# The Igraph Software Package for Complex Network Research

**Article** · November 2005



**2 authors**, including:

Gabor Csardi
Harvard University
**40** PUBLICATIONS **13,026** CITATIONS

SEE PROFILE

# The igraph software package for complex network research

**Gábor Csárdi**
Center for Complex Systems Studies, Kalamazoo College, Kalamazoo, MI, USA
and
Department of Biophysics, KFKI Research Institute for Particle and Nuclear Physics of the Hungarian Academy of Sciences, Budapest, Hungary
csardi@kzoo.edu

**Tamás Nepusz**
Department of Biophysics, KFKI Research Institute for Particle and Nuclear Physics of the Hungarian Academy of Sciences, Budapest, Hungary
and
Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary
ntamas@rmki.kfki.hu

The igraph software package provides handy tools for researchers in network science. It is an open source portable library capable of handling huge graphs with millions of vertices and edges and it is also suitable to grid computing. It contains routines for creating, manipulating and visualizing networks, calculating various structural properties, importing from and exporting to various file formats and many more. Via its interfaces to high-level languages like GNU R and Python it supports rapid development and fast prototyping.

## 1.1 Introduction

This paper does not present results of scientific research, but introduces a software package which gives handy tools into the hands of researchers doing network science. The authors strongly believe that the tools scientists use are important because they can increase productivity by several factors and thereby enhance scientific progress.

### 1.1.1 Why another network analysis package?

The igraph library was developed because of the lack of network analysis software which (1) can handle large graphs efficiently, (2) can be embedded into a higher level program or programming language (like Python, Perl or GNU R) and (3) can be used both interactively and non-interactively.

The capability of handling large graphs was important because the authors were confronted with graphs with millions of vertices and edges.

Embedding igraph into Python or GNU R creates a very productive research environment, well suited for rapid development. All the expressing power of GNU R (or other higher level languange) is readily available in a convenient integrated environment for generating, manipulating and measuring graphs, and evaluating these measurements.

Interactive means of software usage is nowadays considered as superior to non-interactive interfaces, which is very true for most cases. Dealing with large graphs can be different though – if it takes three months to calculate the diameter of a graph, nobody wants that to be interactive.

## 1.2 Features

In the addition to the three goal features in the previous section, others showed up as a side-effect. Let us discuss these features here.

**Open source.** Igraph is open source, it is free for non-commercial or commercial use and distributed according to the GNU General Public License. Being open source means that in addition to the binary format of the program, the user can always get the source code format enabling additions and corrections. This is a very important feature for the users. With open source software, you can add new functionality and correct deficiencies or hire somebody to do this for you. With closed source software this is impossible.

**Efficient implementation.** Igraph uses space and time efficient data structures and implements the current state-of-the-art algorithms. All igraph functions are carefully profiled to create the fastest implementation possible.

**Portability.** The library is written in ANSI C, it is thus portable to most platforms. It is tested on different Linux flavors, Mac OS X, MS Windows and Sun OS. The R and Python interfaces are also portable to many architectures.

**Layered architecture.** The igraph library has a layered architecture, the three layers are connected through well defined interfaces. Each layer can be replaced with an alternate implementation without changing the other components. See the details in §1.4.

**Open, embeddable system.** The core igraph library is an open system, it can be embedded into higher level languages or programs. The current distribution contains interfaces to two high level languages: GNU R and Python.

**High level operations.** The higher level interfaces provide abstract operations and data types. These support rapid program development, see §1.3.2 for an example.

**Documentation.** The C library is very well documented, the documentation is available in various formats supporting both online browsing and printing. For each function its time requirements are documented.

**Drawbacks.** The library lacks functionality in some areas compared to other network analysis packages. One such area is graph visualization, another one is various social network analysis methods like block-modeling, $p^*$ methods, etc. Note that this piece of software is heavily under development, so expect much more functionality in the near future. Igraph also does not have a graphical user interface, but a Python-based GUI is under development and will be available for download soon and the R interface also provides a facility for visual manipulation of small graphs.

## 1.3 Example applications

### 1.3.1 Grid computing

In this section we give an example for using the igraph library for large scale computation. The task presented here is to calculate the diameter of the US patent citation network. In this network the nodes are US patents granted between 1963 and 2000 and two patents are connected if one cites the other. The largest component of the network contains more than 3 million nodes and 15 million edges. The (undirected) diameter of a network is the largest undirected shortest path connecting two vertices.

For calculating the diameter of a graph you need to calculate the length of the shortest path between all possible pairs of nodes, so this is computationally very expensive. We used the following approach with igraph.

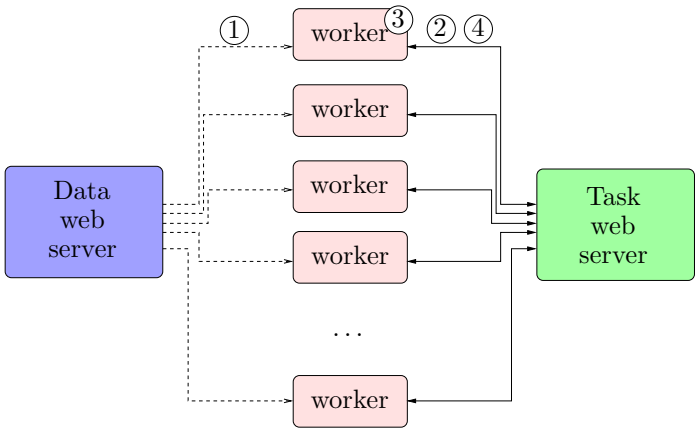


**Figure 1.1**: The architecture of the system used for calculating the diameter of a large graph. A worker node (1) downloads the network data from the data web-server, then (2) it requests a source vertex id from the task web-server, (3) calculates the shortest paths from that source vertex and (4) stores the result on the task web-server. Then a new source vertex id is requested, etc.

First we wrote a simple program in C which downloads the data set from a web server and then starts calculating the shortest paths from a given source node to all other nodes in the network by using Dijkstra's algorithm [5] implemented in the igraph library. We will call this program the *worker*.

The worker downloads the id of the source node from a second web server. This web server simply gives a different source node id every time one is requested by the workers. As soon as the worker has finished with the calculation of the shortest paths to all nodes it stores the result on a third web server and asks the second web-server for a new source node id, etc. The architecture of the system can be seen in Fig. 1.1.

This system is very robust in the sense that there is no single point of failure. The workers can be run in any grid-based environment from which they can access the WWW. They can be run on different platforms as well.

### 1.3.2 Fast prototyping, rapid development

**Newman's community finding algorithm** The second example we present is very different from the first. Here we will use the GNU R interface to the igraph library to implement and apply Newman's spectral community finding algorithm [10]. First we load the igraph package into R and download the Zachary Karate-club network data [17] from the web.

```
1 library(igraph)
2 g <- read.graph("http://geza.kzoo.edu/~csardi/karate.net", format="pajek")
```

Now we implement the community finding algorithm.

```
community.newman <- function(g) {
  deg <- degree(g)
  ec <- ecount(g)
  B <- get.adjacency(g) - outer(deg, deg, function(x,y) x*y/2/ec)
  diag(B) <- 0
  eigen(B)$vectors[,1]
}
```

This algorithm creates a modularity matrix which is the difference of the adjacency matrix of the graph and the null-model matrix. The latter contains the probabilities that two nodes are connected in a random graph if the degrees of the nodes are given. Then the network is divided into two communities based on the eigenvector associated with the largest positive eigenvalue of the modularity matrix: all vertices having the same sign in this eigenvector belong to the same community.

Now we are ready to apply this algorithm to the Karate-club data and set the color of the vertices based on their communities.

```
mem <- community.newman(g)
V(g)$color <- ifelse(mem < 0, "grey", "green")
```

We also set the size of the vertices based on the first eigenvector, the farther this value is from zero the more the given vertex is in the *core* of the community. We also set the color of the edges across the two communities to red.

```
scale <- function(v, a, b) {
  v <- v-min(v) ; v <- v/max(v) ; v <- v * (b-a) ; v+a
}
V(g)$size <- scale(abs(mem), 15, 25)
E(g)$color <- "grey"
E(g)[ V(g)[color=="grey"] %--% V(g)[color=="green"] ]$color <- "red"
plot(g, layout=layout.kamada.kawai, vertex.color="a:color",
       vertex.size="a:size", edge.color="a:color")
```

See the resulting plot in Fig. 1.2.

**PageRank algorithm in 19 lines** Using the Python interface of igraph, one can easily create a prototype of the original PageRank algorithm in only 19 lines of code (not counting empty lines):

```
from igraph import *
from copy import copy

def pagerank(g, damping=0.85, epsilon=0.001, iters=100):
    pageranks = [1-damping] * g.vcount()
    outlinks = g.degree(type=OUT)
    mindiff = epsilon
    newprs = [0] * g.vcount()

    while mindiff >= epsilon and iters > 0:
```

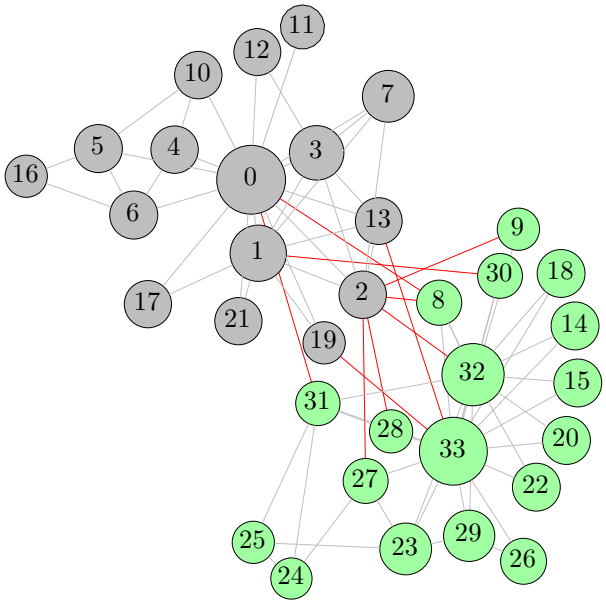


**Figure 1.2**: The two communities identified correctly in the Zachary karate-club network. The size of the vertices is proportional to the absolute value of their coordinate in the first eigenvector and expresses how strongly they belong to a community. All edges across the two communities are painted red.

```
11          iters = iters - 1
12          for n in range(g.vcount()):
13              neis = g.neighbors(n, IN)
14              pr = 0.0
15              if len(neis) > 0:
16                  for n2 in neis: pr = pr + pageranks[n2] / outlinks[n2]
17                  pr = pr*damping
18              newprs[n] = pr+1-damping
19
20          mindiff = min([abs(newprs[n]-pageranks[n]) for n in range(g.vcount())])
21          pageranks = copy(newprs)
22
23      return pageranks
```

## 1.4 The igraph architecture

The igraph system has a layered architecture consisting three layers. The lowest layer contains the very basic operations only, and is implemented in C. It is only this layer which can manipulate the internal igraph data structures directly. This means that this layer can be easily replaced with and alternate graph representation if needed.

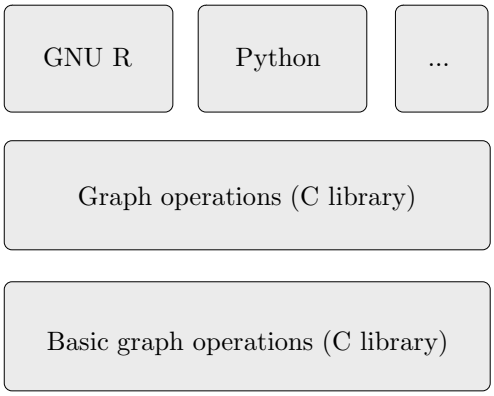


**Figure 1.3**: The architecture of the igraph system. See the text for a description.

The second layer contains almost all network analysis functions, this is also implemented in C.

The third layer contains the higher level interfaces, so far interfaces to GNU R and Python are implemented.

## 1.5 Current functionality

Please note that new functionality is added to the library every week, so check the igraph homepage at `http://cneurocvs.rmki.kfki.hu/igraph` if you cannot see here the algorithms or measures you're looking for.

**Graph generation** Igraph can generate various regular and random graphs: • regular structures: star, ring and full graphs, circular and non-circular lattices with any number of dimensions, regular trees • graphs based on Barabási's preferential attachment model [1], also with nonlinear attachment exponent and various variations • Random (Erdős-Rényi) graphs, both $G(n, p)$ and $G(n, m)$ types [6], directed and undirected ones • graphs having a given degree sequence, directed or undirected ones [11] • growing random graphs, also for modeling citation networks [3] • growing random graphs where the connection probability depends on some vertex properties • graphs from the Graph Atlas [13] • all non-isomorphic graphs of a given size.

**Centrality measures** The following centrality measures [7] can be calculated: • degree • closeness • vertex and edge betweenness • eigenvector centrality • page rank [12].

**Path length based properties** One or all shortest paths between vertices can be calculated, and also based on this the diameter and the average path length of the graph.

**Graph components** Weakly and strongly connected components can be calculated, and also the minimum spanning forest of a graph.

**Graph motifs** Graph motifs of three or four components can be calculated, both undirected and directed motifs [16].

**Random rewiring** Existing graphs can be rewired randomly while preserving their degree distribution, allowing the user to generate an arbitrary set of graphs with the same degree distribution.

**Vertex and edge sets** Igraph provides a simple way to manipulate subsets of vertices and/or edges of a graph, see §1.3.2 for an example.

**Vertex and edge attributes** Numeric or non-numeric attributes can be assigned to the vertices and edges of a graph, and queried and set by using a simple notation, see §1.3.2.

**File formats** Igraph can read and write simple edge list files and also Pajek [4] and GraphML [2] files.

**Graph layouts** The following layout generators are part of igraph: • simple circle and sphere layouts, random layouts • Fruchterman-Reingold layout, 2D and 3D [8] • Kamada-Kawai layout, 2D and 3D [9] • spring embedder layout • LGL layout generator for large graphs [15] • Grid-based Fruchterman-Reingold layout for large graphs • Reingold-Tilford layout [14] for trees.

# Bibliography

[1] Barabási, Albert-László, and Réka Albert, "Emergence of scaling in random networks", *Science* **286**, 5439 (1999), 509–512.

[2] Brandes, U., M. Eiglsperger, I. Herman, M. Himsolt, and M.S. Marshall, "Graphml progress report: Structural layer proposal", *Proc. 9th Intl. Symp. Graph Drawing (GD '01)*, 501–512.

[3] Callaway, D. S., J. E. Hopcroft, M. E. J. Newman J. M. Kleinberg, and S. H. Strogatz, "Are randomly grown graphs really random?", *Phys. Rev. E* **64** (2001), 041902.

[4] de Nooy, W., A. Mrvar, and V. Batagelj, *Exploratory Social Network Analysis with Pajek*, Cambridge University Press (2005).

[5] Dijkstra, E. W., "A note on two problems in connexion with graphs", *Numerische Mathematik* **1** (1959), 269–271.

[6] Erdős, P., and A. Rényi, "On random graphs", *Publicationes Mathematicae* **6** (195), 290–297.

[7] Freeman, L.C., "Centrality in social networks I: Conceptual clarification", *Social Networks* **1** (1979), 215–239.

[8] Fruchterman, T. M. J., and E. M. Reingold, "Graph drawing by force-directed placement", *Software – Practice and Experience* **21** (1991), 1129–1164.

[9] Kamada, T., and S. Kawai, "An algorithm for drawing general undirected graphs", *Information Processing Letters* **31**, 1 (1989), 7–15.

[10] Newman, M. E. J., "Modularity and community structure in networks", *Proc. Natl. Acad. Sci. USA* **in press** (2006).

[11] Newman, M. E. J., S. H. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distributions and their applications", *Phys. Rev. E* **64** (2001), 026118.

[12] Page, Lawrence, Sergey Brin, Rajeev Motwani, and Terry Winograd, "The pagerank citation ranking: Bringing order to the web", *Tech. Rep. no.*, Stanford Digital Library Technologies Project, (1998).

[13] Read, Ronald C., and Robin J. Wilson, *An Atlas of Graphs*, Oxford University Press (1998).

[14] Reingold, E., and J. Tilford, "Tidier drawing of trees", *IEEE Transactions on Software Engineering* **7** (1981), 223–228.

[15] T, Adai A, Date S V, Wieland S, and Marcotte E M, "LGL: creating a map of protein function with an algorithm for visualizing very large biological networks", *J Mol Biol* **340** (2004), 179–90.

[16] Wernicke, S., and F. Rasche, "FANMOD: a tool for fast network motif detection", *Bioinformatics* **22**, 9 (2006), 1152–1153.

[17] Zachary, W. W., "An information flow model for conflict and fission in small groups", *Journal of Anthropological Research* **33** (1977), 452–473.