Package ‘graphframes’

October 13, 2022

Type Package
Title Interface for ‘GraphFrames’
Version 0.1.2
Maintainer Kevin Kuo <kevin.kuo@rstudio.com>
Description A ‘sparklyr’ <https://spark.rstudio.com/> extension that provides an R interface for ‘GraphFrames’ <https://graphframes.github.io/>. ‘GraphFrames’ is a package for ‘Apache Spark’ that provides a DataFrame-based API for working with graphs. Functionality includes motif finding and common graph algorithms, such as PageRank and Breadth-first search.

URL https://github.com/rstudio/graphframes
BugReports https://github.com/rstudio/graphframes/issues
License Apache License 2.0 | file LICENSE
Encoding UTF-8
LazyData true
RoxygenNote 6.1.0
Imports sparklyr, tibble, forge
Suggests testthat, covr, dplyr
NeedsCompilation no
Author Kevin Kuo [aut, cre] (https://orcid.org/0000-0001-7803-7901)
Repository CRAN
Date/Publication 2018-10-30 19:20:03 UTC

R topics documented:

  gf_bfs ................................................................. 2
  gf_cache ............................................................ 3
  gf_chain ............................................................. 3
  gf_connected_components ........................................ 4
  gf_degrees .......................................................... 4
  gf_edges ............................................................. 5
gf_bfs

Breadth-first search (BFS)

Description
Breadth-first search (BFS)

Usage

\[
gf_bfs(x, \text{from_expr}, \text{to_expr}, \text{max_path_length} = 10, \text{edge_filter} = \text{NULL}, \ldots)
\]

Arguments

- **x**: An object coercable to a GraphFrame (typically, a `gf_graphframe`).
- **from_expr**: Spark SQL expression specifying valid starting vertices for the BFS.
- **to_expr**: Spark SQL expression specifying valid target vertices for the BFS.
- **max_path_length**: Limit on the length of paths.
- **edge_filter**: Spark SQL expression specifying edges which may be used in the search.
- **...**: Optional arguments, currently not used.
gf_cache

Examples

```r
## Not run:
g <- gf_friends(sc)
gf_bfs(g, from_expr = "name = 'Esther'", to_expr = "age < 32")
## End(Not run)
```

gf_cache

Cache the GraphFrame

Description

Cache the GraphFrame

Usage

`gf_cache(x)`

Arguments

`x` An object coercable to a GraphFrame (typically, a `gf_graphframe`).

gf_chain

Chain graph

Description

Returns a chain graph of the given size with Long ID type. The vertex IDs are 0, 1, ..., n-1, and the edges are (0, 1), (1, 2), ..., (n-2, n-1).

Usage

`gf_chain(sc, n)`

Arguments

`sc` A Spark connection.

`n` Size of the graph to return.

Examples

```r
## Not run:
gf_chain(sc, 5)
## End(Not run)
```
gf_connected_components

*Connected components*

**Description**
Computes the connected component membership of each vertex and returns a DataFrame of vertex information with each vertex assigned a component ID.

**Usage**
gf_connected_components(x, broadcast_threshold = 1000000L, algorithm = c("graphframes", "graphx"), checkpoint_interval = 2L, ...)

**Arguments**
- **x**: An object coercable to a GraphFrame (typically, a gf_graphframe).
- **broadcast_threshold**: Broadcast threshold in propagating component assignments.
- **algorithm**: One of 'graphframes' or 'graphx'.
- **checkpoint_interval**: Checkpoint interval in terms of number of iterations.
- **...**: Optional arguments, currently not used.

**Examples**
```r
## Not run:
# checkpoint directory is required for gf_connected_components()
spark_set_checkpoint_dir(sc, tempdir())
g <- gf_friends(sc)
gf_connected_components(g)
## End(Not run)
```

gf_degrees

*Degrees of vertices*

**Description**
Degrees of vertices

**Usage**
gf_degrees(x)
**gf_edges**

**Arguments**

- *x*  
  An object coercable to a GraphFrame (typically, a `gf_graphframe`).

---

**gf_edges**  
*Extract edges DataFrame*

---

**Description**

Extract edges DataFrame

**Usage**

```
gf_edges(x)
```

**Arguments**

- *x*  
  An object coercable to a GraphFrame (typically, a `gf_graphframe`).

---

**gf_edge_columns**  
*Edges column names*

---

**Description**

Edges column names

**Usage**

```
gf_edge_columns(x)
```

**Arguments**

- *x*  
  An object coercable to a GraphFrame (typically, a `gf_graphframe`).
Motif finding uses a simple Domain-Specific Language (DSL) for expressing structural queries. For example, `gf_find(g, "(a)-[e]->(b); (b)-[e2]->(a)")` will search for pairs of vertices a,b connected by edges in both directions. It will return a DataFrame of all such structures in the graph, with columns for each of the named elements (vertices or edges) in the motif. In this case, the returned columns will be in order of the pattern: "a, e, b, e2."

Usage

gf_find(x, pattern)

Arguments

x
An object coercable to a GraphFrame (typically, a `gf_graphframe`).

pattern
pattern specifying a motif to search for

Examples

```r
## Not run:
gf_friends(sc) %>%
gf_find("(a)-[e]->(b); (b)-[e2]->(a)")

## End(Not run)
```

Graph of friends in a social network.

Description

Graph of friends in a social network.

Usage

gf_friends(sc)

Arguments

sc
A Spark connection.
gf_graphframe

Examples

## Not run:
library(sparklyr)
sc <- spark_connect(master = "local")
gf_friends(sc)

## End(Not run)

---

gf_graphframe Create a new GraphFrame

Description

Create a new GraphFrame

Usage

gf_graphframe(vertices = NULL, edges)

Arguments

vertices A tbl_spark representing vertices.
edges A tbl_psark representing edges.

Examples

## Not run:
library(sparklyr)
sc <- spark_connect(master = "local", version = "2.3.0")
v_tbl <- sdf_copy_to(
  sc, data.frame(id = 1:3, name = LETTERS[1:3])
)
e_tbl <- sdf_copy_to(
  sc, data.frame(src = c(1, 2, 2), dst = c(2, 1, 3),
                  action = c("love", "hate", "follow"))
)
gf_graphframe(v_tbl, e_tbl)
gf_graphframe(edges = e_tbl)

## End(Not run)
gf_grid_ising_model

Generate a grid Ising model with random parameters

Description
Generate a grid Ising model with random parameters

Usage

gf_grid_ising_model(sc, n, v_std = 1, e_std = 1)

Arguments

- **sc**: A Spark connection.
- **n**: Length of one side of the grid. The grid will be of size n x n.
- **v_std**: Standard deviation of normal distribution used to generate vertex factors "a". Default of 1.0.
- **e_std**: Standard deviation of normal distribution used to generate edge factors "b". Default of 1.0.

Details
This method generates a grid Ising model with random parameters. Ising models are probabilistic graphical models over binary variables $x_i$. Each binary variable $x_i$ corresponds to one vertex, and it may take values -1 or +1. The probability distribution $P(X)$ (over all $x_i$) is parameterized by vertex factors $a_i$ and edge factors $b_{ij}$:

$$P(X) = \frac{1}{Z} \exp\left[ \sum_i a_i x_i + \sum_{ij} b_{ij} x_i x_j \right]$$

Value
GraphFrame. Vertices have columns "id" and "a". Edges have columns "src", "dst", and "b". Edges are directed, but they should be treated as undirected in any algorithms run on this model. Vertex IDs are of the form "i,j". E.g., vertex "1,3" is in the second row and fourth column of the grid.

Examples

```r
## Not run:
gf_grid_ising_model(sc, 5)
## End(Not run)
```
gf_in_degrees

In-degrees of vertices

Description

In-degrees of vertices

Usage

gf_in_degrees(x)

Arguments

x An object coercable to a GraphFrame (typically, a gf_graphframe).

gf_lpa

Label propagation algorithm (LPA)

Description

Run static Label Propagation for detecting communities in networks. Each node in the network is initially assigned to its own community. At every iteration, nodes send their community affiliation to all neighbors and update their state to the mode community affiliation of incoming messages. LPA is a standard community detection algorithm for graphs. It is very inexpensive computationally, although (1) convergence is not guaranteed and (2) one can end up with trivial solutions (all nodes are identified into a single community).

Usage

gf_lpa(x, max_iter, ...)

Arguments

x An object coercable to a GraphFrame (typically, a gf_graphframe).
max_iter Maximum number of iterations.
... Optional arguments, currently not used.

Examples

```r
## Not run:
g <- gf_friends(sc)
gf_lpa(g, max_iter = 5)
## End(Not run)
```
**gf_out_degrees**

*Out-degrees of vertices*

**Description**

Out-degrees of vertices

**Usage**

\[ \text{gf_out_degrees}(x) \]

**Arguments**

- `x` An object coercable to a GraphFrame (typically, a `gf_graphframe`).

---

**gfPagerank**

*PageRank*

**Description**

PageRank

**Usage**

\[ \text{gfPagerank}(x, \text{tol} = \text{NULL}, \text{reset}\_\text{probability} = 0.15, \text{max}\_\text{iter} = \text{NULL}, \text{source}\_\text{id} = \text{NULL}, \ldots) \]

**Arguments**

- `x` An object coercable to a GraphFrame (typically, a `gf_graphframe`).
- `tol` Tolerance.
- `reset_probability` Reset probability.
- `max_iter` Maximum number of iterations.
- `source_id` (Optional) Source vertex for a personalized pagerank.
- `...` Optional arguments, currently not used.

**Examples**

```
## Not run:
g <- gf_friends(sc)
gfPagerank(g, reset_probability = 0.15, tol = 0.01)
## End(Not run)
```
gf_persist

Persist the GraphFrame

Description

Persist the GraphFrame

Usage

gf_persist(x, storage_level = "MEMORY_AND_DISK")

Arguments

- **x**: An object coercable to a GraphFrame (typically, a gf_graphframe).
- **storage_level**: The storage level to be used. Please view the Spark Documentation for information on what storage levels are accepted.

gf_register

Register a GraphFrame object

Description

Register a GraphFrame object

Usage


gf_register(x)

Arguments

- **x**: An object coercable to a GraphFrame (typically, a gf_graphframe).
gf_scc

**Strongly connected components**

**Description**

Compute the strongly connected component (SCC) of each vertex and return a DataFrame with each vertex assigned to the SCC containing that vertex.

**Usage**

```
gf_scc(x, max_iter, ...)
```

**Arguments**

- `x` An object coercable to a GraphFrame (typically, a `gf_graphframe`).
- `max_iter` Maximum number of iterations.
- `...` Optional arguments, currently not used.

**Examples**

```r
## Not run:
g <- gf_friends(sc)
gf_scc(g, max_iter = 10)
## End(Not run)
```

gf_shortest_paths

**Shortest paths**

**Description**

Computes shortest paths from every vertex to the given set of landmark vertices. Note that this takes edge direction into account.

**Usage**

```
gf_shortest_paths(x, landmarks, ...)
```

**Arguments**

- `x` An object coercable to a GraphFrame (typically, a `gf_graphframe`).
- `landmarks` IDs of landmark vertices.
- `...` Optional arguments, currently not used.
gf_star

Examples

## Not run:
g <- gf_friends(sc)
gf_shortest_paths(g, landmarks = c("a", "d"))
## End(Not run)

gf_star Generate a star graph

Description

Returns a star graph with Long ID type, consisting of a central element indexed 0 (the root) and the n other leaf vertices 1, 2, ..., n.

Usage

gf_star(sc, n)

Arguments

sc A Spark connection.
n The number of leaves.

Examples

## Not run:
gf_star(sc, 5)
## End(Not run)

gf_triangle_count Computes the number of triangles passing through each vertex.

Description

This algorithm ignores edge direction; i.e., all edges are treated as undirected. In a multigraph, duplicate edges will be counted only once.

Usage

gf_triangle_count(x, ...)

Arguments

x An object coercable to a GraphFrame (typically, a gf_graphframe).
... Optional arguments, currently not used.
## Examples

```r
## Not run:
g <- gf_friends(sc)
gf_triangle_count(g)
## End(Not run)
```

---

**gf_triplets**

### Description

Triplets of graph

### Usage

`gf_triplets(x)`

### Arguments

- **x**: An object coercable to a GraphFrame (typically, a `gf_graphframe`).

---

**gf_two_blobs**

### Description

Two densely connected blobs (vertices 0->n-1 and n->2n-1) connected by a single edge (0->n).

### Usage

`gf_two_blobs(sc, blob_size)`

### Arguments

- **sc**: A Spark connection.
- **blob_size**: The size of each blob.

### Examples

```r
## Not run:
gf_two_blobs(sc, 3)
## End(Not run)
```
**gf_unpersist**

Unpersist the GraphFrame

**Usage**

gf_unpersist(x, blocking = FALSE)

**Arguments**

- **x**
  - An object coercable to a GraphFrame (typically, a `gf_graphframe`).
- **blocking**
  - whether to block until all blocks are deleted

**gf_vertex_columns**

Vertices column names

**Usage**

gf_vertex_columns(x)

**Arguments**

- **x**
  - An object coercable to a GraphFrame (typically, a `gf_graphframe`).

**gf_vertices**

Extract vertices DataFrame

**Usage**

gf_vertices(x)

**Arguments**

- **x**
  - An object coercable to a GraphFrame (typically, a `gf_graphframe`).
spark_graphframe

Retrieve a GraphFrame

Description
Retrieve a GraphFrame

Usage
spark_graphframe(x, ...)

Arguments
x An object coercable to a GraphFrame (typically, a gf_graphframe).
... additional arguments, not used
Index

gf_bfs, 2
gf_cache, 3
gf_chain, 3
gf_connected_components, 4
gf_degrees, 4
gf_edge_columns, 5
gf_edges, 5
gf_find, 6
gf_friends, 6
gf_graphframe, 7
gf_grid_ising_model, 8
gf_in_degrees, 9
gf_lpa, 9
gf_out_degrees, 10
gf_pagerank, 10
gf_persist, 11
gf_register, 11
gf_scc, 12
gf_shortest_paths, 12
gf_star, 13
gf_triangle_count, 13
gf_triplets, 14
gf_two_blobs, 14
gf_unpersist, 15
gf_vertex_columns, 15
gf_vertices, 15

spark_graphframe, 16