CGraph documentation

Bruno Kim Medeiros Cesar

May 25, 2013

Abstract

Contents

1 sorting **2**

2 list **2**

3 set **2**

4 graph **2**

5 graph_metric **2**

5.1 Constants 2

5.1.1 GRAPH_METRIC_TOLERANCE 2

5.1.2 GRAPH_METRIC_MAX_ITERATIONS 2

5.2 Component identification and extraction 3

5.2.1 graph_undirected_components 3

5.2.2 graph_directed_components 3

5.2.3 graph_num_components 3

5.2.4 graph_components 4

5.2.5 graph_components 4

5.3 Degree metrics 4

5.3.1 graph_degree 4

5.3.2 graph_directed_degree 4

5.4 Clustering metrics 5

5.4.1 graph_clustering 5

5.4.2 graph_num_triplets 5

5.4.3 graph_transitivity 5

5.5 Geodesic distance metrics 6

5.5.1 Definitions 6

5.5.2 graph_geodesic_distance 6

5.5.3 graph_geodesic_vertex 6

5.5.4 graph_geodesic_all 6

5.5.5 graph_geodesic_distribution 6

5.6 Centrality measures 6

5.6.1 graph_betweenness 6

5.6.2 graph_eigenvector 6

5.6.3 graph_pagerank . . . 6
5.6.4 graph_kcore . . . 6
5.7 Correlation measures . . . 6
5.7.1 graph_degree_matrix . . . 6
5.7.2 graph_neighbor_degree_vertex . . . 6
5.7.3 graph_neighbor_degree_all . . . 6
5.7.4 graph_knn . . . 6
5.7.5 graph_assortativity . . . 6

6 graph_layout 6
6.1 Types . . . 6
6.1.1 coord_t . . . 6
6.1.2 box_t . . . 6
6.1.3 circle_style_t . . . 6
6.1.4 path_style_t . . . 7
6.2 Layout . . . 7
6.2.1 graph_layout_random . . . 7
6.2.2 graph_layout_random_wout_overlap . . . 7
6.3 Printing . . . 8
6.3.1 graph_print_svg . . . 8
6.3.2 graph_print_svg_one_style . . . 8
6.3.3 graph_print_svg_some_styles . . . 8

1 `sorting`

2 `list`

3 `set`

4 `graph`

5 `graph_metric`

5.1 Constants

These constants are hard-coded to protect some numeric processes of hanging. They can be redefined during compilation, passing a flag such as
`-DGRAPH_METRIC_TOLERANCE=1E-3`.

5.1.1 `GRAPH_METRIC_TOLERANCE`

Error tolerance for numeric methods.

5.1.2 `GRAPH_METRIC_MAX_ITERATIONS`

Maximum number of iterations for numeric methods.

5.2 Component identification and extraction

5.2.1 `graph_undirected_components`

Label vertices' components treating edges as undirected.

Preconditions `label` must have dimension n.

Postconditions `label[i]` is the component ID of vertex v_i.

Return Number of components

For directed graphs, considers adjacencies as incidences. Labels start from 0 and are sequential with step 1. Component IDs are not ordered according to size.

5.2.2 `graph_directed_components`

Label vertices' components treating edges as directed. NOT IMPLEMENTED YET.

Preconditions `label` must have dimension n.

Postconditions `label[i]` is the component ID of vertex v_i.

Return Number of components

For undirected graphs, simply call `graph_undirected_components`. For directed graphs, two vertices v_i and v_j are in the same component if and only if

$$d(v_i, v_j) \neq \infty$$
$$d(v_j, v_i) \neq \infty$$

where $d(u, v)$ is the geodesic distance between them. In other words, they are in the same component if they are mutually reachable.

Labels start from 0 and are sequential with step 1. Component IDs are not ordered according to size.

5.2.3 `graph_num_components`

Extract number of components from label vector.

Preconditions

$n > 0$

`label` must have dimension n.

`label` must contain sequential IDs starting from 0.

Return Number of components

5.2.4 `graph_components`

Map components to vertices from label vector.

Preconditions
: $n > 0$
`label` must have dimension n.
`label` must contain sequential IDs starting from 0.
`comp` must have size `num_comp` and all sets should be already initialized.
`graph_num_components(g) == num_comp`

Postconditions
: If v_i is in component c_j, then
`label[i] == j` and
`set_contains(comp[j], i)` is true.

Return Number of components

5.2.5 `graph_components`

Creates a new graph from `g`'s largest component.

The guarantee of vertices' order ID is the same as `graph_subset`. If two or more components have the same maximum size, one will be chosen in an undefined way.

Return A new graph isomorphic to `g`'s largest component.

Memory deallocation
```
graph_t *largest = graph_components(g);
delete_graph(largest);
```

5.3 Degree metrics

5.3.1 `graph_degree`

List all vertices' degrees.

Preconditions `degree` must have dimension n.

Postconditions `degree[i]` is the degree of vertex v_i.

The degree of a directed graph's vertex is defined as the sum of incoming and outgoing edges.

5.3.2 `graph_directed_degree`

List all vertices' incoming and outgoing degrees.

Preconditions
: `g` must be directed. `in_degree` must have dimension n. `out_degree` must have dimension n.

Postconditions
: `in_degree[i]` is the number of incoming edges to vertex v_i. `out_degree[i]` is the number of outgoing edges from vertex v_i.

5.4 Clustering metrics

5.4.1 `graph_clustering`

List all vertices' local clustering.

Preconditions
`g` must be undirected.
`clustering` must have dimension n.

Postconditions `clustering[i]` is the local clustering coefficient of vertex v_i.

The local clustering coefficient is only defined for undirected graphs, and gives the ratio of edges between a vertex' neighbors and all possible edges.

Formally,

$$C_i = \frac{e_i}{\binom{k_i}{2}} = \frac{2e_i}{k_i(k_i-1)}$$

where

C_i is the local clustering coefficient of vertex v_i.

e_i is the number of edges between v_i's neighbors.

k_i is the degree of v_i.

If a vertex v_i has 0 or 1 adjacents, $C_i = 0$ by definition.

5.4.2 `graph_num_triplets`

Counts number of triplets and triangles (6 * number of closed triplets).

5.4.3 `graph_transitivity`

Compute the ratio between number of triangles and number of triplets.

5.5 Geodesic distance metrics

5.5.1 Definitions

5.5.2 `graph_geodesic_distance`

5.5.3 `graph_geodesic_vertex`

5.5.4 `graph_geodesic_all`

5.5.5 `graph_geodesic_distribution`

5.6 Centrality measures

5.6.1 `graph_betweenness`

5.6.2 `graph_eigenvector`

5.6.3 `graph_pagerank`

5.6.4 `graph_kcore`

5.7 Correlation measures

5.7.1 `graph_degree_matrix`

5.7.2 `graph_neighbor_degree_vertex`

5.7.3 `graph_neighbor_degree_all`

5.7.4 `graph_knn`

5.7.5 `graph_assortativity`

6 `graph_layout`

6.1 Types

6.1.1 `coord_t`

Euclidean coordinates in 2D.

6.1.2 `box_t`

Box (rectangle) definition in 2D, given by its SW and NE vertices in a positively oriented world frame, such as the screen. Images may have a negatively oriented frame, with y pointing down. It is necessary that `box.sw.y < box.ne.y` and `box.sw.x < box.ne.x`.

6.1.3 `circle_style_t`

SVG circle style.

`radius` Circle radius in pixels.

`width` Stroke width in pixels. This is added to the radius for total size.

`color` Array with 4 colors: red (R), green (G), blue (B) and alpha (A), lying between 0 and 255. $A = 0$ means totally transparent, and $A = 255$ means totally opaque.

6.1.4 `path_style_t`

SVG path style.

`type` Path type.

`from, to` Path origin and destination.

`control` Control point

`width` Stroke width in pixels.

`color` Array with 4 colors: red (R), green (G), blue (B) and alpha (A), lying between 0 and 255. $A = 0$ means totally transparent, and $A = 255$ means totally opaque.

For `style.type == GRAPH_STRAIGHT`, draws a straight line from origin to destination.

For `style.type == GRAPH_PARABOLA`, draws a parabola from origin to destination using the control point.

For `style.type == GRAPH_CIRCULAR`, draws the arc of a circle from origin to destination using the control point as the circle center.

6.2 Layout

6.2.1 `graph_layout_random`

Place points uniformly inside specified box.

Preconditions
`box` must be a valid box.
`p` must have dimension n.

Postconditions `p[i]` is a random coordinate inside `box`.

6.2.2 `graph_layout_random_wout_overlap`

Place points with specified radius uniformly avoiding overlap with probability t.

Preconditions
`radius` must be positive.
t must be a valid probability ($0 \geq t \geq 1$).
`p` must have dimension n.

Postconditions `p[i]` is a random coordinate.

The algorithm determines a box with size l such that, if n points with radius r are thrown within it, will not have any collision with probability t. The formula is derived in Math Exchange.

$$l = \frac{nr}{2}\sqrt{\frac{2\pi}{-\log(1-t)}}$$

6.3 Printing

6.3.1 `graph_print_svg`

Prints graph as SVG to file, using vertex coordinates given in p and with a style for each point and edge.

Preconditions
: `p` must have dimension n. `point_style` must have dimension n. `edge_style` must have dimension m.

Postconditions `filename` is a valid SVG file.

Edges are ordered according to vertices' order. In undirected graphs, an edge E_{ij} is considered only if $i < j$. In directed graphs, mutual edges will superimpose if `edge_style.type == GRAPH_STRAIGHT`.

6.3.2 `graph_print_svg_one_style`

Prints graph as SVG to file, using vertex coordinates given in p and with a single style for all points and edges.

Preconditions
: `p` must have dimension n.

Postconditions `filename` is a valid SVG file.

The edge style type is ignored, using only `GRAPH_STRAIGHT`.

6.3.3 `graph_print_svg_some_styles`

Prints graph as SVG to file, using vertex coordinates given in `p` and with a number of styles given. The mapping vertex→style is given in `ps`, and the mapping edge→style is given in `es`.

Preconditions
: `p` must have dimension n.
`ps` must have dimension n.
`es` must have dimension m.
`point_style` must have dimension `num_point_style`.
`edge_style` must have dimension `num_edge_style`.

Postconditions `filename` is a valid SVG file.

This function tries to avoid extensive memory utilization one just some styles are desired. If vertex v_i should have style S_j, then `ps[i] = j`. Ditto for edges.

Edge order is based on vertices order. In undirected edges, edge E_{ij} is considered only if $i < j$.