The igraph library for complex network research

WHAT IS IGRAPH?

A platform for the analysis of large complex networks
igraph is a tool intended for researchers working with large datasets that can be represented in network form. It contains a simple but efficient indexed edge-list data structure for such data, along with implementations of many classical graph theory algorithms and more recent network analysis methods, from simple shortest path calculations to advanced graph clustering heuristics.

A library optimized for speed and efficiency
The core of *igraph* is implemented in 100% C and C++, so it is fast and efficient. (If not, then it is a bug, mail us and we will fix it). There is no built-in limit on the number of vertices and edges. If your graph can fit into the memory of your computer, then *igraph* can handle it. We have successfully analyzed graphs with 2.1 million vertices and 15 million directed edges with ease.

A tool for rapid algorithm prototyping
igraph has interfaces for higher level languages such as R, Python or Ruby. You can leverage the full power of igraph from these languages without having to write a single line of C code. Combining *igraph* with a higher level language provides a productive research environment where trying out new ideas is only a matter of a few lines of code.

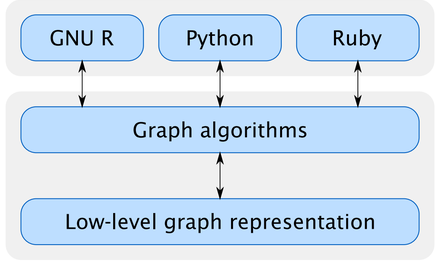
Platform independence
igraph runs on all the three major operating systems: Windows, Linux and Mac OS X.

Free (as in speech, not as in beer)
igraph is released under the GNU General Public License, so it is free for non-commercial and academic uses. You can download and modify it as you please, you can add your own algorithms and we will happily include them in the next release if it is of interest for a broader audience.

WHAT IS IGRAPH *NOT*?

igraph is not a desktop application – it is an extension to programming languages like C, R, Python or Ruby. If you want to use *igraph*, you have to write programs in one of these languages instead of clicking on buttons and menu items on a nice graphical user interface.

THE ARCHITECTURE OF IGRAPH



Higher level interfaces

Core library written in C and C++

USING IGRAPH FROM R

Installation and startup

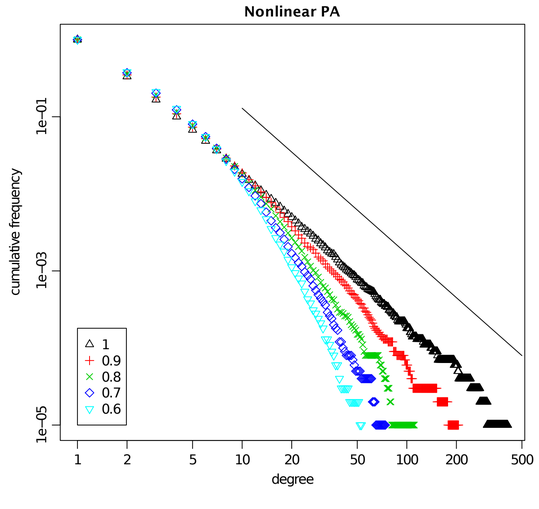
```
$ R
> install.packages("igraph")
> library(igraph)
```

Analysing basic graph properties

```
> graph <- read.graph("netscience.gml", format="gml")
> c(vcount(graph), ecount(graph))
[1] 1589 2742
> transitivity(graph)
[1] 0.6934414
> deg <- degrees(graph)
> pr <- page.rank(graph)$vector
> V(graph)$label[order(deg, decreasing=TRUE)[1:3]]
[1] "BARABASI, A" "JEONG, H"    "NEWMAN, M"
> V(graph)$label[order(pr, decreasing=TRUE)[1:3]]
[1] "NEWMAN, M"   "BARABASI, A" "JEONG, H"
```

Scale-free graphs with different PA exponents

```
> g <- barabasi.game(100000)
> degrees <- degree(g, mode="in")
> dd <- degree.distribution(g, mode="in", cumulative=T)
> fit <- power.law.fit(degrees, xmin=20)
> plot(dd, log="xy", xlab="degree", ylab="cumulative
+      frequency", col=1, pch=2, main="Nonlinear PA")
> lines(10:500, 10*(10:500)^(-coef(fit)+1))
> powers <- c(0.9, 0.8, 0.7, 0.6)
> for (p in seq(powers)) {
+   g <- barabasi.game(100000, power=powers[p])
+   dd <- degree.distribution(g, mode="in", cumulative=T)
+   points(dd, col=p+1, pch=p+2) }
> legend(1, 1e-5, c(1,powers), col=1:5, pch=1:5, ncol=1)
```



USING IGRAPH FROM PYTHON

Installation and startup

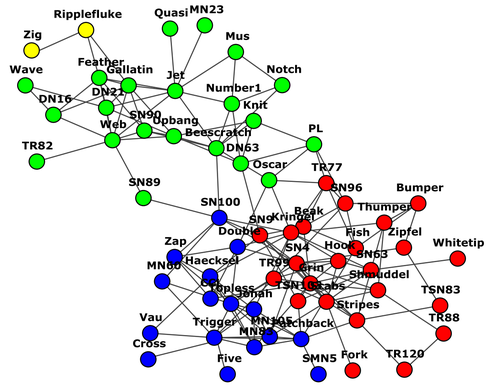
```
$ easy_install python-igraph
$ python
>>> from igraph import *
```

Analysing basic graph properties

```
>>> graph = read("netscience.gml", format="gml")
>>> print graph
Undirected graph (|V| = 1589, |E| = 2742)
>>> graph.transitivity_undirected()
0.69344141488577749
>>> pr = graph.pagerank()
>>> top = sorted(zip(graph.vs, pr), reverse=True)[:3]
>>> print [node["label"] for _, node in top]
['NEWMAN, M', 'BARABASI, A', 'JEONG, H']
```

Community detection and visualization

```
>>> graph = load("lousseau_dolphins.gml")
>>> dendrogram = graph.community_walktrap()
>>> cl = dendrogram.as_clustering()
>>> plot(cl, layout="kamada_kawai", vertex_size=20)
```



Fast prototyping: the clique percolation method (CPM)

```
def clique_percolation_method(graph, k = 4):
    # Find all cliques of size k
    cliques = map(frozenset, graph.cliques(k, k))
    # Construct clique overlap graph
    edgelist = [dict(source=cl1, target=cl2) \
                for cl1 in cliques for cl2 in cliques \
                if len(cl1.intersection(cl2)) == k-1]
    clique_graph = Graph.DictList(edges=edgelist)
    # Return the members of each connected component
    return (set().union(*members) \
            for members in clique_graph.components())
```

Palla *et al*: Uncovering the overlapping community structure of complex networks in nature and society. Nature **435**, 814-818 (2005)

ALGORITHMS IN IGRAPH

Graph generators
rings, stars, lattices, trees ● famous graphs by name ● Erdos-Renyi random graphs ● linear, nonlinear and aging preferential attachment ● Watts-Strogatz model ● forest fire model ● geometric random graphs ● many others...

Centrality properties
degree ● closeness ● betweenness ● eigenvector centrality ● PageRank ● hub and authority scores ● Burt's constraint scores ● betweenness estimation

Structural properties of graphs
density ● reciprocity ● clustering coefficient ● k-cores ● girth ● spanning trees ● articulation points ● max-flows and min-cuts ● vertex and edge connectivity ● maximal and largest cliques ● independent vertex sets

Shortest path calculations
breadth first search ● Dijkstra's algorithm ● Bellman-Ford algorithm ● Johnson's algorithm ● diameter ● *all* shortest paths between two vertices

Community detection methods
strongly/weakly connected components ● Newman-Girvan method ● greedy modularity optimization ● spinglass clustering ● walktrap clustering ● multilevel clustering (Louvain method) ● label propagation ● modularity

Layout algorithms and visualization
circle and sphere layout ● Fruchterman-Reingold in 2D and 3D ● Kamada-Kawai in 2D and 3D ● Large Graph Layout (LGL) ● DrL algorithm ● Reingold-Tilford tree layout ● plotting graphs in PNG, PDF and SVG formats

Importing from and exporting to foreign formats
adjacency matrices ● edge lists (numeric and symbolic) ● NCOL ● LGL ● Pajek ● GraphViz (export only) ● GML ● GraphML ● GraphDB

HOW TO GET HELP OR REPORT BUGS?

Documentation	http://igraph.sf.net/docs.html
Wiki	http://igraph.wikidot.com
Mailing lists	http://igraph.sf.net/support.html
Bug tracker	http://igraph.sf.net/bugs.html

AUTHORS & ACKNOWLEDGMENTS

Gábor Csárdi (Gabor.Csardi@unil.ch)
Department of Medical Genetics, University of Lausanne

Tamás Nepusz (tamas@cs.rhul.ac.uk)
Centre for Systems and Synthetic Biology, Department of Computer Science, Royal Holloway, University of London

Thanks to: the authors of third-party algorithms (BLISS, DrL, spinglass & walktrap clustering, Viger-Latapy graph generator), Jeroen Bruggeman, Tom Gregorovic, Bernie Hogan, Peter McMahan and all the people who reported bugs or contributed code to igraph.

DOWNLOAD IGRAPH CORE LIBRARY
http://igraph.sourceforge.net

DOWNLOAD IGRAPH FOR R
http://cran.r-project.org/web/packages/igraph

DOWNLOAD IGRAPH FOR PYTHON
http://pypi.python.org/pypi/python-igraph

DOWNLOAD IGRAPH FOR RUBY
http://igraph.rubyforge.org