NAME

libpack − support for connected components

SYNOPSIS

```
#include <graphviz/pack.h>

typedef enum { l_clust, l_node, l_graph} pack_mode;

typedef struct {
    unsigned int margin;
    boolean      doSplines;
    pack_mode    mode;
    boolean*     fixed;
} pack_info;

point*     putGraphs (int, Agraph_t**, Agraph_t*, pack_info*);
int        shiftGraphs (int, Agraph_t**, point*, Agraph_t*, int);
int        packGraphs (int, Agraph_t**, Agraph_t*, pack_info*);
int        packSubgraphs (int, Agraph_t**, Agraph_t*, pack_info*);

pack_mode  getPackMode (Agraph_t*, pack_mode dflt);
int        getPack (Agraph_t*, int, int);

int        isConnected (Agraph_t*);
Agraph_t** ccomps (Agraph_t*, int*, char*);
Agraph_t** pccomps (Agraph_t*, int*, char*, boolean*);
int        nodeInduce (Agraph_t*);
```

DESCRIPTION

libpack supports the use of connected components in the context of laying out graphs using other *graphviz* libraries. One set of functions can be used to take a single graph and break it apart into connected components. A complementary set of functions takes a collection of graphs (not necessarily components of a single graph) which have been laid out separately, and packs them together moderately tightly. The packing is done using the polyomino algorithm of K. Freivalds et al.

As this library is meant to be used with *libcommon*, it relies on the *Agraphinfo_t*, *Agnodeinfo_t* and *Agedgeinfo_t* used in that library. The specific dependencies are given below in the function descriptions.

Creating components

Agraph_t** ccomps (Agraph_t* g, int* cnt, char* pfx)

The function *ccomps* takes a graph *g* and returns an array of pointers to subgraphs of *g* which are its connected components. *cnt* is set to the number of components. If *pfx* is non-NULL, it is used as a prefix for the names of the subgraphs; otherwise, the string ‘‘_cc_’’ is used. Note that the subgraphs only contain the relevant nodes, not any corresponding edges. Depending on the use, this allows the caller to retrieve edge information from the root graph.

The array returned is obtained from *malloc* and must be freed by the caller. The function relies on the *mark* field in *Agnodeinfo_t*.

Agraph_t** pccomps (Agraph_t* g, int* cnt, char* pfx, boolean* pinned)

This is identical to *ccomps* except that is puts all pinned nodes in the first component returned. In addition, if *pinned* is non-NULL, it is set to true if pinned nodes are found and false otherwise.

int nodeInduce (Agraph_t* g)

This function takes a subgraph *g* and finds all edges in its root graph both of whose endpoints are in *g*. It returns the number of such edges and, if this edge is not already in the subgraph, it is added.

int isConnected (Agraph_t* g)

This function returns non-zero if the graph *g* is connected.

Packing components

point* putGraphs (int ng, Agraph_t gs, Agraph_t* root, pack_info ip)**

putGraphs packs together a collection of laid out graphs into a single layout which avoids any overlap. It takes as input *ng* graphs *gs*. For each graph, it is assumed that all the nodes have been positioned using *pos*, and that the *xsize* and *ysize* fields have been set. The packing is done using the polyomino-based algorithm of Freivalds et al. This allows for a fairly tight packing, in which a convex part of one graph might be inserted into the concave part of another.

If *root* is non-NULL, it is taken as the root graph of the subgraphs *gs* and is used to find the edges. Otherwise, *putGraphs* uses the edges found in each graph *gs[i]*.

The granularity of the polyominoes used depends on the value of *ip->mode*. If this is *l_node*, a polyomino is constructed to approximate the nodes and edges. If this is *l_clust*, the polyomino treats top-level clusters as single rectangles, unioned with the polyominoes for the remaining nodes and edges. If the value is *l_graph*, the polyomino for a graph is a single rectangle corresponding to the bounding box of the graph.

If *ip->doSplines* is true, the function uses the spline information in the *spl* field of an edge, if it exists. Otherwise, the algorithm represents an edge as a straight line segment connecting node centers.

The parameter *ip->margin* specifies a boundary of *margin* points to be allowed around each node. It must be non-negative.

The parameter *ip->fixed*, if non-null, should point to an array of *ng* booleans. If *ip->fixed[i]* is true, graph *gs[i]* should be left at its original position. The packing will first first place all of the fixed graphs, then fill in the with the remaining graphs.

The function returns an array of points which can be used as the origin of the bounding box of each graph. If the graphs are translated to these positions, none of the graph components will overlap. The array returned is obtained from *malloc* and must be freed by the caller. If any problem occurs, *putGraphs* returns NULL. As a side-effect, at its start, *putGraphs* sets the *bb* of each graph to reflect its initial layout. Note that *putGraphs* does not do any translation or change the input graphs in any other way than setting the *bb*.

This function uses the *bb* field in *Agraphinfo_t*, the *pos*, *xsize* and *ysize* fields in *nodehinfo_t* and the *spl* field in *Aedgeinfo_t*.

int shiftGraphs (int ng, Agraph_t gs, point* ps, Agraph_t* root, int doSplines)**

The function *shiftGraphs* takes *ng* graphs *gs* and a similar number of points *ps* and translates each graph so that the lower left corner of the bounding box of graph *gs[i]* is at point *ps[i]*. To do this, it assumes the *bb* field in *Agraphinfo_t* accurately reflects the current graph layout. The graph is repositioned by translating the *pos* field of each node appropriately.

If *doSplines* is non-zero, the function also translates the *coord* field of each node and the splines coordinates of each edge, if they have been calculated. In addition, edge labels are repositioned. If *root* is non-NULL, it is taken as the root graph of the graphs in *gs* and is used to find the edges. Otherwise, the function uses the edges found in each graph *gs[i]*.

It returns 0 on success. Note also that the bounding boxes of all graphs are left unmodified.

This function uses the *bb* field in *Agraphinfo_t*, the *pos* and *coord* fields in *nodehinfo_t* and the *spl* field in *Aedgeinfo_t*.

int packGraphs (int ng, Agraph_t gs, Agraph_t* root, pack_info* ip)**

This function takes *ng* subgraphs *gs* of a root graph *root* and calls *putGraphs* with the given arguments to generate a packing of the subgraphs. If successful, it then invokes *shiftGraphs* to apply the new positions. It returns 0 on success.

int packSubgraphs (int ng, Agraph_t gs, Agraph_t* root, pack_info* ip)**
This function simply calls *packGraphs* with the given arguments, and then recomputes the bounding box of the *root* graph.

Utility functions
The library provides several functions which can be used to tailor the packing based on graph attributes.

pack_mode getPackMode (Agraph_t* g, pack_mode dflt)
This function returns a *pack_mode* associated with *g*. If the graph attribute *"packmode"* is "cluster", it returns *l_clust*; for "graph", it returns *l_graph*; for "node", it returns *l_node*; otherwise, it returns *dflt*.

int getPack (Agraph_t* g, int not_def, int dflt)
This function queries the graph attribute *"pack"*. If this is defined as a non-negative integer, the integer is returned; if it is defined as "true", the value *dflt* is returned; otherwise, the value *not_def* is returned.

SEE ALSO

dot(1), **neato**(1), **twopi**(1), **libgraph**(3)
K. Freivalds et al., "Disconnected Graph Layout and the Polyomino Packing Approach", GD0'01, LNCS 2265, pp. 378-391.

BUGS

The packing does not take into account edge or graph labels.

AUTHORS

Emden Gansner (erg@research.att.com).