A Walk on Python-igraph

Carlos G. Figuerola - ECyT Institute
University of Salamanca
- 17/04/2015

i-graph

- i-graph is a library
- written in C++ (fast and memory efficient)
- a tool for programmers
- it works from programs in R, Python (and C++, btw)
  here, we are going to work with *Python* + *i-graph*

Python

- a programming language intended for scripting
- interpreted language (semi-compiled/bytecodes, actually)
- it is free software (runing on Linux, Win, Mac machines)
- it has a lot of modules for no matter about wich task
- it is very intuitive but ... **it has powerfull data structures, but complex**

Installing *i-graph*

- Web Home at http://igraph.org/
- in general you will need Python and a C/C++ compiler. Use of *pip* or *easy_install* is more convenient
- Linux: it is in main repositories, better install from there
- Win: there are (unnofficial) installers in: http://www.lfd.uci.edu/~gohlke/pythonlibs/#python-igraph (see instructions on top of this page)
- Mac: see https://pypi.python.org/pypi/python-igraph

i-graph: pros & cons

- it can deal with big amounts of data
- it has a lot of measures, coefficients, procedures and functions
- it is flexible and powerfull, can be used with *ad-hoc* programs/scripts or in interactive way (commands from a text terminal)
- no (native) graphic interface, no mouse nor windows
- not very good branded graphics (although yo can import/export easily from/to another software)

First steps

- start a python terminal
- import the library and create a new (undirected) graph

```
import igraph
X=igraph.Graph()
```

in *python-igraph* a graph is an object with methods and properties

Firts steps

- we get a directed graph with

```
Y=igraph.Graph(directed=True)
```

- and we get an idea about our graphs with:

```
print X
```

```
>>> print Y
IGRAPH D--- 0 0 --
>>>
```

Populating the graph: vertices

- in python-igraph every node (vertices) has an unique ID, which is an automatic number
- vertices can be added with **add_vertices(*n*)**

```
>>>
>>> X.add_vertices(5)
>>> print X
IGRAPH U--- 5 0 --
>>>
```

- vertices are renumbered starting from 0
- in *python-igraph*, vertices are encapsulated as an iterable list: Graph.**vs**

Vertices's attributes

- vertices can have arbitrary attributes
- from python, attributes of a vertex ca be seen as a dictionary. For example:

```
X.vs[0]['university']='uni-dortmund.de'
X.vs[0]['size']=237445
```

  this sets the attribute *university* of vertex *0* to the value of *uni-dortmund.de*, as well as the attribute *size* to the numeric value of *237445*

- attributes are created at the moment of asignment of their values
- different vertices could have different attributes (although this can be unpractical)

More on vertices's attributes

- values of an attribute can be assigned for all vertices in a single sentence

```
X.vs['university']=['uni-dortmund.de', 'usal.es', 'unipd.it']
X.vs['size']=[237445, 27000, 56888]
```

More ...

- this allows us easily read data from a CSV file and populate the graph with nodes and attributes

```
universities, students = [], []
F=open('file.csv','r')
for line in F.readlines()
    u,s = line.strip('\n').split(',')
    universities.append(u), students.append(s)
F.close()
X.vs['university']=universities
X.vs['size']=students
```

Edges

- edges are simmilar to vertices
- we can add edges by supplying a list of them
- an edge is a tuple of two members: the nodes linked by such edge
- in directeds graphs, order of nodes is important (origine and target)

```
X.add_vertices([(0,2),(0,3),(1,2)])
```

- if a vertice does'nt exists, we'll get an Error!

Edges ...

- we can have self-edges

```
X.add_vertices([(2,2)])
```

- every edge has an ID (a sequential number)
- we can delete an edge with:

```
X.delete_edge(2)
```

Import/export graphs from/to disk

- *igraph* deals with major graph file formats
- it has generic *load* and *save* methods, comprising several formats.
- for example, this imports a graph in *graphml* format (*igraph* guesses the format by the extension of filename!)

```
X=igraph.load('red-universidades.graphml')
```

- and this exports an igraph graph in Pajek format:

```
X.save('red-universidades.net')
```

Measures, coefficients, transformations ...

- once we have a graph, either built inside igraph, either imported from another software, we can:
  - compute measures and coefficients about several aspects of such graph
  - perform actions on the graph (transform it, extract parts of it, etc.)
- igraph has hundreds of methods to do hundreds of this kind of operations (see Reference of library!)
- source code is available; since python is an interpreted language, one can modify or even create new methods, if necessary

Measures of whole graph

- *igraph* computes used measures about all the whole graph, for example:

```
X.density()
X.diameter(directed=True)
```

- some of measures needs aditional arguments
- result of this kind of measures is a single value

Measures of nodes

- igraph computes measures about individual nodes (although some of them can require data from the others nodes)

```
d=X.indegree()
b=X.betweenness()
```

- results are a list (in Pyhton sense of lists), in which the value of the element [n] from the list is the value of the [n] node in the graph
- we can add the computed measure as an attribute to he nodes

```
X.vs['in']=d
X.vs['between']=b
```

Measures of nodes ...

- we can sort nodes by the value on an attribute (for example, by the betweenness we have computed and after added as a new attribute)

```
m=sorted(X.vs, key=lambda z:z['between'], reverse=True)
for e in m[:10]:
    print e['name']
```

- this shows the first node names with higher betweenness

Plotting graphs

- making graphics from graphs is not the best of *igraph* library
- we can do graphics in two simple steps:
  - choosing a *layout* (algorithm), it have several *layouts* availables
  - applying such layout to our graph

```
z=X.layout('fr')
igraph.plot(X, layout=z)
```

Plotting graphs...

- *igraph* provide us with tools to adjust colour, shape of nodes and archs, size of nodes and wide of arch's lines, etc.
- as this adjusting task must be done by scripting, it can be a good choice if you have to apply usually the same kind of graphic
- if not, better export your graph to an interactive plotting software (as *gephi*, *netdraw*, etc...)

Big Data

- *igraph* is really good working with big amounts of data
- we performed several tasks with a big graph, to test this
- **Topic**:Individual personalities in the Spanish Transition period (1977-1987)
- **Source of data**: all news from the Digital Archives of the newspaper El País (all news from all sections, included sports, tv, and so on ...)
  - this means **425,335** news full text

Big Data ...

- by means of a Named Entities Recognition system, we obtained all persons appearing in every news. This systems is OpenCalais service.
  - this means **286,580** different persons
- with this data, we built our graph
  - every person is a node
  - if two persons appears in the same news, we can assume they are related in some way; so, this an edge in our graph, linking both persons
  - weight of edge is proportional to the number of news the linked persons appear together
  - this means **1,344,396** edges

Big Data ...

- we have data in CSV format, which is also known as *ncol* in the graph world

```
ABELARDO_COLLAZO_ARAUJO JUAN_MARTIN_LUNA    6
ABELARDO_COLLAZO    FERNANDO_HIERRO_CHOMON  7
ABEL_CABALLERO  ALFONSO_GUERRA  30
ABEL_CABALLERO  CARLOS_SOLCHAGA 14
ABEL_CABALLERO  ERNEST_LLUCH    8
ABEL_CABALLERO  FELIPE_GONZALEZ 33
```

- loading 0.42M vertices and 1.34M edges takes only 2 seconds in my notebook (intel *i3*, 4G RAM)

Big Data ...

- computing a simple used node measure as the degree is instant:

```
d=X.degree()
X.vs['degree']=d
```

- sorting people by their degree and showing the top ten, is also instant

```
m=sorted(X.vs, key=lambda x:x['degree'], reverse=True)
for e in m[:10]:
        print e['name']
```

Big Data ...

```
FELIPE_GONZALEZ
ADOLFO_SUAREZ
JUAN_CARLOS
ALFONSO_GUERRA
RONALD_REAGAN
MANUEL_FRAGA
JAVIER_SOLANA
JUAN_PABLO_II
SANTIAGO_CARRILLO
FRANCISCO_FERNANDEZ_ORDOÑEZ
```

Big Data ...

- betweenness is some slower: about 5 mins.
- and here we have the top ten list:

```
FELIPE_GONZALEZ
ADOLFO_SUAREZ
JUAN_CARLOS
ALFONSO_GUERRA
RONALD_REAGAN
JUAN_PABLO_II
JAVIER_SOLANA
JOSE_BARRIONUEVO
JORDI_PUJOL
SANTIAGO_CARRILLO
```

(remark this is not exactly the same list as with degree, but almost)

Communities Discovery

- a community is a set of vertices which links strongly between them, and weakly with the other nodes outside the community
- detecting communities is a way of automatic organization of information
- it can help us to find new knowledge
- *igraph* has several methods to perform community detection

Communities Discovery ...

- for example, the *InfoMap* based algorithm:

```
C=X.community_infomap(edge_weights='weight')
```

- as a result we obtain a list of communities
  - each element of this list is a community
  - each community is also a list: the list of nodes that belongs to such community
  - actually, ther resul of Community Detection is an *igraph* object, but we access from python as if it were a list of lists

How many communities?

- as the result of InfoMap is a list we know the number of communities with:

```
len(C)
4447
```

this is a lot of communities! but ... take a look to the size of every community

How many communities? ...

```
for n in range(0,len(C)):
    print 'Community nº', n, 'size:', len(C[n])
```

- only 87 communities have more than 50 members
- only 44 have more than 100 members
- only 5 communities have more than 500 people.
- remember, we had more than 0.25M personalities

Who is in a community?

- community 2 (660 members), their few first members are:

```
for z in C[2][:10]:
      print X.vs[z]['name']
```

```
ANTONIO_MACHADO
CAMILO_JOSE_CELA
ABEL_POSSE
ANGEL_GONZALEZ
ARTURO_AZUELA
AUGUSTO_ROA_BASTOS
CARLOS_BARRAL
CARLOS_GERMAN_BELLI
FRANCISCO_UMBRAL
JORGE_EDWARDS
```

- they all are writers or related

Who is in a community?

- in community 1 (761 members), we get:

```
JIMMY_CARTER
LEONIDAS_BREJNEV
ANDREI_GROMIKO
DONALD_RUMSFELD
RONALD_REAGAN
YURI_ANDROPOV
CYRUS_VANCE
HODDING_CARTER
JAVIER_PEREZ_DE_CUELLAR
ABEL_AGAMBEGIAN
```

- all of them people about international affairs

Who is in a community?

- for community 3 (691 members) we have people as:

```
ABEL
MANOLO
ABEL_TORRENTE
ALFONSO_CABEZA
ANTONIO_CALDERON
FRANCISCO_BERMEJO
MANUEL_FERNANDEZ_TRIGO
ABLANEDO
ELOY
GALLEGO
```

- they seem weird names, but older readers can recognize they are football players, coaches and so on

Who is in a community?

- community 0 (2,311 members) has politicians, as we can expect

```
AGUSTIN_RODRIGUEZ_SAHAGUN
ALBERTO_OLIART
ENRIQUE_MUGICA
FELIPE_GONZALEZ
JUAN_CARLOS
RODRIGUEZ_SAHAGUN
SANTIAGO_CARRILLO
FRANCISCO_FERNANDEZ_ORDOÑEZ
LUIS_YAÑEZ
ADOLFO_SUAREZ
```

Communities as subgraphs

- we can transform a single community into a graph

```
Z=C.subgraph(2)
```

  (remember, C was the object returned by InfoMap)

- we have a new graph, Z, which is made of nodes in community 2 and edges between them

Communities as subgraphs ...

- we apply any of available methods:

```
b=Z.betweennes()
Z.vs['betweennes']=b
for a in sorted(Z.vs, key=lambda x:x['betweenness'],reverse=True)[:10]:
     print a['name']
```

```
CAMILO_JOSE_CELA
RAFAEL_ALBERTI
DAMASO_ALONSO
ANTONIO_MACHADO
ANTONIO_TOVAR
ANTONIO_GALA
CARLOS_BARRAL
FERNANDO_SAVATER
JORGE_GUILLEN
JUAN_RAMON_JIMENEZ
```

Communities in a subgraph(subcommunities)

- we can apply community detection methods to our subgraph

```
SC=Z.community_infomap(edge_weights='weight')
```

(remember, Z is a subgraph made from community 2 of the whole big graph, as seen before, this is a community of writers, editors, etc.)

- SC has 31 communities, but only 11 ave more than 10 members, and only 4 more than 50

Subcommunities of writers, editors, ...

- following, we show the first members of the widest subcommunities of writers, editors, etc.
- we can see subcommunities of poets, filologists, some specific kind of novelists, of latinoamerican writers ...

Subcommunities of writers, editors, ...

- subcommunity 0 has people like:

```
ANTONIO_MACHADO
ABRAHAM_SUTZKEVER
RAFAEL_ALBERTI
JOSE_MANUEL_CABALLERO_BONALD
DAMASO_ALONSO
FELIX_GRANDE
GERARDO_DIEGO
JORGE_GUILLEN
LUIS_ROSALES
MARIA_ZAMBRANO
```

Subcommunities of writers, editors, ...

- community 1 has people like:

```
PEDRO_LAIN_ENTRALGO
ALONSO_ZAMORA_VICENTE
ANTONIO_TOVAR
FERNANDO_LAZARO_CARRETER
FRANCISCO_LOPEZ_ESTRADA
JOSE_LUIS_ARANGUREN
MANUEL_SECO
DIONISIO_RIDRUEJO
FERNANDO_LAZARO
MANUEL_DIEZ_ALEGRIA
```

Subcommunities of writers, editors, ...

- community 2 has people like:

```
FRANCISCO_UMBRAL
GONZALO_TORRENTE_BALLESTER
ANTONIO_GALA
CARMEN_MARTIN_GAITE
JOSE_HIERRO
MANUEL_VAZQUEZ_MONTALBAN
MIGUEL_DELIBES
ROSA_MONTERO
FERNANDO_VIZCAINO_CASAS
GARCIA_PAVON
```

Subcommunities of writers, editors, ...

```
ANGEL_GONZALEZ
JOSE_LUIS_SAMPEDRO
JUAN_BENET
JAVIER_MARIAS
RAFAEL_CONTE
JAIME_SALINAS
JUAN_GARCIA_HORTELANO
ANTONIO_GARRIGES
JOSE_MIGUEL_ULLAN
JUAN_CUETO
JUAN_GOYTISOLO
```

Subcommunities of writers, editors, ...

```
AUGUSTO_ROA_BASTOS
JORGE_LUIS_BORGES
LUIS_GOYTISOLO
ADOLFO_BIOY_CASARES
ARTURO_USLAR_PIETRI
JUAN_CARLOS_ONETTI
MIGUEL_R._ORTEGA
OCTAVIO_PAZ
ALEJO_CARPENTIER
JOAQUIN_DE_ENTRAMBASAGUAS
```

Concluding ...

- we have seen major features of *python-igraph*
- classes, methods and data structures are accessed as Python artifacts
- we can easily import/export from/to another software
- is is really good for big graphs!

Thank You!

Contact information:

e-mail figue@usal.es

www reina.usal.es