The igraph package

Description

igraph is a library and R package for network analysis.

Introduction

The main goals of the igraph library is to provide a set of data types and functions for 1) pain-free implementation of graph algorithms, 2) fast handling of large graphs, with millions of vertices and edges, 3) allowing rapid prototyping via high level languages like R.

Igraph graphs

Igraph graphs have a class ‘igraph’. They are printed to the screen in a special format, here is an example, a ring graph created using make_ring:

```
IGRAPH U--- 10 10 -- Ring graph
+ attr: name (g/c), mutual (g/x), circular (g/x)
```

‘IGRAPH’ denotes that this is an igraph graph. Then come four bits that denote the kind of the graph: the first is ‘U’ for undirected and ‘D’ for directed graphs. The second is ‘N’ for named graph (i.e. if the graph has the ‘name’ vertex attribute set). The third is ‘W’ for weighted graphs (i.e. if the ‘weight’ edge attribute is set). The fourth is ‘B’ for bipartite graphs (i.e. if the ‘type’ vertex attribute is set).

Then come two numbers, the number of vertices and the number of edges in the graph, and after a double dash, the name of the graph (the ‘name’ graph attribute) is printed if present. The second line is optional and it contains all the attributes of the graph. This graph has a ‘name’ graph attribute, of type character, and two other graph attributes called ‘mutual’ and ‘circular’, of a complex type. A complex type is simply anything that is not numeric or character. See the documentation ofprint.igraph for details.

If you want to see the edges of the graph as well, then use the print_all function:

```
> print_all(g)
IGRAPH badcafe U--- 10 10 -- Ring graph
+ attr: name (g/c), mutual (g/x), circular (g/x)
+ edges:
 [1] 1-- 2 2-- 3 3-- 4 4-- 5 5-- 6 6-- 7 7-- 8 8-- 9 9--10 1--10
```

Creating graphs

There are many functions in igraph for creating graphs, both deterministic and stochastic; stochastic graph constructors are called ‘games’.

To create small graphs with a given structure probably the graph_from_literal function is easiest. It uses R's formula interface, its manual page contains many examples. Another option is graph, which

takes numeric vertex ids directly.graph.atlas creates graph from the Graph Atlas, make_graph can create some special graphs.

To create graphs from field data, graph_from_edgelist, graph_from_data_frame andgraph_from_adjacency_matrix are probably the best choices.

The igraph package includes some classic random graphs like the Erdos-Renyi GNP and GNM graphs (sample_gnp,sample_gnm) and some recent popular models, like preferential attachment (sample_pa) and the small-world model (sample_smallworld).

Vertex and edge IDs

Vertices and edges have numerical vertex ids in igraph. Vertex ids are always consecutive and they start with one. I.e. for a graph with *n* vertices the vertex ids are between *1* and *n*. If some operation changes the number of vertices in the graphs, e.g. a subgraph is created via induced_subgraph, then the vertices are renumbered to satisfty this criteria.

The same is true for the edges as well, edge ids are always between one and *m*, the total number of edges in the graph.

It is often desirable to follow vertices along a number of graph operations, and vertex ids don't allow this because of the renumbering. The solution is to assign attributes to the vertices. These are kept by all operations, if possible. See more about attributes in the next section.

Attributes

In igraph it is possible to assign attributes to the vertices or edges of a graph, or to the graph itself. igraph provides flexible constructs for selecting a set of vertices or edges based on their attribute values, see vertex_attr, V and E for details.

Some vertex/edge/graph attributes are treated specially. One of them is the 'name' attribute. This is used for printing the graph instead of the numerical ids, if it exists. Vertex names can also be used to specify a vector or set of vertices, in all igraph functions. E.g. degree has a `v` argument that gives the vertices for which the degree is calculated. This argument can be given as a character vector of vertex names.

Edges can also have a 'name' attribute, and this is treated specially as well. Just like for vertices, edges can also be selected based on their names, e.g. in the delete_edges and other functions.

We note here, that vertex names can also be used to select edges. The form '`from|to`', where '`from`' and '`to`' are vertex names, select a single, possibly directed, edge going from '`from`' to '`to`'. The two forms can also be mixed in the same edge selector.

Other attributes define visualization parameters, see igraph.plotting for details.

Attribute values can be set to any R object, but note that storing the graph in some file formats might result the loss of complex attribute values. All attribute values are preserved if you use save and load to store/retrieve your graphs.

Visualization

igraph provides three different ways for visualization. The first is the `plot.igraph` function. (Actually you don't need to write`plot.igraph`, `plot` is enough. This function uses regular R graphics and can be used with any R device.

The second function is `tkplot`, which uses a Tk GUI for basic interactive graph manipulation. (Tk is quite resource hungry, so don't try this for very large graphs.)

The third way requires the `rgl` package and uses OpenGL. See the `rglplot` function for the details.

Make sure you read `igraph.plotting` before you start plotting your graphs.

File formats

igraph can handle various graph file formats, usually both for reading and writing. We suggest that you use the GraphML file format for your graphs, except if the graphs are too big. For big graphs a simpler format is recommended. See `read graph`and `write graph` for details.

Further information

The igraph homepage is at http://igraph.org. See especially the documentation section. Join the igraph-help mailing list if you have questions or comments.
