XGV
A Graphical Tool to Analyze Planning Graphs

Frank Rittinger
Institute for Computer Science
Albert Ludwigs University
Am Flughafen 17
79110 Freiburg, Germany
rittinge@informatik.uni-freiburg.de

May 16, 1997

1 Motivation

XGV is a program to view the planning graphs created by the interference progression planner IP^2[2], whose kernel algorithm is an extension of Graphplan[1] to operators with context-sensitive effects.

The original graphical interface developed for Graphplan was the first easily usable graphical interface and since it was written with X lib it was highly portable. In addition the program was capable of showing the animated graph creation.

Graphplan was missing certain features we found essential to inspect planning graphs containing actions with context-sensitive effects.

Graphplan downscales and overlaps the whole graph until it fits the screen, but as a result of the reduced size no details can be exerted from the graph (see Figure 1). It is impossible to get any extra information on a particular node e.g., the preconditions of an action or the exclusives of a node in that level. Another disadvantage of the program is that the plan is not displayed in the graph.

XGV was written to go beyond those limitations. First, to make the graph more accessible, we decided to display only a clip window of the graph. Secondly, we reduced the information in the displayed graph to a minimum but at the same time made it interactive to give the user the possibility to scroll through the whole graph. This gives the user the opportunity to get all the information about a particular node by simply clicking on the node. Some features supported by *XGV* are:

- highlighting the plan if one was found
- highlighting exclusives and non-exclusives
- showing conditions of effects

To reduce the size of the graphs noops are not displayed. We found that in a large graph most action nodes are noops. Therefore we made a compromise between graph size and completeness of the graph. We think that this was justified by the fact that noops do not add new facts but just copy the facts from level to level and that they are never contained in the plan.

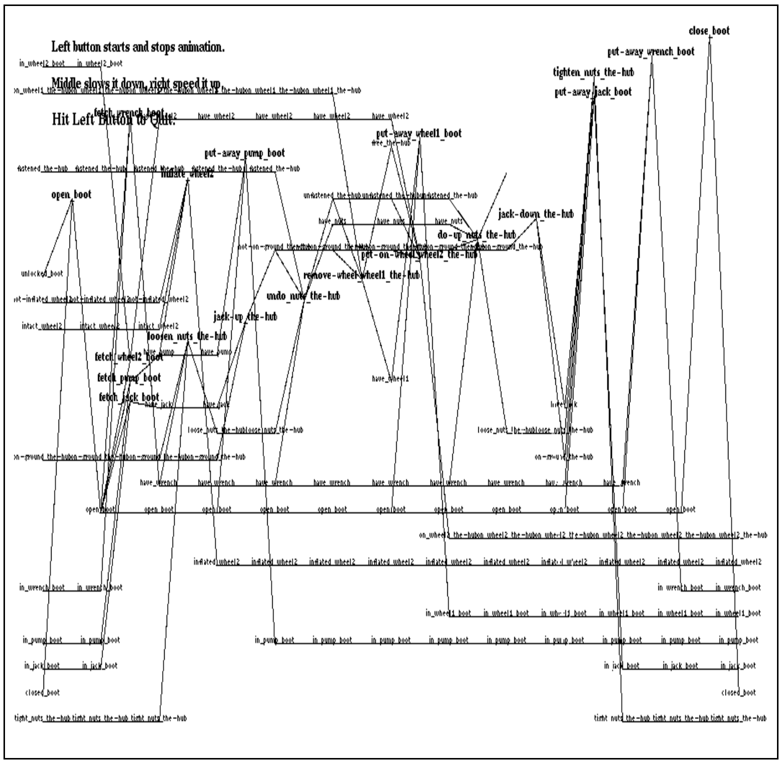

Figure 1: Graphical interface of Graphplan

With these innovations we created a powerful tool for inspecting planning graphs and plans and for finding errors in the domains and operator files.

In the upcoming Section 2, the installation of *XGV* is explained. In Section 3, the usage of the program is illustrated, followed by a discussion of some examples showing how *XGV* can be employed to debug planning task specifications. The format of the input files is displayed in Section 5.

2 Installation

The source code is available as a gziped tar-file via
`http://www.informatik.uni-freiburg.de/~koehler/ipp.html`
The distribution consists of the following files:

Makefile	makefile for SUN SPARCstation
README	installation instructions
draw.c	routines to draw the graph
file.c	routines for the file- and help-menu
load_files.c	the function to load the graph
options.c	routines for the options-menu
search.c	routines to search a node
xgv.c	the main function to setup the program
xgv	the binary compiled for a SUN SPARCstation
xgvS.h	the structure definitions
shuffle.facts	the facts file of a sample graph
shuffle.opers	the actions file of a sample graph

There is a header file for each source file with the function prototypes.

A graph is always saved in two files which must have the same name and the postfixes ".opers" and ".facts", see Section 5.

To install the distribution just unpack the file in the destination directory. If you want to recompile the sources modify the makefile for your system e.g., libraries and include files, and type `make`. You can clean your directory after compilation with `make clean`. This removes all object and temporary files.

The binary in the distribution was compiled on a SUN SPARCstation running SunOs 5.4 and X11R6 and is linked with the Motif 1.2 library. I also compiled the program without any modifications on a SGI INDY running IRIX and on a PC running LINUX.

3 How to use *XGV*

To invoke *XGV*, simply type `xgv` on the command-line. The program understands all the usual X command-line options. Additionally the name of a graph file can be specified. This must be the filename without the postfix, e.g. `tyre` instead of `tyre.facts` or `tyre.opers`. Since the input files are split into two files you only have to specify one file and the other will be loaded accordingly. The same is true if you load a new graph from within *XGV*.

To distinguish action and fact levels, we enclosed actions in arcs and facts in boxes. The names of the nodes in the graph may not be displayed in full length, because the names are truncated at a fixed length (currently set to 12 in xgvS.h). This helps to keep the graphs smaller and to avoid overlapping nodes. You can get the full name of a node by selecting this node.

3.1 The Menubar

- File
  - Load: Load a new graph. Select a `".opers"` or a `".facts"` file. The other file of the same graph will be loaded automatically.
  - Quit: Exit *XGV*.
- Options
  - Clear graph: All edges and highlighted nodes will be deleted and the initial graph will be redrawn.
  - Show plan: All actions belonging to the plan will be highlighted in red and their edges are drawn.
- Search
  - Search Action: You can select an action name in the popup-dialog and all occurrences of this action in the graph will be highlighted in blue.
  - Search Fact: Same as for actions.
- Help
  - Help: Gives help on the mouse functions and displays the URL of this document.

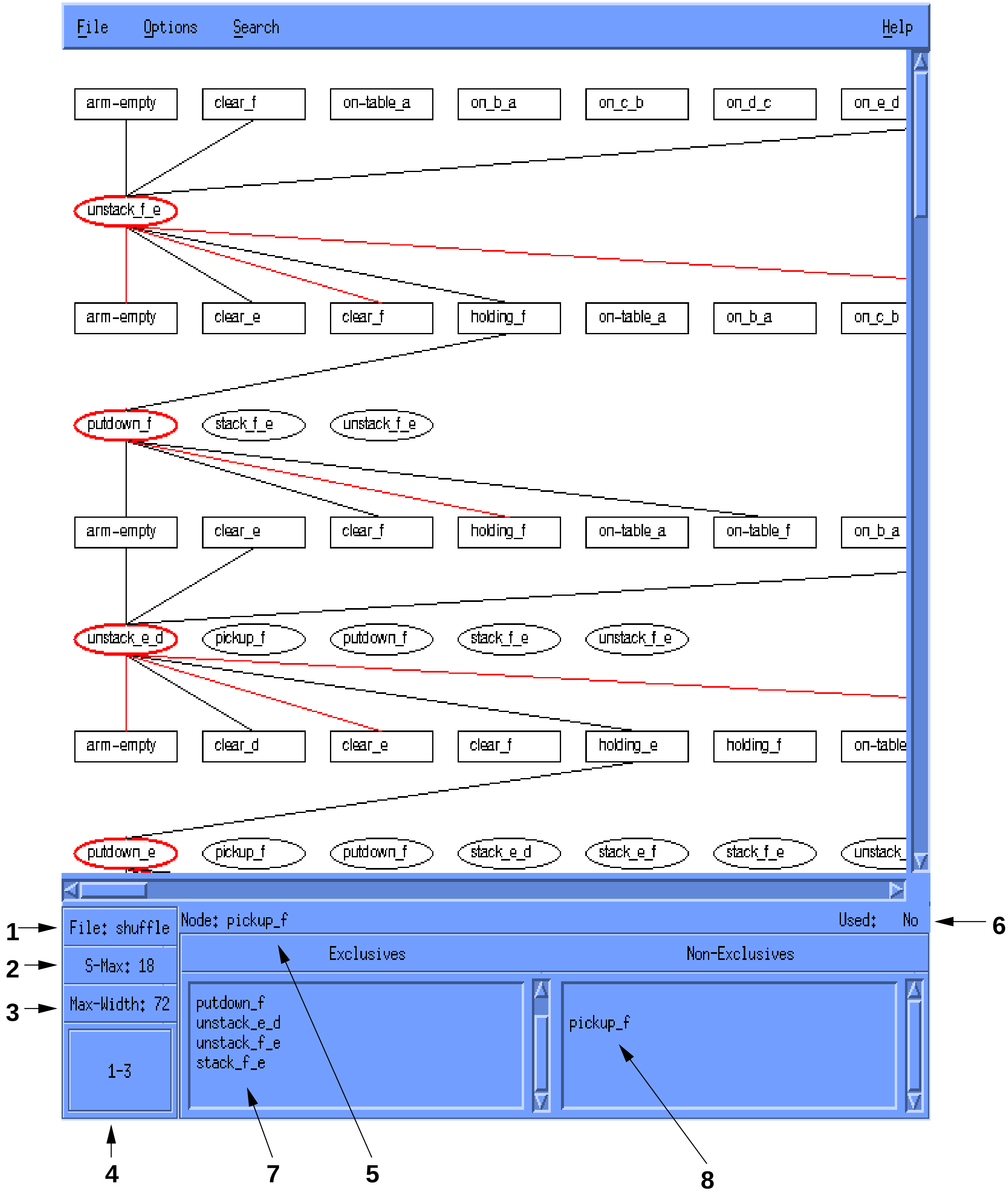

Figure 2: Graph of shuffle.[opers,facts]

3.2 The Mouse

- Left button: Show information on the selected node in the information area. The precondition edges (black), ADD edges (black) and DEL edges (red) are drawn. The node itself and all non-exclusive nodes on the same level are highlighted with thick frames.
- Middle Button: Erase all edges of this node and reset all highlighting in this level.
- Right button: Popup a dialog that shows the full node-name, the preconditions and the ADD / DEL effects together with their effect conditions in the form "*effect* <= *condition*".

3.3 The information area

When a node is selected with a mouse click the following information is displayed in the various widgets (the numbers refer to the black thick numbers in Figure 2):

1. Filename: Show the name of the loaded graph.
2. S-Max: The maximum number of action levels.
3. Max-Width: The maximum number of nodes per level found in the graph.
4. Coordinates: The level and position of the selected node. For example, "1-3" means 3. node in the 1. level. Action level coordinates range from [1-1,...,n-k] and proposition levels range from [0-1,...,n-k].
5. Node-name: Displays the full name of the selected node.
6. Used: "Yes" if the selected node is used in the plan, "No" otherwise.
7. Exclusives: The nodes in this level that have been marked by the planner as mutually exclusive to the selected node.
8. Non-exclusives: The nodes in this level that have been marked by the planner as non-exclusive to the selected node.

4 Hints on Debugging IP^2 with XGV

We found that *XGV* is a helpful tool to debug planning task specifications. Very often, a task is unsolvable by the planner because of simple typing errors. Inspecting the resulting graph (IP^2 also writes a graph when no plan was found) can help to detect such typing errors quickly:

1. action list:
   (a) no ground instance (action) of an operator occurs
      - wrong type declaration for operator parameter
      - no object of this type in domain file
   (b) a special ground instance is missing
      - missing object in type declaration
2. fact list

- fact does not appear in graph
  - missing ADD effect in operator or misspelled name
  - typing error in a precondition of actions that can add this effect

5 Format of input files

When IP^2 is run with the `-W` option a ".opers" and a ".facts" file are written to disk. Because IP^2 keeps actions and facts in two separated arrays it is easier to have one file for the actions and one for the operators. The two files have a similar format and the following information is stored for a node (see Figure 3 and 4):

- level
- node name
- if it is used in the plan
- the nodes that are marked exclusive in that level
- for actions:
  - preconditions
  - ADD / DEL effects with their conditions
- for facts:
  - the actions it is precondition of
  - the effects it is condition of

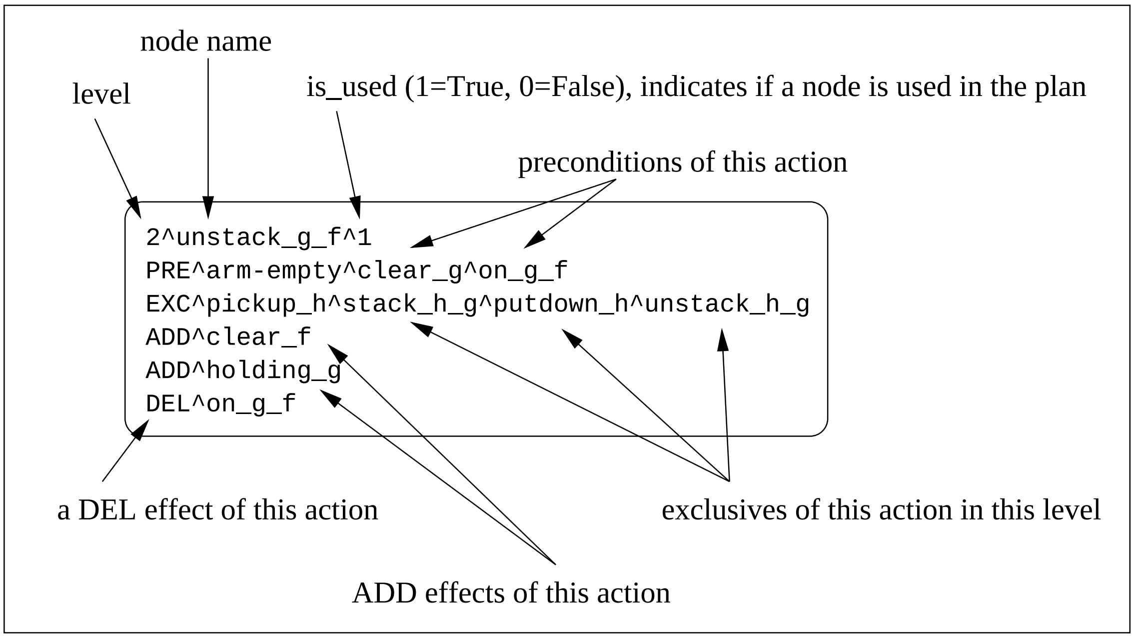

Figure 3: An example node of `shuffle.opers`.

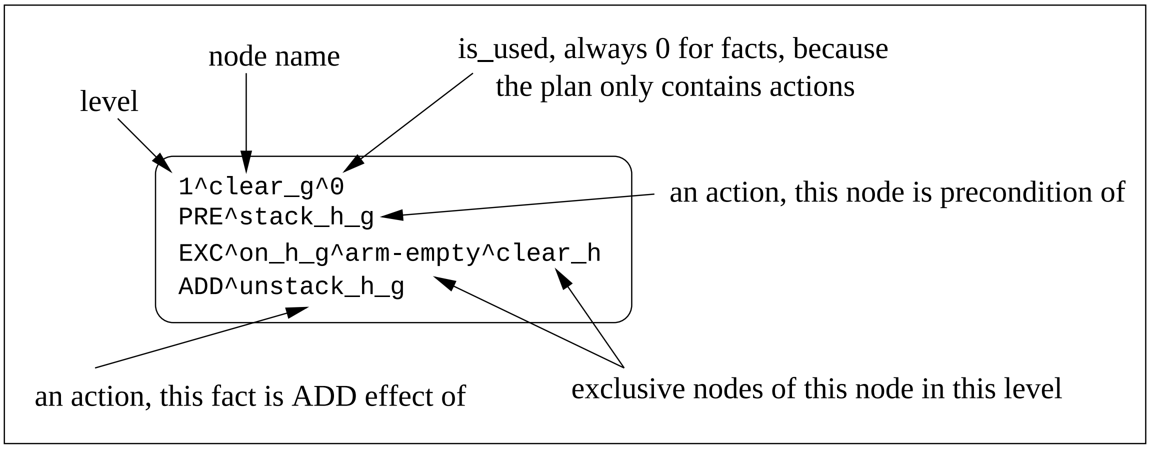

Figure 4: An example node of `shuffle.facts`.

6 Limitations and Open Problems

One major problem is the size of the graphs. The whole graph is drawn into one large pixmap in memory and the visible part is copied to the drawing-area. The X server can only allocate pixmaps with a limited size and for some graphs this size is too small. If you try to load such a graph, the X server complains and the program crashes. Currently all graphs created by IP^2 can be drawn, although the performance of the system decreases quickly with growing graph size. A graph size the system can still handle is approximately 5 by 2 meters, which corresponds to 26 levels and a maximum of 128 nodes per level. This actually means that you can view graphs that are two to three times larger, because noops are not displayed.

Another limitation is that most resources are hard-coded. The fonts and strings are all specified in the source code and cannot be changed in a resource file.

We are also thinking about having one file for the whole graph and not a separate one for facts and one for actions, because it would make managing the graph files easier.

We also suggest to use short names in "*.fct" and "*.ops" files because then the node names will not be truncated, as mentioned earlier.

7 Acknowledgments

I would like to thank the KI-Group at the Department of Computer Science at the University of Freiburg and especially Steffen Gutmann for his tips on C and Motif programming.

References

[1] A. Blum and M. Furst. Fast planning through planning graph analysis. pages 1636–1642.

[2] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. Technical report, University of Freiburg, Institute of Computer Science, 1997. available over http://www.informatik.uni-freiburg.de/~ koehler/ipp.html.