmcguffin/research/hotterBox/

automatically chosen based on metadata associated with the nodes and edges.

A first set of functions to which we wish to give easy access are related to *selection* of subgraphs. In particular, we want the user to be able to easily select (or deselect) individual nodes, as well as perform selections by dragging out a rectangle or by drawing a lasso curve around nodes. We would also like to facilitate selection based on the edge-connectivity within the network, for example, to make it easy to select the neighbours of one or more nodes.

A second set of functions are used to perform operations (i.e. *commands*) on the selected nodes. These include commands to change the displayed shape (circle, diamond, square, triangle) of the selected nodes; to adjust their opacity; to show or hide their labels; to "fix" their positions so nodes remain anchored in place during force-directed layout, or "unfix" them to let them move freely; to adjust per-node force parameters or weights; to collapse or expand meta-nodes; to translate, rotate, or scale[2] the positions of the selected nodes; to "linearize" or "circularize" the positions of the selected nodes (explained below); or to delete the selected nodes. Examples of the application of these commands are in Figure 1.

The "linearize" and "circularize" commands, which were developed during our research, are examples of commands that apply a *layout pattern* to the selected nodes. When the linearize command is invoked, a straight line that fits the initial positions of the nodes is found using Principle Component Analysis (PCA). The positions of the nodes are then projected onto this line, and shifted to be evenly spaced. The circularize command similarly projects the positions of nodes onto a circle whose centre is the centroid of the initial node positions, and whose radius is the average distance between the centroid and the initial node positions. The user may precede either of these commands with a command to fix the positions of the nodes, so that the node positions will not be disturbed during force-directed layout. The user may also then apply any combination of translation, rotation, and scaling to change the positioning of nodes, while preserving their layout along a line or circle. Note that neither linearize nor circularize creates an explicitly-stored constraint on the node positions; they simply change the positions of nodes into a pattern that is naturally preserved during translation, rotation, and scaling of the set of nodes. We found these commands to be quite useful for customizing the layout of subgraphs (Figure 10 shows another example).

A third set of functions could be described as "global" functions, since they do not operate specifically on the selected subset of nodes. These include functions to select all nodes, to deselect all nodes, to invert the set of selected nodes, and to turn the force-directed layout on or off. (An "undo" function would also fall in this category, although our prototype does not currently support undo.)

As new functions have been added to NAViGaTOR, we found that an increasing number of widgets (such as menu items) and keyboard shortcuts were cropping up in the interface, and there was temptation to add toolbar buttons that would introduce (persistent) modes. A standard point of view within the HCI community is that *persistent* modes, that involve no kinesthetic feedback and little or no visual feedback (beyond perhaps a change in mouse cursor shape, or some text in a status bar), are to be avoided, because they create opportunities for mode errors [17]. Examples of persistent modes are the various drawing tools often found in bitmap editing programs, and in the classic text editor `vi`, in which users frequently become confused as to which mode they are in, and even expert users have a habit of hitting the Escape key more than necessary to ensure they are in one mode rather than another. Far preferable are interfaces that are sometimes called *modeless*, i.e. that involve only *transient* modes (also known as quasi-modes, spring-loaded modes, or kinesthetically-held modes) where there is strong visual or kinesthetic feedback that the user is in a special, temporary mode. Examples of transient modes include when a popup menu is popped up, or when the user must hold down a special key to remain in a temporary mode (in which the mouse performs some special function) and returns to their initial mode as soon as the key is released.

We avoided persistent modes in versions 1.1 and 1.2 of NA-

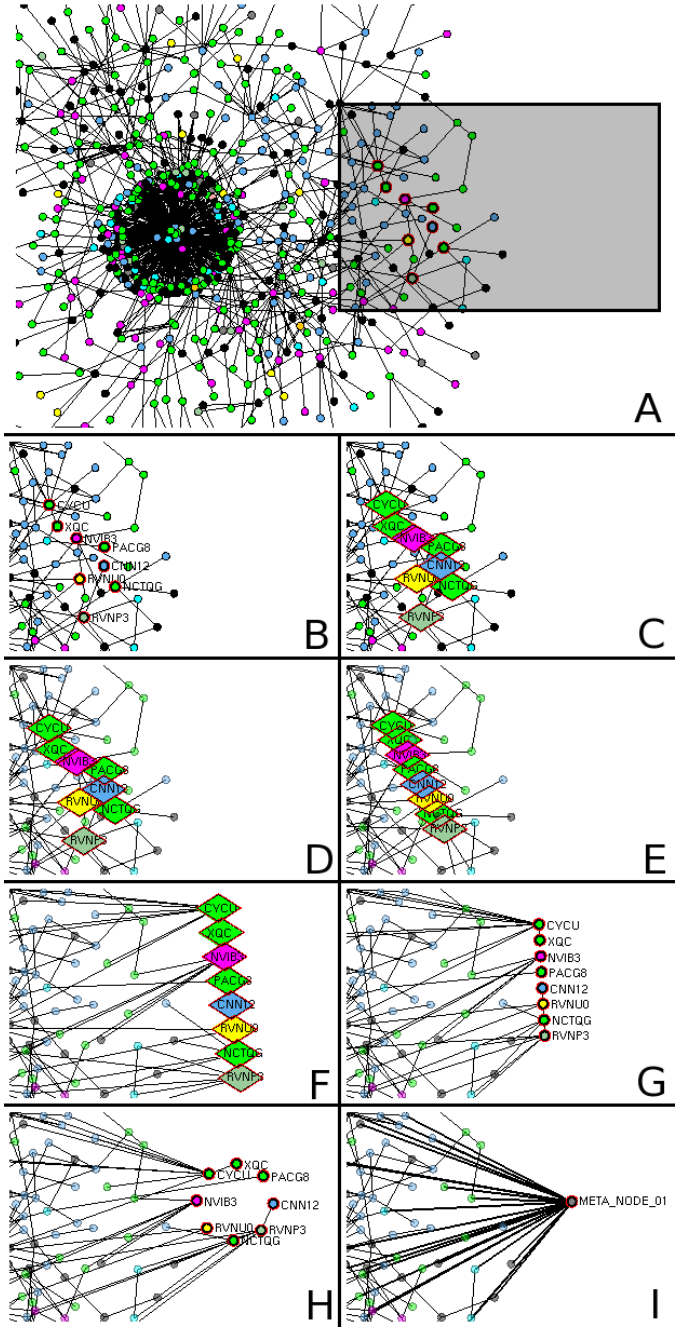

Fig. 1. Individual steps in some modifications of a network visualization. *A*: Beginning with this network, the user performs several operations within the shaded area, shown in the subsequent steps. *B*: After selecting 8 nodes of interest, the user turns on the displaying of labels for those nodes. *C*: The shape of the selected nodes is set to diamonds. *D*: The users inverts the selection (to select all other nodes), reduces the opacity of these other nodes, and inverts the selection a 2nd time to result in the 8 original nodes again being selected. *E*: The 8 nodes are "linearized", i.e. their positions are changed to fall at evenly-space positions along a straight line. *F*: The positions of the nodes are translated, rotated, and scaled, to avoid occluding the rest of the network. *G*: Node shapes are changed back to circles, and positions are again scaled and rotated slightly. *H*: The nodes are "circularized". *I*: Nodes collapsed to a single meta-node.

[2]Note that rotation and scaling are with respect to the centroid of a set of selected nodes, and have no effect if there is only one node selected.

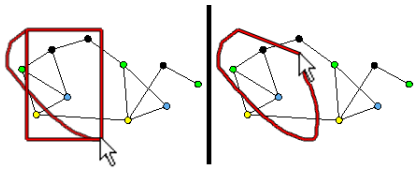

Fig. 2. Integrated rectangle/lasso selection. *Left:* The user begins dragging; an ink trail and a rubber-band rectangle are both displayed. If the user releases while both are displayed, the drag is interpreted as a rectangle selection. *Right:* The user drags back toward the starting point until the ink trail exceeds some threshold measure of "closure". The system ceases to display the rectangle, and draws a line segment from the cursor position to the starting position to show how the lasso would be completed. The user may continue to drag to specify the remaining part of the lasso curve, and may release at any time to finish the lasso selection.

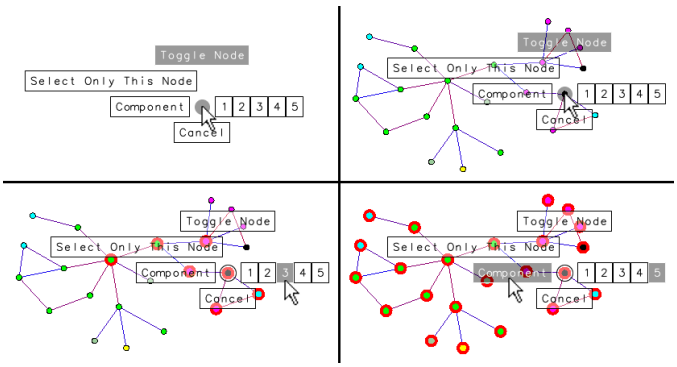

Fig. 3. The node-specific menu, which is invoked over a given node (*Upper Right*). For clarity we also show it with no data behind it (*Upper Left*). The centre and north items co-hilite, indicating that the default action is to toggle the node's selection state, which can be achieved by quickly press-releasing over a node (the press invokes the menu, the release selects the centre). Dragging east specifies a neighbourhood of a given radius (*Lower Left*), during which nodes hilite to give a preview of the specified neighbourhood; a release completes the selection. Flicking west (*Lower Right*) is a shortcut for the entire connected component, as indicated by the co-hiliting of the west item and the east-most "5" item. Flicking north-west is a shortcut to select only the given node and deselect all others.

ViGaTOR by adding several keyboard modifier key + mouse button combinations to invoke transient, kinesthetically-held modes for mouse input, including rectangle selection, lasso selection, and translation/rotation/scaling of selected nodes. Keyboard shortcuts and modifier key + mouse button combinations are useful to expert users for quickly invoking special functions; however, novice users must first learn them before using them, and they do not scale up to a large number of functions since we eventually run out of keys and buttons. Hence, for an improved interface, we wanted to enable fast access to functions in a manner that would scale up to a large number of functions, while avoiding persistent modes (e.g. modal toolbar buttons), and without forcing the user to frequently move their cursor between the network they are visualizing and various menus or panels to access functions.

The solution described in this paper involves designing techniques around popup widgets. Popup widgets, invoked with kinesthetic tension (i.e. by holding down a button or stylus tip), have the well known advantages of saving screen space when not in use, of "bringing the interface to the user" so the pointing device does not need to travel as far to access controls, and of resulting in a transient mode when the widget is popped up. Popup widgets can be used in an "object-oriented" manner, where the widget is invoked over a "noun" (selected data) and used to specify a "verb" (command), sometimes in a single phrase with kinesthetic tension throughout [4]. Popup widgets have been proposed that enable fast, gestural input [11] and in some cases also allow 1D or 2D arguments to be specified [15, 7, 13] with a command, or can scale up to hundreds of commands or more [12].

In our application domain, the "noun" is a subgraph that the user selects, and the "verb" is a command to be applied to the noun. Both the noun and verb can have arguments associated with them, such as the position and dimensions of a selection rectangle, or the change in opacity or translation vector to be applied to the subgraph. In considering interaction techniques for selection and manipulation of subgraphs, we found that the previously proposed popup widgets were insufficient for our purposes, and thus we developed the novel extensions described in the next sections of this paper.

3 Techniques for Selection

We used a technique from [16] to combine rectangle and lasso selection, eliminating the need for two separate modes, keys, or buttons. The idea is to use the shape of the stroke dragged out by the user to determine whether to interpret it as a rectangle or a lasso (Figure 2). During dragging, the system keeps track of the length of the stroke along the stroke, as well as the distance between the starting cursor location and current cursor location. If the ratio of these two numbers exceeds some threshold (e.g. ≈ 2.5), then the stroke is interpreted as "mostly closed" and hence a lasso, otherwise it is "mostly straight" and so interpreted as (the diagonal of) a rectangle.

We also wished to support an additional selection method that would depend on the structure or connectivity of the network. Given any node, the user may wish to select (or deselect) all neighbouring nodes out to some number of edges. We envisaged a selection technique where the user could perform *neighbourhood-radius* selection, dragging out the desired radius (in edges) of a neighbourhood within a kind of popup slider. This idea was partly inspired by the subtree-drag-out widget in [14]. We then realized that the direction of dragging to specify a radius might be only one of several possible directions within a kind of radial menu containing additional functionality. The result is the *node-specific* radial menu shown in Figure 3, which contains additional functionality beyond neighbourhood-radius selection. To select an entire connected component, the user may either drag all the way to the east-most radius menu item, or quickly flick westward to the (connected) "Component" item as a shortcut. As shown in Figure 3 (lower right), these two items co-hilite when the cursor is over either of them, to indicate to the user that they perform the same function. We have found neighbourhood-radius selection to be very useful in situations where the visual depiction of the network is so dense that the user cannot easily point at, or even see, which nodes are the neighbours of a given node, and/or the user wishes to quickly "grab" a node and its closest neighbours, e.g., to move them somewhere else.

There are two other items in the node-specific menu that co-hilite: the north "Toggle Node" item that toggles the selection state of the node, and the center of the menu. This is because the north item is the default item, selected either with a flick to the north, or by releasing over the center of the menu. Thus, this item can be selected simply by pressing the mouse button to invoke the menu, and then immediately releasing the button. (If the user invokes the menu and decides they do not want to select anything, they must drag southward to the "Cancel" item.)

Finally, the user may also drag north-west to the "Select Only This Node" item, to select this node and deselect all other nodes.

As is well known within the HCI community, the items within radial menus and marking menus (which are equivalent techniques when only 1 level deep) can normally be selected ballistically with rapid flick motions, often much faster than in *status quo* linear menus. Although some menu items may have small labels, the user simply needs to flick in the appropriate direction within a 45 degree sector, and does not need to stop their cursor *on* the label. In the case of the linear arrangement of neighbourhood radius menu items, the user *does* actually need to point at the appropriate menu label, but the pointing performance required in that case is the same as would be required in

a 1D slider, with the advantage that the radial menu pops up wherever the user's cursor currently is.

To allow the user to invoke all of the above selection techniques with just one mouse button, the interface makes use of the information in the position of the press down event. If the user presses down *over whitespace* and starts dragging, they invoke rectangle/lasso selection. If instead they press down *over a node*, they invoke the node-specific menu in Figure 3. If they quickly press-release over a node, the node-specific menu is popped up for only a fraction of a second, and the node's selection state is toggled, by virtue of this being the default item in the radial menu.

To further enable selection based on edge-connectivity, we also implemented two additional functions: one to "grow" the selection by 1 edge, and another to "shrink" the selection by 1 edge. Growing involves adding to the selected set all nodes that are adjacent to at least one selected node. Shrinking involves removing from the selected set all nodes that are adjacent to at least one non-selected node. (These two operations turn out to not, in general, be the inverse of each other, but can still be useful.) These operations can be invoked from the hotbox, described in the next section.

4 An Enhanced Hotbox for Invoking Commands

The original hotbox [12] is a semi-transparent widget in the Maya 3D graphics software package that is popped up by holding down the spacebar with the non-preferred hand (NPH). The hotbox in Maya is essentially a 2D menu; once popped up, the preferred hand (PH) can then use the mouse to point at menu labels in the hotbox and popup smaller menus (such as marking menus). So long as the spacebar is held down by the NPH, the hotbox remains popped up, and the PH can invoke multiple commands. When the spacebar is released, the hotbox is dismissed. An important feature of the hotbox is that when it is popped up, it is centred around the cursor's current position, allowing the user to learn a mapping between relative movements of the mouse and corresponding menus. The main purpose of the hotbox in Maya is to enable menuing techniques to scale up to thousands of commands. The hotbox presented in [12], however, only contains (linear or marking) popup menus, and arguments to commands can only be specified through a disconnected mechanism such as a dialog box.

The hotbox used in our interface (Figure 4) is also semi-transparent and is invoked using a NPH keyboard key, but unlike [12], our hotbox contains additional kinds of widgets. Widgets are grouped partly according to function, and partly to have more commonly accessed commands near the centre and/or larger and easier to point to. Our current hotbox only contains ≈ 30 commands, so there is no need to nest any of them within submenus (though this could be done if more commands were added). In addition to standard push buttons and checkbox items, our hotbox also contains 1D and 2D "sliders", i.e. widgets the user presses down on and then drags on to specify a 1D or 2D argument, releasing to complete the command. These extend the hotbox in [12] by integrating specification of commands with their arguments. During dragging, the sliders do not show feedback in the form of a traditional slider's wiper; instead, the hotbox is temporarily ghosted out (Figure 5) and feedback is visible as the network is immediately updated during dragging.

Because the hotbox is always popped up centred at the cursor's current position, the 2D "Move" slider in the centre of the hotbox is the easiest to invoke, and translates all selected nodes. To the west, north, and east of it are large rectangular 2D sliders that extend to the edge of the screen so they can be aimed at ballistically like items in a radial menu. "Scale and Rotate", to the west, uniformly scales the positions of selected nodes around their centroid in response to vertical dragging, and rotates selected nodes around their centroid in response to horizontal dragging. "Scale", to the north, performs non-uniform scaling. "Move and Rotate", to the east, allows selected nodes to be translated and rotated around their centroid in a manner similar to the way that windows can be dragged and rotated in [1, Fig. 4 (left)].

The 1D sliders in the hotbox include one to adjust the opacity of selected nodes, another to adjust a per-node force parameter associated with nodes, and a third slider to customize the hotbox's own opacity

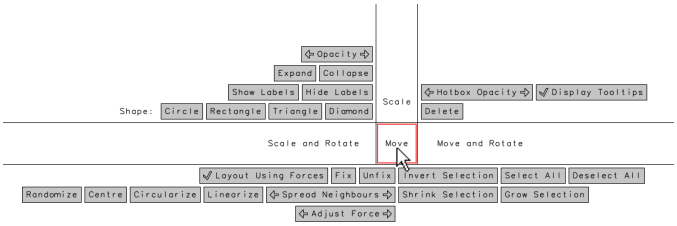

Fig. 4. When the hotbox is popped up, the cursor is initially in the central "Move" region, a 2D slider, shown here with its border hilited. To the left, the right, and above this region, are other (large and easily selectable) 2D sliders for moving, rotating, and scaling the selected nodes in various ways. The grey rectangles are buttons, checkbox items, or (if they contain arrow-shaped icons) 1D sliders.

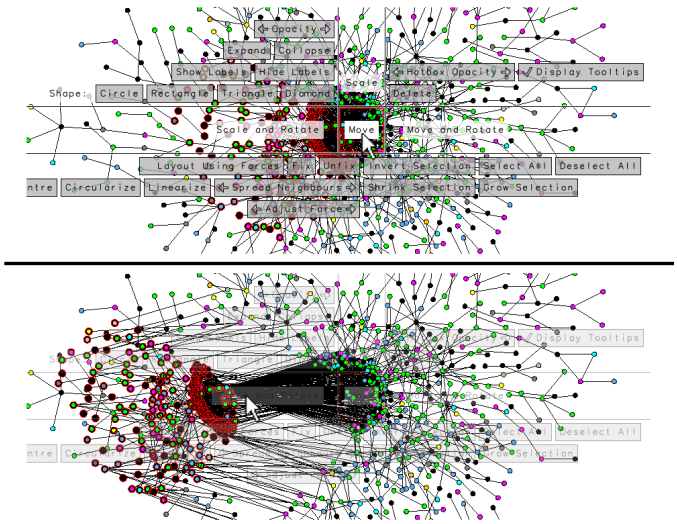

Fig. 5. The user selects a set of nodes and then invokes the hotbox (*Top*) which has a certain opacity, allowing some of the network to be visible through the controls. The user then presses the mouse button on the "Move" 2D slider causing the hotbox to automatically decrease its opacity, and the user drags left to translate the selected nodes (*Bottom*). After releasing the mouse button, the hotbox's opacity returns to normal, and the user can invoke some other widget within it. This continues until the user dismisses the hotbox by releasing the Ctrl key.

according to the user's taste. There is also a checkbox button to turn on or off tooltips within the hotbox.

The remaining buttons in the hotbox allow the other commands listed in §2 to be invoked, including global functions (e.g. the "Invert Selection" push button, and the "Layout Using Forces" checkbox to toggle force directed layout) and commands to change the shape of nodes (e.g. "Triangle", "Diamond", etc.) or apply layout patterns ("Linearize", "Circularize"). Using the layout pattern push buttons, in combination with the 2D sliders for translation, rotation and scaling, the user can flexibly and quickly change the layout of a network, as demonstrated in the video accompanying this paper.

Rather than use the spacebar with the NPH as in [12], we chose the Control (Ctrl) key for invoking the hotbox, because it is more familiar to users as a modifier key. An additional difference with our hotbox is the use of an *activation circle* during invocation (Figure 6), to enable shortcuts to the two most commonly used commands: moving a single node, and moving all selected nodes. Placing the cursor over a node, pressing Ctrl, and press-dragging the mouse moves the one node under the cursor without affecting the position or selection state of any other nodes (and, so long as Ctrl is held down, the user may release the mouse button and subsequently move other individual nodes). Doing the same over whitespace moves *all* the selected nodes.

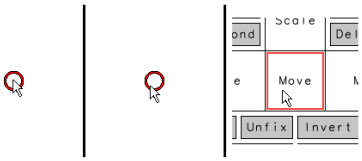

Fig. 6. Pressing the Ctrl key causes an *activation circle* to appear centred at the cursor's current position (*Left*). This circle does not follow the cursor if the cursor is moved (*Middle*). If the user presses and holds the mouse button while inside the circle, the circle disappears and the user enters a special "move" mode where dragging will move all selected nodes (or move just one node, if the cursor was initially over a single node). If, instead, the user moves the cursor all the way outside the circle without dragging, then the circle disappears and the hotbox is popped up (*Right*), with the cursor initially inside the "Move" 2D slider, afterwhich the user may press and optionally drag on any controls in the hotbox. In either case, releasing the Ctrl key (without holding down the mouse button) returns the user to their initial state, with neither circle nor hotbox displayed.

However, pressing Ctrl and moving outside the *activation circle* that appears (Figure 6), before dragging, causes the hotbox to popup. Having the "Move" slider in the centre of the hotbox, and hence under the cursor initially, reinforces the association between it and the shortcut of Ctrl-dragging without the hotbox (and also forces novice users who popup the hotbox to rehearse a close approximation of the shortcut). As for all the other widgets in the hotbox, getting to them requires an initial movement away from the centre anyway. So, although our mechanism for popping up the hotbox is different from Kurtenbach et al.'s original, there is essentially no cost incurred for expert users in supporting the shortcuts.

5 Tight Integration of the Interaction Techniques

The interaction techniques described so far only use one keyboard key (the Ctrl key) and one mouse button. We considered how to minimize the number of user actions necessary to get to various functions, as well as how to pack the most utility into these two buttons, to achieve a tight integration of interaction techniques. We noticed that, when performing rectangle/lasso selection, the user always presses down on whitespace *and then begins to drag*, which implies that press-releasing over whitespace (with no intervening drag) could be mapped to some other action or widget. We chose to map press-releasing over whitespace to a radial menu containing global selection functions ("Invert Selection", "Select All", "Deselect All"). Although these functions are already available in the hotbox, they are so frequently used that it seemed reasonable to offer a shortcut to them via this mapping.

Specifically, when the user press-releases over whitespace, the *global menu* in Figure 7 appears attached to the cursor, and follows the cursor until the user presses down a second time, at which point the menu locks in place and the user may then drag within the menu and release to complete a selection. Having the menu follow the cursor after the first press-release allows the user to comfortably reposition the menu if necessary, and also provides a strong visual cue (due to the menu's size and the movement created when the cursor is moved) to the user that they are in a special mode, and that they must press-drag-release to complete the action. A similar technique is used in [13] to create "visual tension" across two separate drags, when maintaining kinesthetic tension is not possible. Once the user is familiar with this global menu, they can, for example, quickly invert the set of selected nodes with a rapid press-release-press-flick down and to the right (in other words, a "click-and-a-half" + drag down and right).

Table 1 summarizes the actions in the interface.

To further increase the power of our interaction techniques, we considered how to allow users to build up selections incrementally. For example, many interfaces in other software allow users to select multiple objects by clicking on the first object, and then Ctrl-clicking on additional objects to add them to the selection. In such interfaces, accidentally clicking on an object without the Ctrl modifier key will deselect

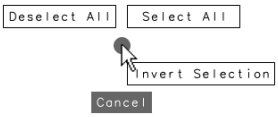

Fig. 7. The global menu, invoked by press-releasing over whitespace, can be used to change the selection state of all nodes at once. The default item is to cancel, so this menu can be easily dismissed with a 2nd press-release in case the user accidentally invokes the menu. Invoking this menu and then selecting a non-default item involves press-releasing over whitespace and then press-drag-releasing, which together feels like a "double-click-drag".

Left-press-release on node: toggle selection state of node
Left-press on node: popup node-specific radial menu
(then drag-release to select item,
or release in centre of menu to select default item)
Left-press-release on whitespace: popup global radial menu
(then press-drag-release to select item,
or press-release in centre of menu to select default item)
Left-press-drag on whitespace: rectangle/lasso selection
(then release to complete)
Ctrl-Left-press-drag on node: select and move one node
(other nodes already selected remain selected but are not affected)
Ctrl-Left-press-drag on whitespace: move all selected nodes
Ctrl-move: popup hotbox

Table 1. Seven key actions in the new user interface. Only one mouse button, and one keyboard key, are used. The first four use all combinations of {press-release, press-drag}×{on whitespace, on a node}.

all other selected objects, which can be a nuisance. In NAViGaTOR, there may be hundreds or thousands of nodes, each of which is small and may require careful work to identify and select, so such accidental deselection of nodes is not acceptable. Thus, selection and deselection of nodes should be invoked using distinct actions. At the same time, we want to provide users with the ability to build up selections incrementally, for example, using first a lasso curve to initially select one subset of nodes, and then a rectangle selection to *add* another subset of nodes, and then perhaps a neighbourhood-radius selection to *remove* all nodes within a given number of edges of some node. Essentially, we want the user to be able to combine selections using set operations such as union, set difference, and perhaps other operators.

To support this, whenever the user completes a rectangle, lasso, or neighbourhood-radius selection by releasing the mouse button, a *set operation* menu (Figure 8) then appears attached to the cursor (creating the same visual tension as used by the global menu). The user then presses, optionally drags, and releases to select a set operation within this menu, completing the interaction. For example, if the previously existing selection of nodes is S, and the user drags out a rectangle containing a set S' of nodes, then the set operation menu allows the user to "Add to Current Selection" ($S \leftarrow S \cup S'$), or "Deselect This" ($S \leftarrow S \setminus S'$), or "Select Only This" ($S \leftarrow S'$). The first option is the default, corresponding to the menu's centre, and can be selected with a very fast press-release within the menu. The other two options can be selected with sideways flicks that can be performed almost as fast.

Figure 9 shows a state-transition for our set of interaction techniques. We note that there are two kinds of commands that can be invoked multiple times without having to start over at the initial state each time. The first are the commands in the hotbox: once the hotbox is popped up, the user may use the mouse to invoke several commands in succession by simply holding the Ctrl key down to keep the hotbox popped up. The second is the command to move individual nodes (corresponding to the "move one node" state in Figure 9). Once the user has applied this command on one node, they may continue to hold down the Ctrl key to move other nodes individually, without changing the set of currently selected nodes. This is useful for quickly adjusting the positions of a few nodes before returning to manipulation of the selected set of nodes.

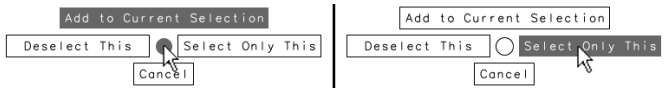

Fig. 8. The set operation menu. This menu appears immediately after a rectangle, lasso, or neighbourhood-radius has been input, effectively asking the user "what to do" with the nodes they've just specified. North, west, and east correspond to set union, set difference, and assignment, respectively. As with the other radial menus, the centre item corresponds to a default menu item, in this case the north item as indicated the co-hiliting of north and centre. So, flicking up has the same effect as releasing over the centre with no dragging (*Top*). Flicking in other directions causes the centre to un-hilite (*Bottom*).

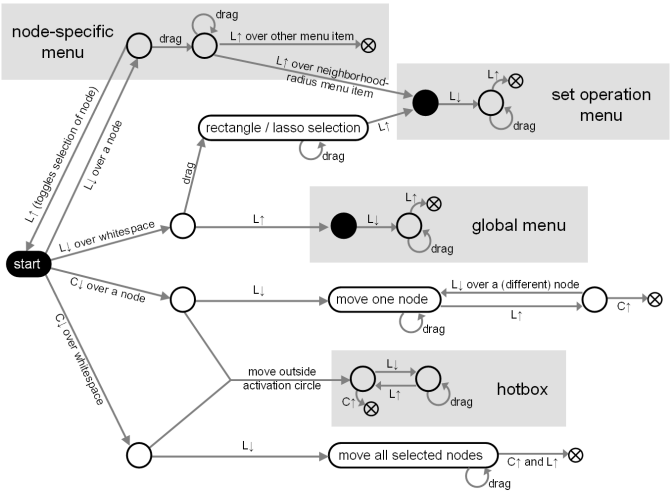

Fig. 9. A state-transition diagram of the integrated set of interaction techniques. L and C refer to the Left mouse button and the Ctrl key, respectively, and the symbols ↓ and ↑ refer to press and release events. The shaded rectangles correspond to the hotbox and each of the radial menus. The terminal state symbol ⊗ simply means that the user returns to the "start" state. Black nodes are where neither button is being held down; in the case of the two black nodes within menus, there is visual tension to compensate for the lack of kinesthetic tension.

Finally, we note that although all our interaction techniques can be invoked with just one mouse button and one keyboard key, there is nothing preventing a designer (or user) from adding additional hotkeys to quickly access functions to which they want the fastest possible shortcuts. For example, some users may like having Ctrl-A and Ctrl-D mapped to selecting *a*ll nodes and *d*eselecting all nodes, in addition to having these functions in a menu or the hotbox.

6 Relation to Previous Work

The previous work on the hotbox technique [12] only uses it to invoke menu items. Our hotbox is novel in supporting 1D and 2D sliders so that the specification of 1D and 2D arguments is integrated with the invocation of commands. In addition, when these arguments are specified during dragging, the hotbox automatically fades out so the user can easily see the immediate effect on the data being manipulated. Our hotbox is also novel in the activation circle it uses, allowing shortcuts to additional actions to be mapped to the Ctrl key.

Our node-specific radial menu (Figure 3) is novel in its use of a linear arrangement of menu items for specifying a discrete argument (the neighbourhood radius). In previous work, control menus [15] allow adjusting *continuous* parameters in a (transient) mode that is separate from menuing mode, and do not offer the values of the parameter as menu items. "Overflow" marking menu items [10] and the sectors within bullseye menus [6] contain linear arrangements of items but do not use them for controlling parameters. Our radial menus are also novel in having a default item associated with the centre of the menu that can be invoked with a shortcut single click, and in their use of co-hiliting of equivalent menu items. Finally, as a selection technique, dragging to select the radius of a neighbourhood of nodes (Figure 3) is novel, and because we have integrated it within a menu, alternative functionality can be accessed with the same invocation.

In our early design stages, we considered using bimanual (2-handed) input methods [3, 2] involving 2 pointing devices, since they allow for rich input phrasings allowing simultaneous specification of nouns and verbs. However, to facilitate deployment of the NAViGaTOR software, we wanted to assume only status-quo input devices, i.e., one pointing device and a keyboard. (Interestingly, CPN/Tools [2] has dropped support for multiple pointing devices as of version 2.0, in part due to technical difficulties in supporting bimanual input.) It is interesting that the hotbox is not just effective at scaling up to a large number of commands as shown in [12], but is also a compromise "poor man's" bimanual interface, where the 2nd hand (NPH) uses a status-quo keyboard. Although the 2nd hand cannot be used for pointing, the use of a NPH key creates kinesthetic feedback, which is critically important for avoiding persistent modes even when the PH is performing multiple drags. We also note that many pen-based interfaces assume at least one hardware button that can be operated by the NPH to augment pen input with the PH (e.g. [8]).

We designed our techniques to provide kinesthetic feedback (i.e. by requiring that the user hold down the Ctrl key and/or Left mouse button) as much as possible throughout each input phrasing, to avoid persistent modes. There are only two modes in Figure 9 where neither the Ctrl key nor the Left mouse button are held down, corresponding to two of the radial menus. As already mentioned, to compensate for the lack of kinesthetic feedback in these two states, we provide visual tension by attaching the menu to the cursor until the user presses down again. We also considered using techniques that allow for a single, unbroken drag to specify a noun and verb (or arguments) for input, maintaining kinesthetic tension across the entire phrasing. For example, during an unbroken drag to select nodes, a transition from lasso to specifying a set operation (or *vice versa*) could be induced by a crossing event, such as with a pigtail gesture [8], or by having a Control Menu [15] or FlowMenu [7] at the beginning of the drag gesture to select the set operation. Unfortunately, because nodes in the network can be densely packed, the user needs to be able to position the cursor precisely at the start *and* end of the lasso curve. This could be difficult to do if a pigtail had to be drawn immediately before or after, and using a Control Menu or FlowMenu at the start of the drag gesture would mean the cursor position would have changed by the time the user left the menu. We therefore decided it would be cleaner to break the input into multiple drags where necessary, linking the drags together with visual tension.

7 Extensibility of the Interaction Techniques

There is much room to add additional functionality to our interaction techniques without requiring any extra hardware keys or buttons. For example, many new commands could be added to the hotbox. Sliders could be added for panning and zooming the viewport. Thousands of additional commands could be added to the hotbox by nesting them within marking menus, as in [12]. Any of these menu items could be invoked with 1D or 2D arguments, by performing a marking menu stroke to select the item, then releasing (causing some visual tension to be engaged), and then press-dragging to specify the argument.

Our radial menus (Figures 3, 7, 8) could be made into marking menus with submenus containing dozens to hundreds of menu items each. (Note that we have referred to our menus as *radial* menus because they are single-level menus, and it is only in multi-level menus that marking menus' scale-invariant interpretation of mark shapes really distinguishes them from radial menus.) For example, the set operation menu (Figure 8) could have items added to it to support set intersection and symmetric difference (XOR) [19].

Additional layout patterns beyond linearize and circularize could be added to the hotbox, some of which could use 1D or 2D arguments to specify the layout. For example, we have added a 1D slider to the hotbox for creating a layout of concentric circles. When the user clicks

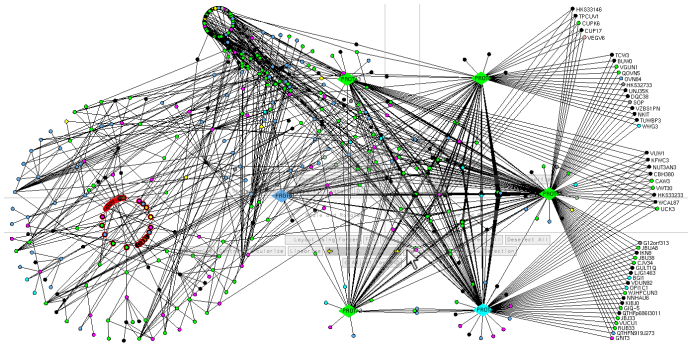

Fig. 10. A sample layout produced using our interface. 6 hub nodes were identified and "circularized" to form a hexagon. Groups of nodes adjacent to hubs were "linearized" and positioned to the right, making it easy to see that the nodes within each group share the same neighbours. The set of currently selected nodes (hilited in red on the left) are all nodes at least 6 edges away from any of the 6 hubs; this set of nodes was easily found by selecting the entire graph and then *de*selecting the neighbourhood of radius 5 centred at each hub. The hotbox (barely visible due to automatic fading) has been popped up and the user is currently dragging on the "Spread Neighbours" 1D slider, causing successive concentric circles of nodes to be laid out around the currently selected nodes.

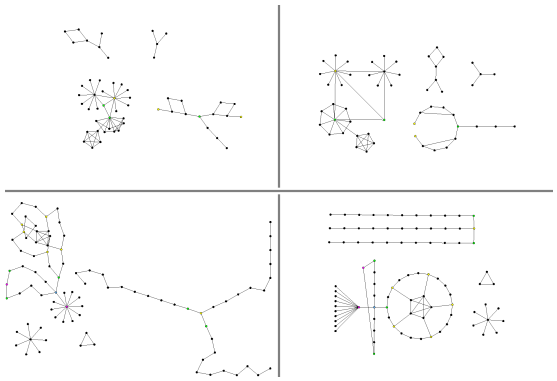

Fig. 11. *Upper and lower left:* initial layout of the network in tasks #3 and #4, respectively, as determined by force-directed layout. *Upper and lower right:* target layout that users had to achieve, in the same tasks.

down on this slider, the selected nodes are circularized, and as the user drags toward the right, additional circles of nodes that are 1, 2, or more edges away are added, in proportion to the distance dragged. The positions of nodes are smoothly interpolated toward circles as the user drags, and the user can reverse the direction of their drag to "backup" before releasing (Figure 10).

8 Initial User Testing

To evaluate our design, we performed an initial round of qualitative user testing. Our goals were to investigate how easy the interface is to learn, how well users are able to operate it, what problems become apparent, and to solicit opinions and feedback. Eight participants (4 female; 7 right-handed; 1 user a regular user of NAViGaTOR with its v1.1 interface; 5 engineers/computer scientists, 3 biologists; all daily users of computers; ranging in age from 26 to 61 (median 33)) tested the interface in individual sessions that lasted approximately 2 hours each, with audio recorded to capture verbal comments. None of the users had had any prior exposure to the new user interface.

Each session began with a pre-questionnaire, after which the user was shown the software displaying a network, was given a brief description of the force-directed layout, and was told there is some method to select nodes and move them around. The user was then asked to play with the interface while thinking aloud, during a 5 minute *discovery phase*, using only the left mouse button and the Ctrl key, to try to discover how to operate it. Next was an *instructional phase*, where the user was given a hardcopy list of the 7 actions that can be performed with the interface (Table 1), and each action was explained and demonstrated to the user who then practiced it. After the instructional phase, the user performed between 4 and 6 layout tasks (5 of the users performed all 6 tasks, but 2 users chose to stop after 5 tasks and 1 user chose to stop after 4, due to constraints in the participants' personal schedules). Finally, all users completed a post-questionnaire.

For each layout task, a network was loaded into the software and was given some initial layout by the force-directed layout algorithm, and a 2nd window displayed a static image showing the target layout to be achieved by the user by interacting with the software in the 1st window (Figure 11). Laying out each network required selection of subgraphs, rotation and translation of selections, use of "Circularize" and "Linearize", and/or fixing some or all nodes in place.

In each task, the network consisted of 4 connected components, each of which had to be laid out according to the target layout (so, users laid out between 16 and 24 components each, with most users completing all 24). Each network was made of a mixture of large and small components. The size of the networks in the 6 tasks were 30, 43, 55, 90, 67, and 111 nodes respectively, and the size of components varied between 3 and 59 nodes.

During the first two tasks, users were given many hints as to the sequence of actions they could take to lay out each component, and even after correctly laying out a component they were sometimes shown alternative techniques (e.g. alternative ways of selecting groups of nodes) to achieve the same result. This was done to expose the user to as many uses of the interaction techniques as possible. The fact that many valid strategies could be adopted by the user, with no single "correct way", was emphasized. During the later tasks, users were given many fewer hints, but were still allowed to ask questions if they wished, and were given hints if they were performing the layout in a clearly inefficient way such as manually moving nodes one-by-one.

8.1 Results

As expected, users were able to operate the interface successfully and complete the tasks they set out to perform. However, the interface took a non-negligible amount of time to learn, and we suspect the users did not have time within a 2-hour session to converge to expert-level performance. Users found the interface only somewhat "intuitive" (average: 3 out of 5), and by the end of the session only partially felt they could "operate the user interface quickly" (3.5/5). The instructional phase alone of the session took between approximately 15 and 45 minutes (median: 23), and the total time for the instructional phase together with the first two tasks (during which many hints were given) was approximately 30 to 60 minutes. The total time to perform tasks #3 and #4, when far fewer hints were given, was between 20 and 25 minutes for half the users, but two users took 50 minutes to complete these same two tasks (note that the slowest users are such that there is no clear suggestion of a correlation with gender, background, or age group). In contrast, the first author, who has extensive experience with the interface, can perform the same tasks #3 and #4 in a total of under 6 minutes, suggesting that participants had not converged to expert-level performance. User comments also suggest this: at the end of the session, one user stated *"I'm not yet really comfortable using the interface... cuz I think there was a lot to grasp, but at the same time... I don't worry about it, cuz it is self-explanatory, given the few fundamentals... explained to me... I know how to get the hotbox and everything so... I'm happy... I think if I were to repeat these tasks, it would go way faster, but already, it feels good."* Similarly, another user stated *"I'd have to practice it a lot more I think... on the other hand, I don't know how you would actually make it easier because really it does a lot of things"*

When asked what aspect (from a list of suggestions) of the tasks was most difficult, the most common responses were "remembering how to operate the user interface" (3 users) and "mentally matching elements of the network to the target layout" (3 users), suggesting that

the cognitive load on the users was due not only to learning and remembering the new interface, but also due to the inherent difficultly of the task. (Interestingly, none of the users referred to the hardcopy sheet of paper listing the 7 actions available to them, despite it being within easy reach throughout the tasks.) We suspect that a more realistic free-form browsing task, where the user is exploring the network and choosing a desired layout incrementally and opportunistically as they go, and also not switching attention between two windows, would be easier and allow for better performance.

Despite the above evidence for a non-negligible learning curve involved with the interface, users rated the interface as "easy to learn" (4.3/5), were "able to accomplish what [they] were trying to do" (4.2/5), felt "comfortable using the interface" (4.1/5), found "the user interface easy to operate" (4.1/5), were "satisfied" with the user interface (4.1/5), found the interface "fluid" (4.1/5), and liked the interface (4.4/5). All 8 users said they would want to use the same interface if they had to perform additional tasks of a similar nature.

Following are some comments made during the sessions. Some of these were made partly in response to the suggestion that an alternative interface might use pull-down menus and/or hotkeys, however the users were not "led on" in their opinions by suggesting what tradeoffs such alternatives might have. Each of these comments was made by a different user.

"I really like [the hotbox]"
"[the interface is] fun to learn"
"It's pretty complete from what I can see... I just found that not having to go anywhere else than on the [main workspace] was pretty convenient"
"I find the things [in the hotbox] easy to select... I think this [new interface is] easier to use... than a pull-down menu for example... at first I'm not sure how to do this but once I know how to select those things it becomes very easy... I like this hotbox"
"This is fun... I think the thing that I really like about it is that, you don't have to use the keys, except for the Control key... because it's faster, you don't have to remember various shortcuts... I like the fact that there are no pull-down menus; it makes it faster after the initial learning phase... It's a cool interface!"
"[It's a] refreshing experience... I like that [in] the hotbox... all the buttons stay up, cuz, the first thing I'll wanna do is say linearize this selection, and then rotate it, and then move it to just the right spot, and I like that I can just move the mouse cursor back, just over to the button [in the hotbox] instead of having to re-navigate some popup and then navigate some menu that disappears between [invocations]... [In other software] I'm constantly going back to [a menu]... and that's annoying"

During the discovery phase, no user was able to discover all 7 actions in Table 1 within the allotted 5 minutes. 7 of the 8 users successfully popped up the hotbox, however some users tried to click through the hotbox onto nodes underneath it; at least one user found the activation circle in Figure 6 too small and hence too sensitive; and two users did not realize that the hotbox was popped up by moving their cursor outside the activation circle—instead they hypothesized that the hotbox was being popped up after a temporal delay.

To aid in learning the interaction techniques, and to address the previous paragraph, we suggest the following adjustments: making the activation circle larger (hence less sensitive) by default (but allowing subsequent customization of its size), popping up an instructional tooltip if the user lingers in the activation circle for a relatively long period without moving, drawing attention to the circle by displaying an animated hilite when the cursor nears or crosses the circumference, popping up an error message if the user tries to click through the hotbox in an invalid region of it, and occasionally popping up tips (similar to a "Tip of the Day") that remind the user of actions in Table 1 that have not been used for a long time.

Subsequent to the above initial evaluation, a 9th user already familiar with the interaction techniques was observed for approximately 2 hours while using them to layout and analyze a real data set in a self-directed task. The user stated that, initially, the interaction techniques require practice, however he likes the hotbox, finds it handy to have all things in one "menu", says it saves him time (over traditional menus), and he wants to see it in future versions of the software.

9 Conclusions and Future Directions

We have presented an integrated set of interaction techniques, using only one keyboard key and one mouse button, that allow for flexible selection and manipulation of subgraphs. The novel aspects of this work are the neighbourhood-radius selection method performed by dragging, that is integrated within a modified radial menu (Figure 3), and an enhanced hotbox that integrates selection of commands with specification of 1D or 2D arguments and that is popped up using an activation circle to provide shortcuts to additional functions.

For future work, we are also interested in exploring related techniques for easing selection, especially those that use a network's structure to constrain and guide selection, and new methods for changing the network's layout with little effort, possibly using a kind intelligent "auto-completion" to facilitate input. We are also interested in applying our interaction techniques in domains beyond interactive graph layout. For example, in a 3D scene, the 1D and 2D sliders in a hotbox could be used to control parameters of a camera (position, orientation, field-of-view) or of selected objects (position, orientation, scale, color, alpha, material).

Acknowledgments

Thanks to Bowen Hui and Gord Kurtenbach for feedback; Michael Tsang, Aaron Hertzmann, and Patricio Simari for implementation tips; our users; and members of Jurisica Lab. This work was supported by funding from Genome Canada through the Ontario Genomics Institute, US Army DOD #W81XWH-05-1-0104, and IBM.

References

[1] M. Beaudouin-Lafon. Novel interaction techniques for overlapping windows. In *Proc. ACM UIST*, 2001.

[2] M. Beaudouin-Lafon and et al. CPN/Tools: A post-WIMP interface for editing and simulating coloured petri nets. In *Proc. Int. Conf. on Application and Theory of Petri Nets (ICATPN)*, pages 71–80, 2001.

[3] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: The see-through interface. In *Proc. SIGGRAPH*, 1993.

[4] W. Buxton. Chunking and phrasing and the design of human-computer dialogues. In *Proc. IFIP World Computer Congress*, pages 475–480, 1986.

[5] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[6] N. Friedlander, K. Schlueter, and M. Mantei. Bullseye! when Fitts' law doesn't fit. In *Proc. ACM CHI*, 1998.

[7] F. Guimbretière and T. Winograd. FlowMenu: Combining command, text, and data entry. In *Proc. ACM UIST*, pages 213–216, 2000.

[8] K. Hinckley, F. Guimbretière, M. Agrawala, G. Apitz, and N. Chen. Phrasing techniques for multi-stroke selection gestures. In *Proc. Graphics Interface (GI)*, 2006.

[9] Jurisica Lab. Network Analysis, Visualization, & Graphing TORonto (NAViGaTOR). http://ophid.utoronto.ca/navigator/.

[10] G. Kurtenbach. Methods and system of controlling menus with radial and linear portions. U.S. Patent #5689667. 1997 (filed 1995).

[11] G. Kurtenbach and W. Buxton. The limits of expert performance using hierarchic marking menus. In *Proc. ACM CHI*, pages 482–487, 1993.

[12] G. Kurtenbach, G. Fitzmaurice, R. Owen, and T. Baudel. The Hotbox: Efficient access to a large number of menu-items. In *Proc. ACM CHI*, 1999.

[13] M. McGuffin, N. Burtnyk, and G. Kurtenbach. FaST Sliders: Integrating Marking Menus and the Adjustment of Continuous Values. In *Proc. Graphics Interface (GI)*, pages 35–41, 2002.

[14] M. J. McGuffin and R. Balakrishnan. Interactive visualization of genealogical graphs. In *Proc. IEEE InfoVis*, pages 17–24, 2005.

[15] S. Pook, E. Lecolinet, G. Vaysseix, and E. Barillot. Control menus: Execution and control in a single interactor. In *Extended abstracts of CHI*, 2000.

[16] E. Saund, D. Fleet, D. Larner, and J. Mahoney. Perceptually-supported image editing of text and graphics. In *Proc. ACM UIST*, 2003.

[17] A. J. Sellen, G. P. Kurtenbach, and W. A. S. Buxton. The prevention of mode errors through sensory feedback. *Human Computer Interaction*, 7(2), 1992.

[18] M. Suderman and M. Hallett. Tools for visually exploring biological networks. *Bioinformatics*, 23(20):2651–2659, 2007.

[19] G. J. Wills. Selection: 524,288 ways to say "this is interesting". In *Proc. IEEE InfoVis*, pages 54–60, 1996.