Fast Downward Visualiser

1 What is the Fast Downward Visualiser?

The Fast Downward Visualiser is a plugin for the Graph Visualiser that can visualise the search process that the logging fork of Fast Downward does. This is done by processing the log file that is created by the planner during the search. The program takes one argument and that's the path to the log file. No configuration flags can be passed to the visualiser at the moment.

This visualizer is written for use in the TDDC17 and TDDD48 courses at IDA, LiU.

2 Fast Downward naming

There are a couple of things that Fast Downward does behind the scenes which causes changes in the naming compared to the domain and problem files. Among these are:

- Transformation to state variables. Fast Downward will transform all the predicates to state variables. In some cases, this yields a variable that directly corresponds to a predicate being true or false (compared to having a variable that corresponds to multiple different predicates that are mutex). This causes the name to change when transforming back to the predicate. Namely, there are now two possible names "Atom" followed by the predicate name and "NegatedAtom" followed by the predicate name. If the negated atom is in the state, then it means that the atom is false. Similarly, for state variables which represent multiple predicates there can be a case where none of them are true. This can cause the naming "none of those" to appear.

  In conclusion, Fast Downward works internally with state variables which are named "var" followed by a number. These are presented in the node information. Moreover, the predicates that are true that those variable represents are also presented in the node information, within parenthesises. However, there can be some called "NegatedAtom" or "none of those" which can safely be ignored for the purpose of the TDDC17 lab.

3 The interface

The visualiser has three main areas (see figure 1):

- The canvas for the graph. This is simply a view of how the search graph looks at a particular point in the search process.

  One can **mark** nodes in the graph by selecting them with the left mouse button just like icons in most operative systems. Once the nodes are marked one can **move** them by using the middle mouse button (or `alt`+the left mouse button). However, the nodes might also be **moved automatically**, depending on which algorithm is used to place the nodes.

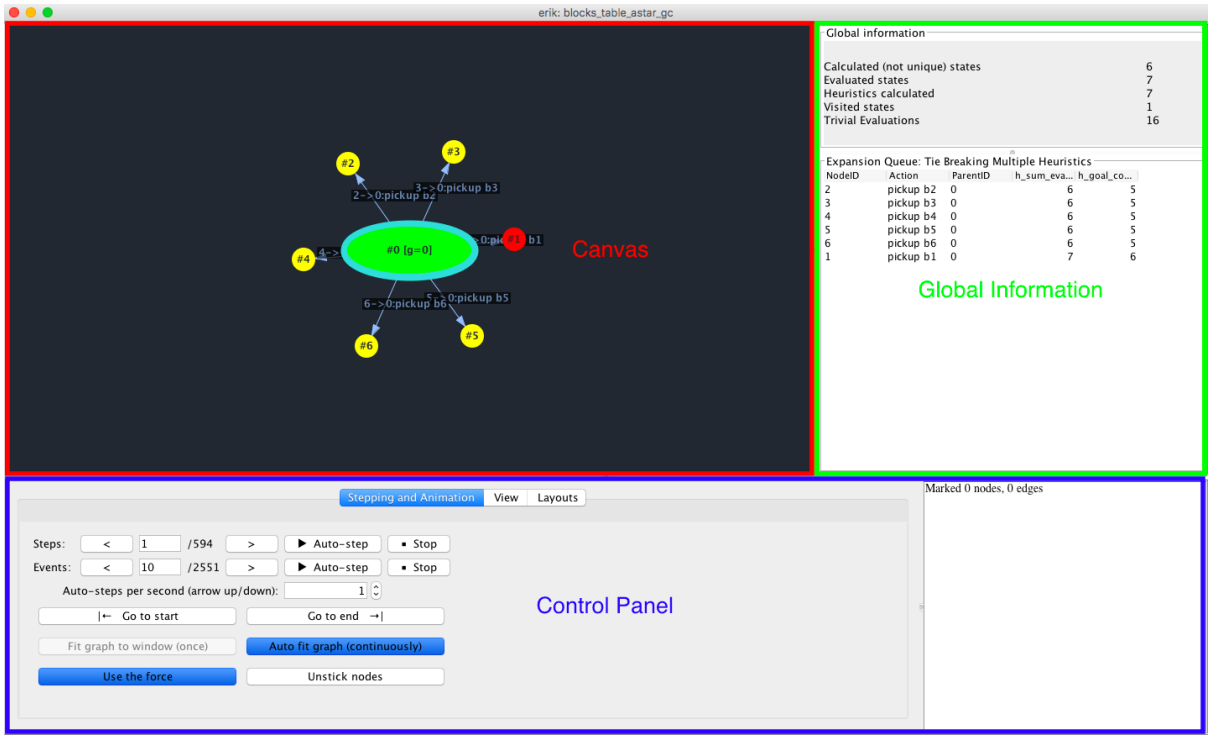

Fig. 1: Overview of the interface of the Fast Downward Visualiser.

If no nodes are selected, one can **move the view port** of the graph ("scroll it") by pressing the middle button and moving it. (If you have selected nodes or edges, this won't have the expected effect – to get rid of the selection, draw an outline where there are no nodes or edges.)

The view port can also be moved using the arrow keys, as long as you have clicked in the canvas to "activate" it.

Finally, it's possible to **zoom in or out** by using the scroll wheel.

- **Global information**. This section, at the right hand side of the interface, shows global information about the graph.

  The upper part contains information such as how many nodes that have been expanded, how many nodes that have been visited and how many heuristic values that have been calculated.

  The lower part displays the **queue** that the planner used to select which node to visit next. Here you can see which alternatives the planner was considering at the given point in the search process, and how those alternatives were ranked.

  - **NodeID**: The index number of the search node (state), in order of node generation.
  - **Action**: The action that was applied in order to generate the new state.
  - **ParentID**: The ID of the parent state
  - **h_goal_count**: The value of the $h_{\text{goalcount}}$ heuristic for the state that you get after applying the given action. There can of course be other heuristics shown instead of h_goal_count, depending on the parameters you gave to Fast Downward!

– **h_sum_eval(g, goal_count)**: This column shows the cost of $f = g + h$, the cost g of reaching the given state plus the estimated cost h of reaching the closest goal state from there.

As you should know, some algorithms (such as $A*$) base their heuristic decisions on the value of $f = g + h$. Others, such as greedy forms of search, ignore g and only care about h.

If multiple queues are used (for example the alternating queue of Fast Downward or a preferred queue) then this view will show the queue that will be used to select the next node. Note that the view doesn't support any randomisation in the queues.

- Control panel and selection information. This area consists of the controller for the visualiser and information about the current selection in the graph. The controller will be explained in more details below.

3.1 Control panel

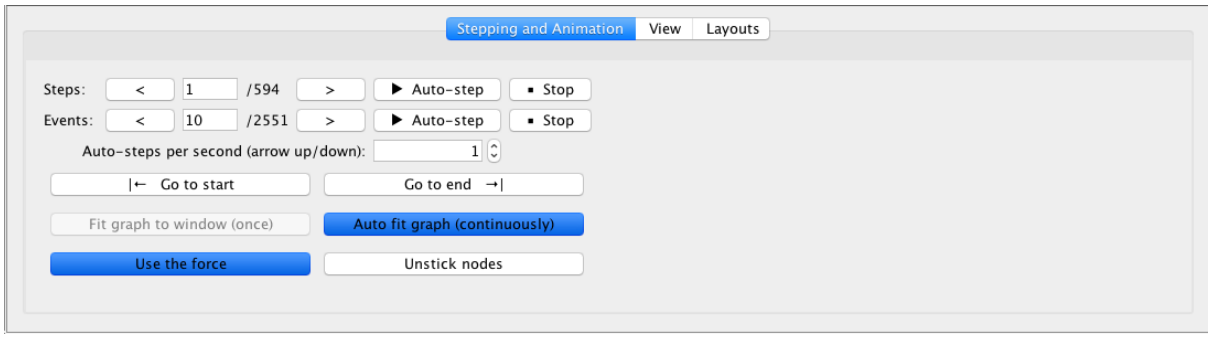

Fig. 2: The control panel for the Fast Downward Visualiser.

The control panel is divided into multiple sub areas (see figure 2): Stepping and Animation, View, and Layout.

3.1.1 Stepping and Animation

This tab handles how to step through the search process. There are two different ways of stepping. The first is by stepping in discrete **time steps**. This is the recommended way for all tested search engines except for the enforced hill climbing since one step in that equals one phase (see the documentation for Fast Downward with logging). For all other tested search engines, one time step means going through all the events until the next node is expanded. This leads to the second option for stepping through the search process: **events**. An event in the search process is defined as one log entry or a group of log entries. This could be everything from visiting a node to reporting an updated heuristic value or boosting the preferred queues. However, the log entries are most often grouped together so that each event should contain visiting a node or expanding a node. For most cases this is completely unnecessary to use.

The stepping through the time points (or events) can be done manually or automatically. The most simple way is to use the arrow buttons to go forward or backward one time point/event. One can also enter the event number or the time point and press enter to jump to a specific point. The final manual way of progressing or regressing is to use

the "Go to start" and "Go to end" button. However, one should be warned that using a jumping method (to the end, start or specified time point or event) can take some time for larger graphs. This is because all the events to the selected event/time point have to be applied or undone, one by one.

Automatic progression can be done by using the "Auto-step" and "Stop" buttons. These options step forward in time points and events, respectively. The frequency for the stepping can be set in the "Auto-steps per second" field. However, to update the speed one has to **press enter after the value has been updated**.

There are two options of **automatically zooming** so that the graph fits the canvas. The first button is the "Fit graph to window" button which forces the graph in its current form to fit the graph. "Auto fit graph (continuously)", the second button, makes this behaviour automatic (and may be active from the start). This means that the zoom and positioning will always be updated so that the graph fits the canvas even as it is modified and moved.

Also, there are a few additional commands, mostly related to how nodes are positioned on screen in order to avoid overlaps.

- **Rotate**: Rotates the graph 90 degree around the origin (not possible with the default force directed layout).
- **Use the force**: Turns on the forces that (a) pull nodes closer to each other if they are connected by edges, and (b) push nodes apart if they are too close to each other. This can be useful if you want nodes to stay still so you can rearrange them.
- **Stop using the force**: Stop moving nodes according to node/edge forces.
- **Unstick nodes**: Some layouts that incrementally move nodes will eventually stop doing so, when they seem to be close to reaching an equilibrium. If the nodes got stuck before you wanted them to, use this button to unstick them so they continue moving.

3.1.2 View

These are options to configure how the shown graph components should look. For example:

- The **non-cheapest edges** option decides how to show edges that aren't part of the cheapest route to a node. These edges were at some point generated by the planner, but then it found a better (cheaper) path to reach the same state and the old edge was discarded.

  Hidden (default): Ignore the edge completely.

  Dashed: Show the edge as dashed and ignore it when doing force-based layout.

  Filled: Show as an ordinary edge, and apply forces along the edges as well. This can have a large impact on the look of your graph.

- The **node colouring option** is very important as it decides how nodes in the graph are coloured, which can provide a lot of information at a glance.

  The "Visited/Expanded" option simply differentiates between if a node is expanded or visited while the rest of the options change the colour of the node between green

and red depending on some value. These values consists of some common values for all graphs (for example, the expansion order and the g-value) as well as all the heuristics that has been used in the search process.

3.1.3 Layouts

This tab shows the possible selection of layouts. That is, the selection of algorithm to place the nodes on the canvas. The layouts that are available for the Fast Downward search graphs are:

- **Force Directed**: This is an iteratively improving placement of nodes based on the algorithm family of forced-directed drawing. In essence, nodes will repel each other and each edge will pull the connected nodes together. This causes the nodes to move until a stable position is found.

  This is a somewhat costly algorithm and to make it a bit more usable for larger graphs some simplifications have been done. Therefore, the nodes that stay almost still for a longer period of time will stop moving completely. Unfortunately they can in some cases stop too early and become quite badly placed. If this happens one can use the option to unstick nodes (see layout options) which causes the algorithm to evaluate their positions again.

  One can also **move** the nodes in the graph to help the positioning. To do this, first select the nodes you want to move (left mouse button), then pull them somewhere using the middle mouse button. Note that unless you deactivate the forces or switch to the free layout, your nodes will to a large extent be pulled back when you release the middle button! Still, this can be enough to fix cases where nodes get entangled or stuck in the wrong place.

  The force directed layout works well to display graphs if one hides all the cheaper edges or shows them as dashed.

- **Free** This layout simply places all the nodes until the users moves them to a wanted placement.