graphtools Documentation

Release 1.0

Deepak Bal and Gautam Sisodia

September 27, 2014

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 Hierarchy on Twitter

Who are the power players on Twitter (or in a Twitter community)? A good indicator of power is the number of followers a user has. But an identification of the powerful users based solely on the number of followers would miss the users with few but powerful followers. We describe a measure of a user's power which incorporates the power of the user's followers.

Consider the directed graph with vertices the users in a Twitter community and an arrow from user A to user B if and only if A follows B. In this graph, the powerful users will have many in neighbors (followers), and many of their out neighbors (friends) will be powerful themselves. On the other hand, the average user will have many out nieghbors but few in neighbors. Thus the Twitter graph may exhibit high hierarchy as defined below.

Let $G = (V, E)$ be a directed graph. Write $n = |V|, m = |E|$. All of the following definitions are (directly or adapted) from [FHSN].

Definition A **ranking** of G is a map $V \to \mathbf{N}$ where $\mathbf{N}$ is the natural numbers, and the value of a vertex under the map is called the **rank** of the vertex. Denote the set of rankings of G by $R(G)$.

Definition The **agony** of a ranking $r \in R(G)$ is

$$A(r) := \sum_{(u,v)\in E} \max\{r(u) - r(v) + 1, 0\}.$$

The **hierarchy** of r is

$$h(r) := 1 - (1/m)A(r).$$

The agony of a ranking counts the arrows in the graph which don't go up in rank (counted with one more than the difference in ranks). The naive ranking by the zero map has agony m, so any ranking at least as good as the naive one has hierarchy between 0 and 1.

If a ranking of the graph of a Twitter community has high hierarchy, then the graph exhibits hierarchal structure and rank is a good indicator of power. In the next section, we describe an algorithm to find a ranking of a graph with high hierarchy. In the next chapter, we document an implementation of that algorithm.

1.2 The descent algorithm

Let $G = (V, E)$ be a directed graph and let r be a ranking of G. Pick a vertex $v \in V$. Will updating r by increasing or decreasing the rank of v by 1 decrease the agony of r? The change in agony if the rank of v is increased by 1 is

$$i(v) := \left|\{w \in V \mid v \to w \in E \ \ \&\ \ r(w) \leq r(v) + 1\}\right| - \left|\{w \in V \mid w \to v \in E \,\&\, r(w) \geq r(v)\}\right|$$

while the change in agony if the rank of v is decreased by 1 is

$$i(v) := \big|\{w \in V \mid w \to v \in E \ \&\ r(w) \geq r(v) - 1\}\big| - \big|\{w \in V \mid v \to w \in E \ \ \&\ \ r(w) \leq r(v)\}\big|.$$

If $i(v) \leq -1$, then increasing the rank of the vertex by 1 will decrease the agony of the ranking, as will decreasing the rank by 1 if $d(v) \leq -1$

Our algorithm, which we call the descent algorithm, starts with the naive ranking (all ranks are 0) and iteratively picks a vertex at random and changes its rank by 1 if doing so will decrease the agony of the ranking.

CHAPTER

TWO

AN IMPLEMENTATION OF THE ALGORITHM

2.1 graphtools Package

This package includes graphtools.gengraph.GenGraph, a superclass implementing the descent algorithm, and subclasses of GenGraph which accomodate various graph data structures. Namely

- graphtools.dbgraph.DBGraph reads graph data stored in a sqlalchemy-supported database,
- graphtools.sagegraph.SageGraph wraps a sage.graphs.digraph.DiGraph object, and
- graphtools.listgraph.ListGraph reads graph data stored as a list of arrows.

Multiple types for the vertices are supported. The type *vertex* refers to the instance-specific type of the vertices.

2.2 gengraph Module

class graphtools.gengraph.**GenGraph**

Bases: object

A graph superclass implementing the descent algorithm. Subclasses should overwrite

- get_num_arrows(),
- get_vert_list(),
- get_rank(),
- set_rank() and
- count_neighbors()

The method descent() runs the algorithm, updating the ranks of the vertices.

count_neighbors(*vert*, *out=True*, *cond=False*, *less=True*, *cutoff=0*)

Count the number of neighbors of a vertex.

Parameters

- **vert** (*vertex*) – the vertex to count the neighbors of
- **out** (*bool*) – if True, count out neighbors, else in
- **cond** (*bool*) – if True, count the neighbors satisfying a condition on rank

- **less** (*bool*) – if True, count the neighbors with rank less than or equal to *cutoff*, else more
- **cutoff** (*int*) – the cutoff rank for the conditional

Returns the number of (in or out) neighbors of *vert* (perhaps satisfying a condition on the rank)

Return type int

descend(*vert*, *debug=False*)
Run one iteration of the descent algorithm.

Parameters

- **vert** (*vertex*) – the vertex whose rank may change
- **debug** (*bool*) – if True, verbose output

descent(*num=1*, *debug=False*)
Run `descend()` a number of times on random vertices.

Parameters

- **num** (*int*) – the number of times to run `descend()`
- **debug** (*bool*) – if True, verbose output

get_num_arrows()
Return the number of arrows.

Returns the number of arrows in the graph

Return type int

get_rank(*vert*)
Return the rank of vertex vert.

Parameters **vert** (*vertex*) – the vertex to get the rank of

Returns the rank of *vert*

Return type int

get_vert_list()
Return a list of the vertices of the graph.

Returns a list of vertices of the graph

Return type list

hierarchy_list = None
The list of hierarchy scores, updatedwith each run of descend.

reset_ranks()
Set all ranks to zero.

set_rank(*vert*, *newrank*)
Set the rank of vertex *vert* to int *newrank*.

Parameters

- **vert** (*vertex*) – the vertex to set the rank of
- **newrank** (*int*) – the new rank of *vert*

2.3 dbgraph Module

class graphtools.dbgraph.**DBGraph**(*users*, *arrows*, *conn*, *group=None*)

Bases: graphtools.gengraph.GenGraph

A subclass of GenGraph for graphs stored in databases.

The vertices are stored in a table called **users** with columns

- *user_id* (any type) and
- *rank* (int)

(the name comes from the original motivation which was Twitter user subgraphs). The table may also have a column *group* (any type) specifying the particular graph that the user belongs to if the database contains multiple graphs. If the table has no *group* column, *user_id* should be a unique identifier; if there is a *group* column, *user_id* and *group* together should be unique.

The arrows are stored in a table called **arrows** with columns

- *follow_id* and
- *lead_id*

both refering to **users**.*user_id*. If **users** has a *group* column then **arrows** should have a corresponding *group* column.

Parameters

- **users** (*sqlalchemy.schema.Table*) – the table of vertices, described above
- **arrows** (*sqlalchemy.schema.Table*) – the table of arrows, described above
- **conn** (*sqlalchemy.engine.base.Connection*) – a connection to the database
- **group** (*any*) – an optional identifier, described above

The initializer sets all entries in **user**.*rank* to 0.

For example,

```
>>> from graphtools.dbgraph import make_db, DBGraph
>>> users, arrows, conn = make_db([[1, 2], [2, 3]])
>>> graph = DBGraph(users=users, arrows=arrows, conn=conn)
>>> print graph.get_num_arrows()
2
>>> set(graph.get_vert_list()) == set([1, 2, 3])
True
>>> print graph.get_rank(1)
0
>>> graph.set_rank(3,2)
>>> graph.get_rank(3)
2
>>> graph.reset_ranks()
>>> graph.descend(2)
>>> graph.descent(20)
>>> hl = graph.hierarchy_list #get the list of hierarchy scores
>>> print len(hl) #descend has been run 21 times, plus the initial score
22
>>> print hl[0] #the first score is always 0
0
>>> print hl[-1] #the score after 21 descends will probably be 1.0
1.0
```

graphtools.dbgraph.**make_db**(*arrows_list*, *name=None*)
Make a SQLite database in the correct format for DBGraph.

Parameters

- **name** (*str*) – the name of the database; if no name is given, the database will be in-memory-only
- **arrow_list** (*list*) – the arrows of the graph as a list of lists of two ints, the first int representing the tail of the arrow and the second the head.

Returns the users table, arrows table and database connection as two sqlalchemy.schema.Table objects and a sqlalchemy.engine.base.Connection object, respectively

Return type tuple

For example,

```
>>> from graphtools.dbgraph import make_db
>>> users, arrows, conn = make_db([[1, 2], [2, 3]])
>>> from sqlalchemy.sql import select
>>> result = conn.execute(select([users]))
>>> result.fetchall()
[(1, 0), (2, 0), (3, 0)]
>>> result = conn.execute(select([arrows]))
>>> result.fetchall()
[(1, 1, 2), (2, 2, 3)]
```

2.4 listgraph Module

class graphtools.listgraph.**ListGraph**(*arrows_list*)
Bases: graphtools.gengraph.GenGraph

A subclass of GenGraph for graphs given as a list of arrows.

Parameters **arrows_list** (*list*) – a list of the arrows in the graph; each arrow is an ordered list of 2 vertices (any type) where the first vertex is the tail of the arrow and the second is the head.

We use the following simple example throughout.

```
>>> from graphtools.listgraph import ListGraph
>>> graph = ListGraph([['a', 'b'], ['b', 'c']])
>>> print graph.get_num_arrows()
2
>>> set(graph.get_vert_list()) == set(['a', 'b', 'c'])
True
>>> print graph.get_rank('a')
0
>>> graph.set_rank('b',2)
>>> graph.get_rank('b')
2
>>> graph.reset_ranks()
>>> graph.descend('a')
>>> graph.descent(20)
>>> hl = graph.hierarchy_list #get the list of hierarchy scores
>>> print len(hl) #descend has been run 21 times, plus the initial score
22
>>> print hl[0] #the first score is always 0
```

```
0
>>> print hl[-1] #the score after 21 descends will probably be 1.0
1.0
```

Graphs with isolated vertices (vertices with no neighbors) are not supported. Isolated vertices don't affect the hierarchy of the graph.

neighbors_in (*vert*)
Return the list of in neighbors of vertex *vert*.

Parameters **vert** (*vertex*) – the vertex to get the neighbors of

Returns the list of in neighbors of *vert*

Return type list

For example,

```
>>> graph.neighbors_in('a')
[]
>>> graph.neighbors_in('c')
['b']
```

neighbors_out (*vert*)
Return the list of out neighbors of vertex *vert*.

Parameters **vert** (*vertex*) – the vertex to get the neighbors of

Returns the list of out neighbors of *vert*

Return type list

For example,

```
>>> graph.neighbors_out('a')
['b']
>>> graph.neighbors_out('c')
[]
```

2.5 sagegraph Module

class graphtools.sagegraph.**SageGraph** (*dg*)
Bases: graphtools.gengraph.GenGraph

A subclass of GenGraph which wraps a Sage DiGraph object.

Parameters **dg** (*sage.graphs.digraph.DiGraph*) – the Sage DiGraph to run descent on

BIBLIOGRAPHY

[FHSN] M. Gupte, P. Shankar, J. Li, S. Muthukrishnan, L. Iftode, Finding hierarchy in directed online social networks.

PYTHON MODULE INDEX

g

INDEX