1 Galant User Documentation

What follows are instructions for interacting with the Galant GUI interface.

1.1 Overview

Galant provides three major components across two windows:

1. a text window that can serve two distinct purposes –
   (a) as an editor of algorithms
   (b) as an editor of GraphML representations of graphs
2. a graph window that displays the current graph (independent of whether the text window shows an algorithm or the GraphML representation of the graph)

It is usually more convenient to edit algorithms offline using a program editor. The primary use of the text editor is to correct minor errors and to see the syntax highlighting related to macros and Galant API functions. The graph window is the primary mechanism for editing graphs. One exception is when precise adjustments node positions are desired. Weights and labels are sometimes also easier to edit in the text window.

These components operate in two modes: edit mode and animation mode. Edit mode allows the user to modify graphs – see Sec. 0.3, or algorithms – see Sec. 0.4. Animation mode disables all forms of modification, allowing the user to progress through an animation by stepping forward or backward, as described in Sec. 0.5.

1.2 Workspace

Opened graph and algorithm files are displayed in the text window, which has tabs that allow the user to switch among different algorithms/graphs. New algorithms can be created using the "page" icon at the top right of the window and new graphs using the graph/tree icon to the left of that. More commonly, algorithm and graph files are loaded via the `File->Open` browser dialog. The `File` drop-down menu also allows saving of files and editing preferences. Algorithm files have the extension `.alg` and graph files the extension `.graphml`.

1.3 Graph editing

Graphs can be edited in their GraphML representation using the text window or visually using the graph window. These editors are linked: any change in the visual representation is immediately reflected in the text representation (and will overwrite what was originally there); a change in the GraphML representation will take effect in the visual representation when the file is saved.

An improperly formatted GraphML file loaded from an external source will result in an error. Galant reports errors of all kinds (during reading of files, compilation of animation programs or execution of animations) by displaying a pop up window that allows the user to choose whether to continue (and usually return to a stable state) or quit the program.

The graph window, as illustrated in Fig. 1 has a toolbar with four sections:

1. **Graph edit mode** – this includes the *select*, *create node*, *create edge*, and *delete* buttons. Only one button is active at any time; it determines the effect of a user's interaction (mouse clicking, dragging, etc.) with the window. if there are conflicts in selection of objects, nodes with higher id numbers have precedence (are above those with lower id numbers) and nodes have precedence over edges (are above edges).

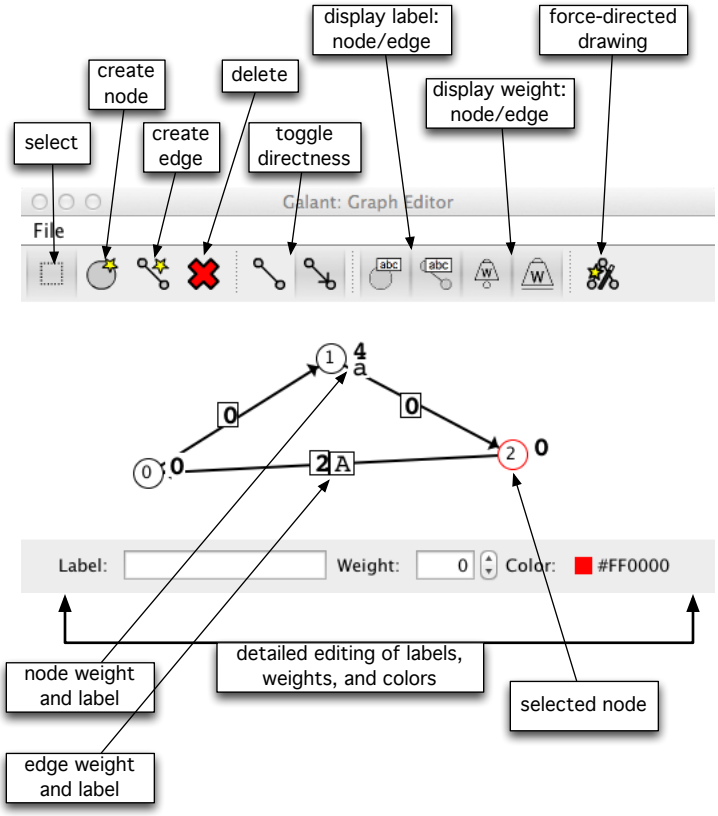

Figure 1: The Galant graph window with annotations.

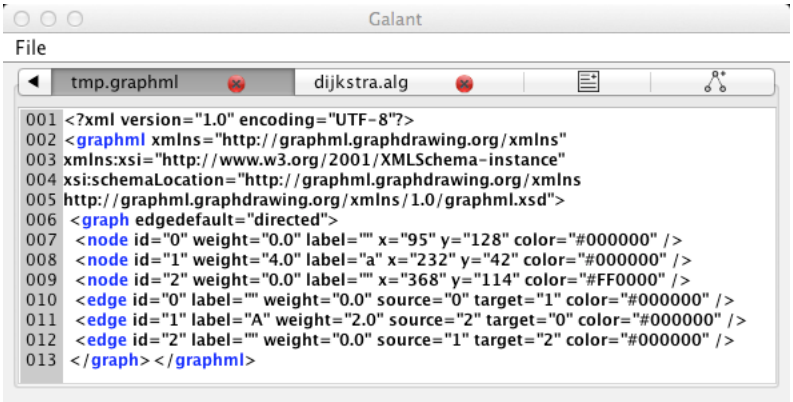

Figure 2: The text window with the GraphML representation of the graph in Fig. 1.

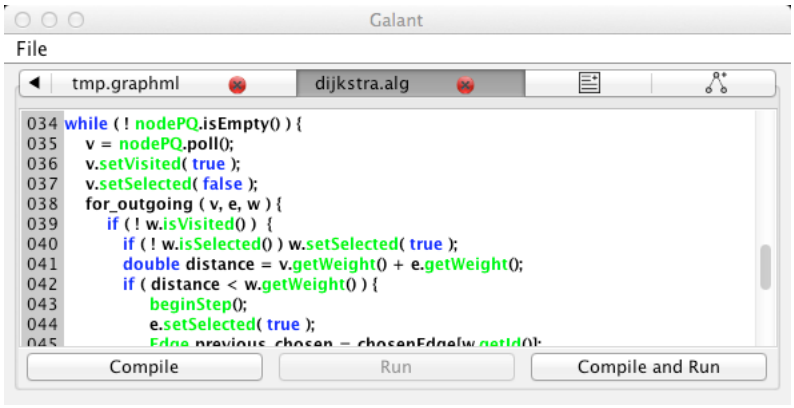

Figure 3: The text window showing Dijkstra's algorithm.

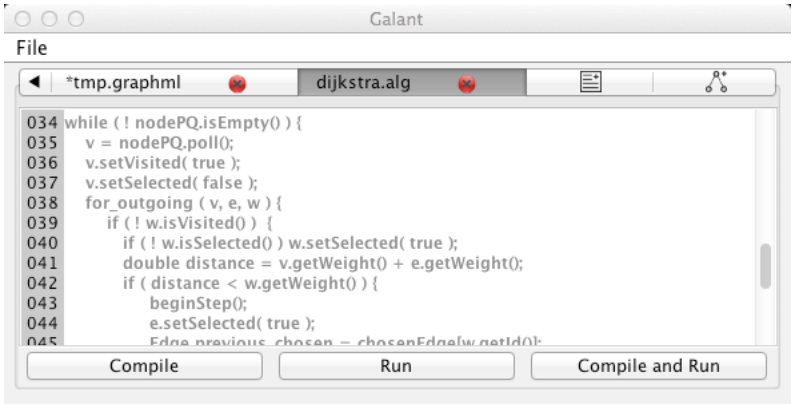

Figure 4: The text window when Dijkstra's algorithm is running.

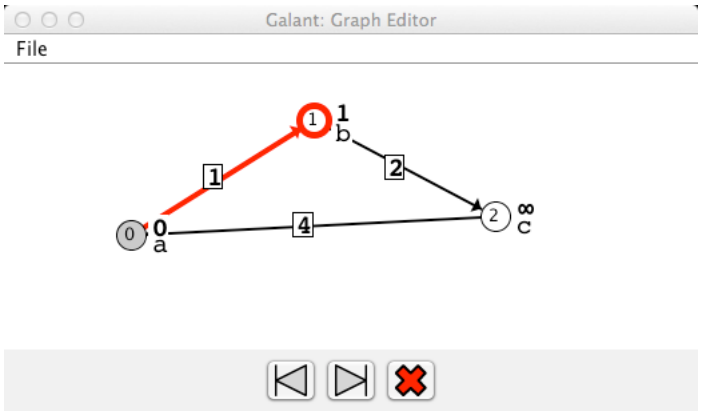

Figure 5: The graph window when Dijkstra's algorithm is running.

- *Select.* A mouse click selects the graph component with highest precedence. If the component is a node, it is shaded; if it's an edge, it turns blue. The inline editor at the bottom of the graph window allows editing of the component's label, weight, and color.
- *Create node.* A node is created at the location of a mouse click if there is not already a node there. If another node is present it is simply selected.
- *Create edge.* Two clicks are required to create an edge. The first falls on the desired source node and the second on the target node. The line representing the edge is shown after the first click. If the first click does not land on a node, no edge is created. If the second click does not land on a node, creation of the edge is canceled.
- *Delete.* A mouse click deletes the highest-precedence component at the mouse location. If a node is deleted, all of its incident edges are deleted as well.

2. **Directedness toggles** – These change both the interpretation and the display of the graph between directed and undirected. Pressing the undirected (line between two dots) button causes all edges to be interpreted as undirected: this means that, when the code calls for all incoming/outgoing edges, all incident edges are used. Undirected edges are displayed as simple lines.

   Pressing the directed (line with arrow) button causes the macros `for_incoming`, `for_outgoing`, and `for_adjacent` to have three distinct meanings (they are all the same for undirected graphs): Incoming edges have the given node as target, outgoing as source, and adjacent applies to all incident edges.

3. **Display toggles** – The four display toggles turn on/off the display of node/edge labels and node/edge weights. A shaded toggle indicates that the corresponding display is *on*. When Galant is executed for the first time, all of these are *on*, but their setting persists from session to session. Labels and weights are also all displayed at the beginning of execution of an animation. The animation program can choose to hide labels and/or weights with simple directives. Hiding is usually unnecessary – the graphs that are subjects of the animations typically have only the desired attributes set.

4. **Force directed drawing button** – Applies Hu's force directed algorithm [1] to the graph. *Caution: this operation cannot (currently) be undone.*

Keyboard shortcuts for graph editing operations are as follows:

ctrl-n – create a new node in a random position
ctrl-e – create a new edge; user is prompted for id's of the nodes to be connected
ctrl-i – do a smart repositioning of nodes of the graph; useful when positions were chosen randomly
del-n – (hold delete key when typing `n`) delete a node; user is prompted for id
del-e – delete an edge; user is prompted for id's of the endpoints
ctrl-ℓ – toggle display of node labels
ctrl-L – toggle display of edge labels
ctrl-w – toggle display of node weights
ctrl-W – toggle display of edge weights

1.4 Algorithm editing

Algorithms can be edited in the text window. The editor uses Java keyword highlighting (default blue) and highlighting of Galant API fields and methods (default green). Since the current algorithm editor is fairly primitive (no search and replace, for example), it is more efficient to edit animation code offline using a program editor – for example `emacs` with Java mode turned on. The Galant editor is, however, useful for locating and correcting minor errors. For more details on how to compose animation code, see the programmer guide (Section 1).

1.5 Animating algorithms

To animate an algorithm the code for it must be compiled and then run via the algorithm controls – the bottom tabs on the text window shown in Figs. 3 and 4. The algorithm runs on the *active graph*, the one currently displayed on the graph window. If there are errors in compilation these will show up on the console (terminal from which Galant was run) and in a dialog box that allows the user to ask for more details and decide whether to exit Galant or not. The console also displays the what the code looks like after macro replacement in case obscure errors were the result of unexpected macro expansion. Line numbers in the macro-expanded code match those of the original so that all errors reported by the Java compiler will refer to the correct line number in the original Galant code. Runtime errors also open the above mentioned dialog box.

When the user initiates execution of an animation by pushing the Run button the animation program steps forward until displays the next animation event or, if a `beginStep()` call has marked the start of a sequence of events, until it reaches the next `endStep()` call. It then pauses execution and waits for the user to decide whether to step forward, step backward, or exit. A step forward resumes execution while a step backward returns the display to a previous state. The algorithm resumes execution only when the *display state* indicated by the user's sequence of forward and backward steps ($f - b$, where f is the number of forward and b the number of backward steps) exceeds the *algorithm state*, the number of animation steps the algorithm has executed. The user controls forward and backward steps using either the buttons at the bottom of the graph window (shown in Fig. 5) or the right/left arrow keys. During the execution of the animation, all graph editing functions are disabled. These are re-enabled when the user exits the animation by pressing the red **X** button or the esc (escape) key on the terminal.

1.6 Preferences

Galant preferences can be accessed via the `File->Preferences` menu item or by using the keyboard shortcut ctrl-P (cmd-P for Mac). Preferences that can be edited are:

- Default directories for opening and saving files (`Open/Save`).
- Directory where compiled animation code is stored (`Compilation`).
- Font size and tab size for text window editing (`Editors`).
- Colors for keyword and Galant API highlighting (`Algorithm Editor`).
- Color for GraphML highlighting (`Textual Graph Editor`).
- Node radius (`Graph Display`); when the radius is below a threshold (9 pixels), node id's are not displayed; this is useful when running animations on large graphs.
- Edge width (`Graph Display`).

2 Programmer guide

This is an external document. It appears both as an appendix in the technical report and as a separate document.

References

[1] Yifan Hu. Efficient, high-quality force-directed graph drawing. *The Mathematica Journal*, 10(1), 2006. 4