PegasusN

User guide

August 29, 2017

Contents

1 Overview

1.1 General Information

- PegasusN: Peta-Scala Graph Mining System
- Version: 3.0
- Date: Sep. 1st, 2017
- Authors: Chiwan Park, Ha-Myung Park, U Kang

PegasusN is a Peta-scale graph mining system on hadoop and spark. It computes the degree distribution, PageRank, RWR(Random Walk with Restart) scores, radii/diameter, connected components, subgraphs that match a query graph including a triangle from very large graphs with more than billions of nodes and edges. The details of PegasusN can be found in the following papers:

References will be placed here

If your work uses or refers to PegasusN, please cite the papers using the following bibtex entries:

Bibtex will be placed here

For more information and demo, visit PegasusN homepage: `http://datalab.snu.ac.kr/pegasus-n`

1.2 License

PegasusN is licensed under Apache License, Version 2.0. To obtain a copy of the license, visit `http://www.apache.org/licenses/LICENSE-2.0`.

If you use PegasusN for research or commercial purposes, please let me know your institution(company) and whether it's ok to mention it among the users of PegasusN.

2 Installation

3 Running

This section guides you how to run PegasusN, from preparing input to running commands.

3.1 Preparing Graph

PegasusN works on graphs with TAB-separated plain text format. Each line contains the source and destination node id of an edge. The node id starts from 0. For example, here is an example graph containing 16 nodes and 14 edges.

```
0	1
1	2
1	3
3	4
3	6
5	6
6	7
6	8
6	9
10	11
10	12
10	13
10	14
10	15
```

For some algorithms which require weighted graphs such as the shortest path computation, the weights for each edge should be provided.

```
0	1	1.0
1	2	0.226
1	3	0.17
3	4	0.53
```

3.2 Running individual scripts

You can run each algorithm with corresponding shell scripts. In Table 1 is the list of the algorithms, shell scripts, and corresponding demo scripts.

The 'running script' is used to run each algorithm. It requires several parameters which will be described next. The 'demo script' is the script to tell you how to

Table 1: Algorithms and scripts

Algorithm	Running Script	Demo Script
Degree	degree_spark.sh degree_hadoop.sh	degree_spark_demo.sh degree_hadoop_demo.sh
PageRank/RWR	pagerank_spark.sh pagerank_hadoop.sh	pagerank_spark_demo.sh pagerank_hadoop_demo.sh
Single source shortest path	sssp_spark.sh sssp_hadoop.sh	sssp_spark_demo.sh sssp_hadoop_demo.sh
Label propagation	labelprop_spark.sh labelprop_hadoop.sh	labelprop_spark_demo.sh labelprop_hadoop_demo.sh
Radii/Diameter	diameter_spark.sh diameter_hadoop.sh	diameter_spark_demo.sh diameter_hadoop_demo.sh
PACC (connected component)	pacc_spark.sh pacc_hadoop.sh	pacc_spark_demo.sh pacc_hadoop_demo.sh
PTE (triangle)	pte_spark.sh pte_hadoop.sh	pte_spark_demo.sh pte_hadoop_demo.sh
PSE (subgraph)	pse_spark.sh pse_hadoop.sh	pse_spark_demo.sh pse_hadoop_demo.sh
TeG-RMAT (graph generator)	teg_rmat_spark.sh teg_rmat_hadoop.sh	teg_rmat_spark_demo.sh teg_rmat_hadoop_demo.sh
TeG-Kronecker (graph generator)	teg_krnk_spark.sh teg_krnk_hadoop.sh	teg_krnk_spark_demo.sh teg_krnk_hadoop_demo.sh
TeG-Random (graph generator)	teg_rand_spark.sh teg_rand_hadoop.sh	teg_rand_spark_demo.sh teg_rand_hadoop_demo.sh
NetRay-distribution (distribution plot)	netray_distr_spark.sh netray_distr_hadoop.sh	netray_distr_spark_demo.sh netray_distr_hadoop_demo.sh
NetRay-scatter (scatter plot)	netray_scatter_spark.sh netray_scatter_hadoop.sh	netray_scatter_spark_demo.sh netray_scatter_hadoop_demo.sh
NetRay-spy (spy plot)	netray_spy_spark.sh netray_spy_hadoop.sh	netray_spy_spark_demo.sh netray_spy_hadoop_demo.sh

use the 'running script'. The demo scripts do not require any parameters. Just type the demo script name and it will run.

3.2.1 Degree

To run degree computation, you need to do the two things:

- Copy the graph edge file to a HDFS directory, say `degree_edge`. The edge file is a plain text file where each line is SRC_ID TAB DST_ID format.
- Execute `sh degree_spark.sh` or `sh degree_hadoop.sh`

The syntax of `degree_spark.sh` is:

```
sh degree_spark.sh [OPTION...]  SOURCE DEST DIRECTION

SOURCE:  edge file/directory path
DEST:  output directory path
DIRECTION:  direction of degrees (in, out, both)

OPTION...
 -p, --parallelism  the number of aggregate tasks (default:
                    20).
```

ex: `sh degree_spark.sh -p 120 degree_edge degree_result in`

The syntax of `degree_hadoop.sh` is:

```
sh degree_hadoop.sh [OPTION...]  SOURCE DEST DIRECTION

SOURCE:  edge file/directory path in HDFS
DEST:  output directory path in HDFS
DIRECTION:  direction of degrees (in, out, both)

OPTION...
 -r, --num-reduce-tasks  the number of reduce tasks (default:  20).
```

ex: `sh degree_hadoop.sh -r 60 degree_edge degree_result out`

Each line of the output contains the degree of each node in the format of:

```
node id TAB degree of the node
```

Working examples of running degree computation on hadoop and spark are in `degree_spark_demo.sh` and `degree_hadoop_demo.sh`, respectively.

3.2.2 PageRank/RWR

PegasusN provides the PageRank and the Random Walk with Restart (RWR) computation with single script. To run PageRank or RWR algorithm, you need to do the two things:

- Copy the graph edge file to a HDFS directory, say `pagerank_edge`. The edge file is a plain text file where each line is SRC_ID TAB DST_ID format.

- Execute `sh pagerank_spark.sh` or `sh pagerank_hadoop.sh`

The syntax of `pagerank_spark.sh` is:

```
sh pagerank_spark.sh [OPTION...]  SOURCE DEST

SOURCE:  edge file/directory path
DEST:  output directory path


OPTION...
 -m, --max-iterations  the number of maximum iterations (default:
                       40).
 -b, --num-blocks      the number of vector blocks (default:  20).
 -a, --alpha           the damping factor of random walker
                       (default:  0.85).
 -p, --personalized    if this flag is true, the algorithm
                       computes RWR scores (personalized PageRank
                       scores) instead of normal PageRank scores
                       (default:  false).
 -s, --source-node     the source node of RWR algorithm.
 -t, --threshold       the degree threshold to classify the
                       sub-graph into two regions:  sparse region
                       and dense region (default:  120).
```

ex: `sh pagerank_spark.sh -b 30 -p true -s 377 -t 110 pagerank_edge pagerank_result`

The syntax of `pagerank_hadoop.sh` is:

```
sh pagerank_hadoop.sh [OPTION...]  SOURCE DEST

SOURCE:  edge file/directory path
DEST:  output directory path


OPTION...
```

```
 -m, --max-iterations  the number of maximum iterations (default:
                       40).
 -b, --num-blocks      the number of vector blocks (default:  20).
 -a, --alpha           the damping factor of random walker
                       (default:  0.85).
 -p, --personalized    if this flag is true, the algorithm
                       computes RWR scores (personalized PageRank
                       scores) instead of normal PageRank scores
                       (default:  false).
 -s, --source-node     the source node of RWR algorithm.
 -t, --threshold       the degree threshold to classify the
                       sub-graph into two regions:  sparse region
                       and dense region (default:  120).
```

ex: `sh pagerank_hadoop.sh -b 30 -t 100 pagerank_edge pagerank_result`

Each line of the output contains the PageRank score of each node in the format of:

`node id TAB the PageRank score of the node`

Working examples of running PageRank or RWR computation on hadoop and spark are in `pagerank_spark_demo.sh` and `pagerank_hadoop_demo.sh`, respectively.

3.2.3 Single Source Shortest Path

To run the single source shortest path algorithm, you need to do the two things:

- Copy the graph edge file to a HDFS directory, say `sssp_edge`. The edge file is a plain text file where each line is SRC_ID TAB DST_ID TAB WEIGHT format.
- Execute `sh sssp_spark.sh` or `sh sssp_hadoop.sh`

The syntax of `sssp_spark.sh` is:

```
sh sssp_spark.sh [OPTION...]  SOURCE DEST START

SOURCE:  edge file/directory path
DEST:  output directory path
START:  the id of source node


OPTION...
```

```
 -m, --max-iterations  the number of maximum iterations (default:
                       40).
 -b, --num-blocks      the number of vector blocks (default:  20).
 -t, --threshold       the degree threshold to classify the
                       sub-graph into two regions:  sparse region
                       and dense region (default:  120).
```

ex: `sh sssp_spark.sh -b 30 -m 50 -t 50 sssp_edge sssp_result 1772`

The syntax of `sssp_hadoop.sh` is:

`sh sssp_hadoop.sh [OPTION...]  SOURCE DEST START`

```
SOURCE:  edge file/directory path
DEST:  output directory path
START:  the id of source node


OPTION...
 -m, --max-iterations  the number of maximum iterations (default:
                       40).
 -b, --num-blocks      the number of vector blocks (default:  20).
 -t, --threshold       the degree threshold to classify the
                       sub-graph into two regions:  sparse region
                       and dense region (default:  120).
```

ex: `sh sssp_hadoop.sh -b 30 -m 100 -t 110 sssp_edge sssp_result 337`

Each line of the output contains the shortest distance of each node from the source node in the format of:

`node id TAB the distance of the node from the source node`

Working examples of running the single source shortest path computation on hadoop and spark are in `sssp_spark_demo.sh` and `sssp_hadoop_demo.sh`, respectively.

3.2.4 Label Propagation

To run the label propagation algorithm, you need to do the two things:

- Copy the graph edge file to a HDFS directory, say `labelprop_edge`. The edge file is a plain text file where each line is SRC_ID TAB DST_ID format.
- Execute `sh labelprop_spark.sh` or `sh labelprop_hadoop.sh`

The syntax of `labelprop_spark.sh` is:

```
sh labelprop_spark.sh [OPTION...]  SOURCE DEST

SOURCE:  edge file/directory path
DEST:  output directory path


OPTION...
 -m, --max-iterations  the number of maximum iterations (default:
                       40).
 -b, --num-blocks      the number of vector blocks (default:  20).
 -t, --threshold       the degree threshold to classify the
                       sub-graph into two regions:  sparse region
                       and dense region (default:  120).
```

ex: `sh labelprop_spark.sh -b 30 -m 50 -t 50 labelprop_edge labelprop_result`

The syntax of `labelprop_hadoop.sh` is:

```
sh labelprop_hadoop.sh [OPTION...]  SOURCE DEST

SOURCE:  edge file/directory path
DEST:  output directory path


OPTION...
 -m, --max-iterations  the number of maximum iterations (default:
                       40).
 -b, --num-blocks      the number of vector blocks (default:  20).
 -t, --threshold       the degree threshold to classify the
                       sub-graph into two regions:  sparse region
                       and dense region (default:  120).
```

ex: `sh labelprop_hadoop.sh -b 30 -m 100 -t 110 labelprop_edge labelprop_result`

Each line of the output contains the assigned label of each node in the format of:

`node id TAB the assigned label of the node`

Working examples of running the label propagation algorithm on hadoop and spark are in `labelprop_spark_demo.sh` and `labelprop_hadoop_demo.sh`, respectively.

3.2.5 Radii/Diameter

To compute effective diameter of graph, you need to do the two things:

- Copy the graph edge file to a HDFS directory, say `diameter_edge`. The edge file is a plain text file where each line is SRC_ID TAB DST_ID format.
- Execute `sh diameter_spark.sh` or `sh diameter_hadoop.sh`

The syntax of `diameter_spark.sh` is:

```
sh diameter_spark.sh [OPTION...]  SOURCE DEST

SOURCE:  edge file/directory path
DEST:  output directory path


OPTION...
 -s, --symmetric        the algorithm makes the input graph
                        symmetric if this flag is true (default:
                        false).
 -m, --max-iterations   the number of maximum iterations (default:
                        40).
 -b, --num-blocks       the number of vector blocks (default:  20).
 -t, --threshold        the degree threshold to classify the
                        sub-graph into two regions:  sparse region
                        and dense region (default:  120).
```

ex: `sh diameter_spark.sh -s true -b 30 -m 50 -t 50 diameter_edge diameter_result`

The syntax of `diameter_hadoop.sh` is:

```
sh diameter_hadoop.sh [OPTION...]  SOURCE DEST

SOURCE:  edge file/directory path
DEST:  output directory path


OPTION...
 -s, --symmetric        the algorithm makes the input graph
                        symmetric if this flag is true (default:
                        false).
 -m, --max-iterations   the number of maximum iterations (default:
                        40).
 -b, --num-blocks       the number of vector blocks (default:  20).
 -t, --threshold        the degree threshold to classify the
                        sub-graph into two regions:  sparse region
                        and dense region (default:  120).
```

ex: `sh diameter_hadoop.sh -b 30 -m 50 -t 50 diameter_edge diameter_result`

Each line of the output contains the 90% effective radius of each node in the format of:

`node id TAB the 90% effective radius of the node`

The algorithm prints the average diameter and the 90% effective diameter of the input graph on the screen.

Working examples of running the radii/diameter computation on hadoop and spark are in `diameter_spark_demo.sh` and `diameter_hadoop_demo.sh`, respectively.

3.2.6 PACC

To run PACC, you need to do the two things:

- Copy the graph edge file to a HDFS directory, say `pacc_edge`. The edge file is a plain text file where each line is SRC_ID TAB DST_ID format.
- Execute `sh pacc_spark.sh` or `sh pacc_hadoop.sh`

The syntax of `pacc_spark.sh` is:

```
sh pacc_spark.sh [OPTION...]  SOURCE DEST

SOURCE:  edge file/directory path
DEST:  output directory path

OPTION...
 -p, --num-partitions  the number of partitions (default:  80).
 -t, --threthold       pacc run an in-memory cc algorithm if the
                       remaining number of edges is below this
                       value (default:  1000000).
```

ex: `sh pacc_spark.sh -p 120 -t 20000000 pacc_edge pacc_result`

The syntax of `pacc_hadoop.sh` is:

```
sh pacc_hadoop.sh [OPTION...]  SOURCE DEST

SOURCE:  edge file/directory path in HDFS
DEST:  output directory path in HDFS

OPTION...
```

```
 -r, --num-reduce-tasks   the number of reduce tasks (default:  1).
 -p, --num-partitions     the number of partitions (default:  the
                          number of reduce tasks).
 -t, --threthold          pacc run an in-memory cc algorithm if the
                          remaining number of edges is below this
                          value (default:  1000000).
```

ex: `sh pacc_hadoop.sh -r 60 -p 120 -t 20000000 pacc_edge pacc_result`

Each line of the output contains the component id of each node in the format of:

`node id TAB component id of the node`

where the component id is the minimum node id in the component.

Working examples of running PACC on hadoop and spark are in `pacc_spark_demo.sh` and `pacc_hadoop_demo.sh`, respectively.

3.2.7 PTE

To run PTE, you need to do the two things:

- Copy the graph edge file to a HDFS directory, say `pte_edge`. The edge file is a plain text file where each line is SRC_ID TAB DST_ID format.
- Execute `sh pte_spark.sh` or `sh pte_hadoop.sh`

The syntax of `pte_spark.sh` is:

```
sh pte_spark.sh [-c NUM_COLORS] SOURCE [DEST]

SOURCE:  edge file/directory path
DEST:  output directory path.  If DEST is not given, PTE operates in
counting mode.
NUM_COLORS:  the number of colors (default:  10)
```

ex: `sh pte_spark.sh -c 12 pte_edge pte_result`

The syntax of `pte_hadoop.sh` is:

```
sh pte_hadoop.sh [OPTION...]  SOURCE [DEST]

SOURCE:  edge file/directory path
DEST:  output directory path.  If DEST is not given, PTE operates in
counting mode.
```

```
OPTION...
 -r, --num-reduce-tasks  the number of reduce tasks (default:  1).
 -c, --num-colors        the number of colors (default:  10).
```

ex: `sh pte_spark.sh -r 120 -c 12 pte_edge pte_result`

Each line of the output contains a triangle in the format of:

`1st node id TAB 2nd node id TAB 3rd node id`

where the order of nodes is arbitrary.

Working examples of running PTE on hadoop and spark are in `pte_spark_demo.sh` and `pte_hadoop_demo.sh`, respectively.

3.2.8 PSE

To run PSE, you need to do the two things:

- Copy the graph edge file to a HDFS directory, say `pse_edge`. The edge file is a plain text file where each line is SRC_ID TAB DST_ID format.
- Execute `sh pse_spark.sh` or `sh pse_hadoop.sh`

The syntax of `pse_spark.sh` is:

```
sh pse_spark.sh [-c NUM_COLORS] SOURCE PATTERN_STRING [DEST]

SOURCE:  edge file/directory path
DEST:  output directory path.  If DEST is not given, PSE operates in
counting mode.
NUM_COLORS:  the number of colors (default:  10)
PATTERN_STRING:  query pattern graph in a binary string format that
represents the upper triangle of the query graph's adjacency matrix.
Some example graphs are as follows:

111:  triangle
111100:  triangle with a tail
110011:  square
111111:  clique of four nodes
1100010011:  pentagon
1111111111:  clique of five nodes
```

ex: `sh pse_spark.sh -c 10 pse_edge 110011 pse_result`

The syntax of `pse_hadoop.sh` is:

```
sh pse_hadoop.sh [OPTION...]  SOURCE PATTERN_STRING [DEST]

SOURCE:  edge file/directory path
DEST:  output directory path.  If DEST is not given, PSE operates in
counting mode.
PATTERN_STRING:  query pattern graph in a binary string format that
represents the upper triangle of the query graph's adjacency matrix.
Some example graphs are as follows:

111:  triangle
111100:  triangle with a tail
110011:  square
111111:  clique of four nodes
1100010011:  pentagon
1111111111:  clique of five nodes


OPTION...
 -r, --num-reduce-tasks  the number of reduce tasks (default:  1).
 -c, --num-colors        the number of colors (default:  10).
```

ex: `sh pse_spark.sh -r 120 -c 10 pse_edge 110011 pse_result`

Each line of the output contains a subgraph that match the query graph in the format of:

1st node id TAB 2nd node id TAB $\cdots$ TAB n-th node id

where the k-th node matches the k-th query node, and n is the node number in the query graph.

Working examples of running PSE on hadoop and spark are in `pse_spark_demo.sh` and `pse_hadoop_demo.sh`, respectively.

3.2.9 TeG-RMAT

To run TeG-RMAT, execute `sh teg_rmat_spark.sh` or `sh teg_rmat_hadoop.sh`.

The syntax of `teg_rmat_spark.sh` is:

```
sh teg_rmat_spark.sh LEVEL NUM_EDGES NUM_TASKS PROBABILITIES NOISE DEST

DEST:  output directory path
LEVEL:  level of recursion
NUM_EDGES:  the number of edges
NUM_TASKS:  degree of parallelism (the nuber of tasks to run)
PROBABILITIES:  the probabilities of quarters.  Four numbers whose
```

```
sum is 1
NOISE for smoothing.  This MUST be below than the second and the third
probabilities and 0.5
```

ex: `sh teg_rmat_spark.sh 18 2000000 240 0.5 0.2 0.2 0.1 0.1 teg_rmat_graph`

The syntax of `teg_rmat_hadoop.sh` is:

```
sh teg_rmat_hadoop.sh [-r NUM REDUCE TASKS] LEVEL NUM EDGES NUM TASKS
PROBABILITIES NOISE DEST

DEST:  output directory path
LEVEL:  level of recursion
NUM EDGES:  the number of edges
NUM TASKS:  degree of parallelism (the nuber of tasks to run)
PROBABILITIES:  the probabilities of quarters.  Four numbers whose
sum is 1
NOISE for smoothing.  This MUST be below than the second and the third
probabilities and 0.5
NUM REDUCE TASKS:  the number of reduce tasks (default:  1)
```

ex: `sh teg_rmat_hadoop.sh -r 80 18 2000000 240 0.5 0.2 0.2 0.1 0.1 teg_rmat_graph`

Each line of the output contains an edge in the format of

source node id TAB destination node id

Working examples of running TeG-RMAT on hadoop and spark are in `teg_rmat_spark_demo.sh` and `teg_rmat_hadoop_demo.sh`, respectively.

3.2.10 TeG-Kronecker

To run TeG-Kronecker, you need to do the two things:

- Prepare a seed graph edge file, say `kronecker_seed`. The edge file is a plain text file where each line is SRC_ID TAB DST_ID format.
- Execute `sh teg_krnk_spark.sh` or `sh teg_krnk_hadoop.sh`.

The syntax of `teg_krnk_spark.sh` is:

```
sh teg_krnk_spark.sh LEVEL NUM TASKS SEED DEST

SEED:  seed graph file path
DEST:  output directory path
LEVEL:  level of recursion
```

```
NUM TASKS:  degree of parallelism (the nuber of tasks to run)
```

ex: `sh teg_krnk_spark.sh 18 240 kronecker_seed teg_krnk_graph`

The syntax of `teg_krnk_hadoop.sh` is:

```
sh teg_krnk_hadoop.sh [-r NUM REDUCE TASKS] LEVEL NUM EDGES NUM TASKS
PROBABILITIES NOISE DEST

DEST:  output directory path
LEVEL:  level of recursion
NUM EDGES:  the number of edges
NUM TASKS:  degree of parallelism (the nuber of tasks to run)
PROBABILITIES:  the probabilities of quarters.  Four numbers whose
sum is 1
NOISE for smoothing.  This MUST be below than the second and the third
probabilities and 0.5
NUM REDUCE TASKS:  the number of reduce tasks (default:  1)
```

ex: `sh teg_krnk_hadoop.sh -r 80 18 240 kronecker_seed teg_krnk_graph`

Each line of the output contains an edge in the format of

`source node id TAB destination node id`

Working examples of running TeG-Kronecker on hadoop and spark are in `teg_krnk_spark_demo.sh` and `teg_krnk_hadoop_demo.sh`, respectively.

3.2.11 TeG-Random

To run TeG-Random, execute `sh teg_rand_spark.sh` or `sh teg_rand_hadoop.sh`.

The syntax of `teg_rand_spark.sh` is:

```
sh teg_rand_spark.sh NUM NODES NUM EDGES NUM TASKS DEST

DEST:  output directory path
NUM NODES:  the number of nodes
NUM EDGES:  the number of edges
NUM TASKS:  degree of parallelism (the nuber of tasks to run)
```

ex: `sh teg_rand_spark.sh 100000 2000000 240 teg_rand_graph`

The syntax of `teg_rand_hadoop.sh` is:

```
sh teg_rand_hadoop.sh [-r NUM REDUCE TASKS] NUM NODES NUM EDGES NUM TASKS
DEST

DEST:  output directory path
NUM NODES:  the number of nodes
NUM EDGES:  the number of edges
NUM TASKS:  degree of parallelism (the nuber of tasks to run)
NUM REDUCE TASKS:  the number of reduce tasks (default:  1)
```

ex: `sh teg_rand_hadoop.sh -r 80 100000 2000000 240 teg_rand_graph`

Each line of the output contains an edge in the format of

`source node id` TAB `destination node id`

Working examples of running TeG-Random on hadoop and spark are in `teg_rand_spark_demo.sh` and `teg_rand_hadoop_demo.sh`, respectively.

4 Rebuilding Source Codes

PegasusN distribution includes the source code. You can modify and rebuild the code. This section guide you how to rebuild the code.

4.1 List of source codes

Here is the list of source codes and short descriptions.

Source code table will be placed here

4.2 Building the codes