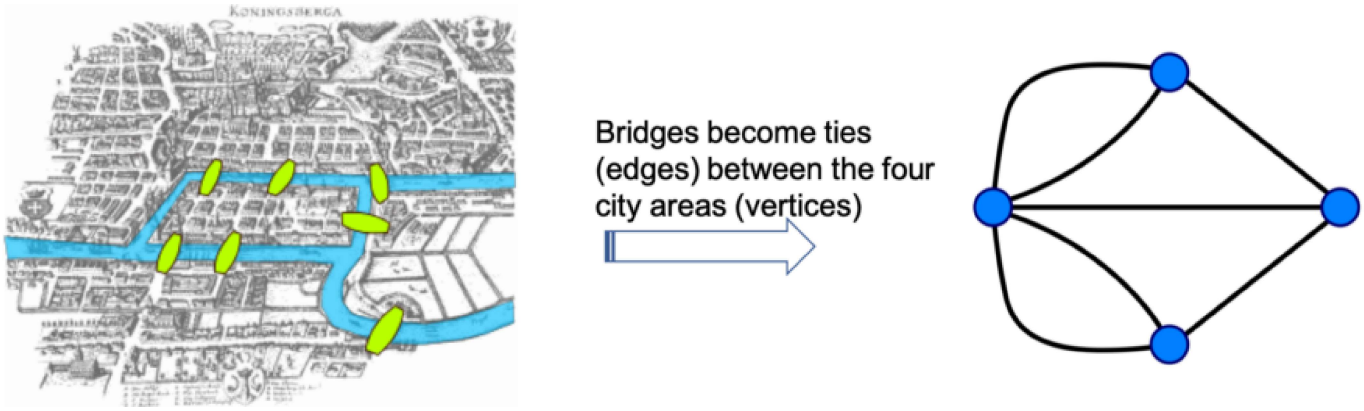
RAPIDS cuGraph

Brad Rees Following
Mar 28, 2019 · 6 min read

The Data Scientist has a collection of techniques within their proverbial toolbox. Data engineering, statistical analysis, and machine learning are among the most commonly known. However, there are numerous cases where the focus of the analysis is on the relationship between data elements. In those cases, the data is best represented as a graph. Graph analysis, also called network analysis, is a collection of algorithms for answering questions posed against graph data. Graph analysis is not new.

The first graph problem was posed by Euler in 1736, *the Seven Bridges of Konigsberg*, and laid the foundation for the mathematical field of graph theory. The application of graph analysis covers a wide variety of fields, including marketing, biology, physics, computer science, sociology, and cyber to name a few. RAPIDS version 0.6 includes the first official release of cuGraph.



https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

RAPIDS cuGraph is a library of graph algorithms that seamlessly integrates into the RAPIDS data science ecosystem and allows the data scientist to easily call graph algorithms using data stored in a GPU DataFrame. For the initial release, graph data needs to be in Coordinate Format (COO), also known as an edge list, and is represented by two columns, Source and Destination. Consider the following example of NetFlow data and looking at just the first five columns. The "id.orig_h" and "id.resp_h" columns identify an event between two IP addresses. Those two columns represent an edge list that can be used for graph processing.

ts	uid	id.orig_h	id.orig_p	id.resp_h
1331901000	CCUIP21wTjqkj8ZqX5	192.168.202.79	50463	192.168.229.251
1331901000	Csssjd3tX0yOTPDpng	192.168.202.79	46117	192.168.229.254
1331901000	CHEt7z3AzG4gyCNgci	192.168.202.79	50465	192.168.229.251
1331901000	CKnDAp2ohlvN6rpiXl	192.168.202.79	50467	192.168.229.251
1331901000	CGUBcoXKxBE8gTNl	192.168.202.79	46119	192.168.229.254
1331901000	CRksHc2i8qahpGOP19	192.168.202.79	46121	192.168.229.254
1331901000	C56nlH3SloWOj44ABi	192.168.202.79	46123	192.168.229.254

Cyber NetFlow data example

Once the two edge columns are identified the data scientist can then leverage any of the cuGraph analytics. To run the PageRank algorithm the user would perform something like:

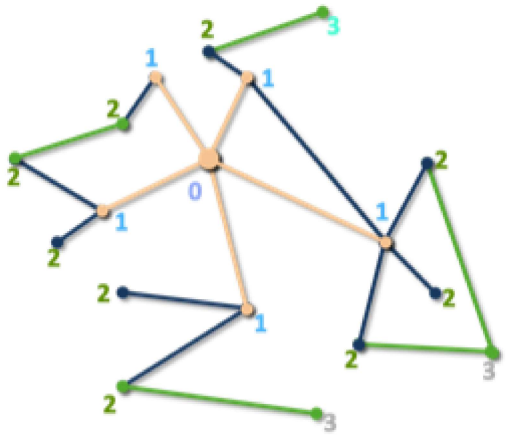
```
G = cugraph.Graph()
G.add_edge_list(gdf["src"], gdf["dst"])
df = cugraph.pagerank(G)
```

One of the design goals of cuGraph is to have an API familiar to the data scientist used to performing graph analytics. Therefore, the data scientist with experience with NetworkX will recognize the process of creating a graph object and then running an analytic against that graph object. A more in-depth example is given at the bottom of this blog.

RAPIDS cuGraph 0.6 release

RAPIDS version 0.6 represents the first official release of cuGraph and the first step towards a complete graph analysis package. This initial release focuses on providing a foundation and includes several algorithms optimized for single-GPU analytics.

- *Jaccard Similarity* — a measure of neighborhood similarity between connected vertices. Within recommendations systems, this is very useful for finding customers with similar behavior.
- *Weighted Jaccard* — this is similar to Jaccard except that the algorithm sums the vertex weights.
- *Page Rank* — this is a measure of relative importance, most famously used in search engines, however it has applications in social network analysis, recommendation systems, and for novel uses in natural science when studying the relationship between proteins and in ecological networks.
- *Single Source Shortest Path (SSSP)* — is used to identify the shortest path between a pair of vertices. Within a road network it can be used to find the fastest path from A to B. Moreover, SSSP can be used for optimizing a wide range of logistics problems.



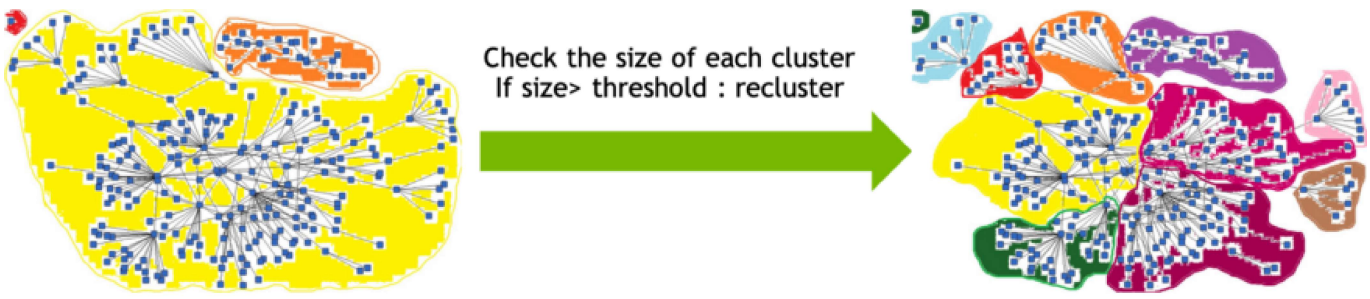
BFS Traversal

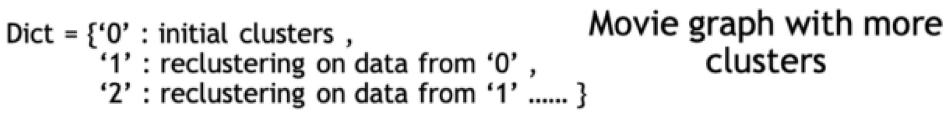
- *Breadth-First Search (BFS)* — this is a classic search algorithm that iteratively explores the graph. Starting at a seed point, the algorithm steps out one hops per iteration. As shown in the image to the left.

- *Spectral Clustering* — graph clustering consists of grouping vertices based on their characteristics such that there is high intra-cluster similarity and low inter-cluster similarity. There are many ways of determining these groups. The spectral clustering scheme constructs a matrix, solves an associated eigenvalue problem, and extracts splitting information from the calculated eigenvectors. Both a Modularity Maximization and Min-Cut version are included.

Spectral Clustering Min-Cut vs Ground Truth

- *Louvain Clustering* — is another graph clustering technique. Louvain uses Modularity as the metric for iteratively combining vertices into clusters. Louvain starts with each vertex in its own cluster and iteratively mergers clusters based on modularity.

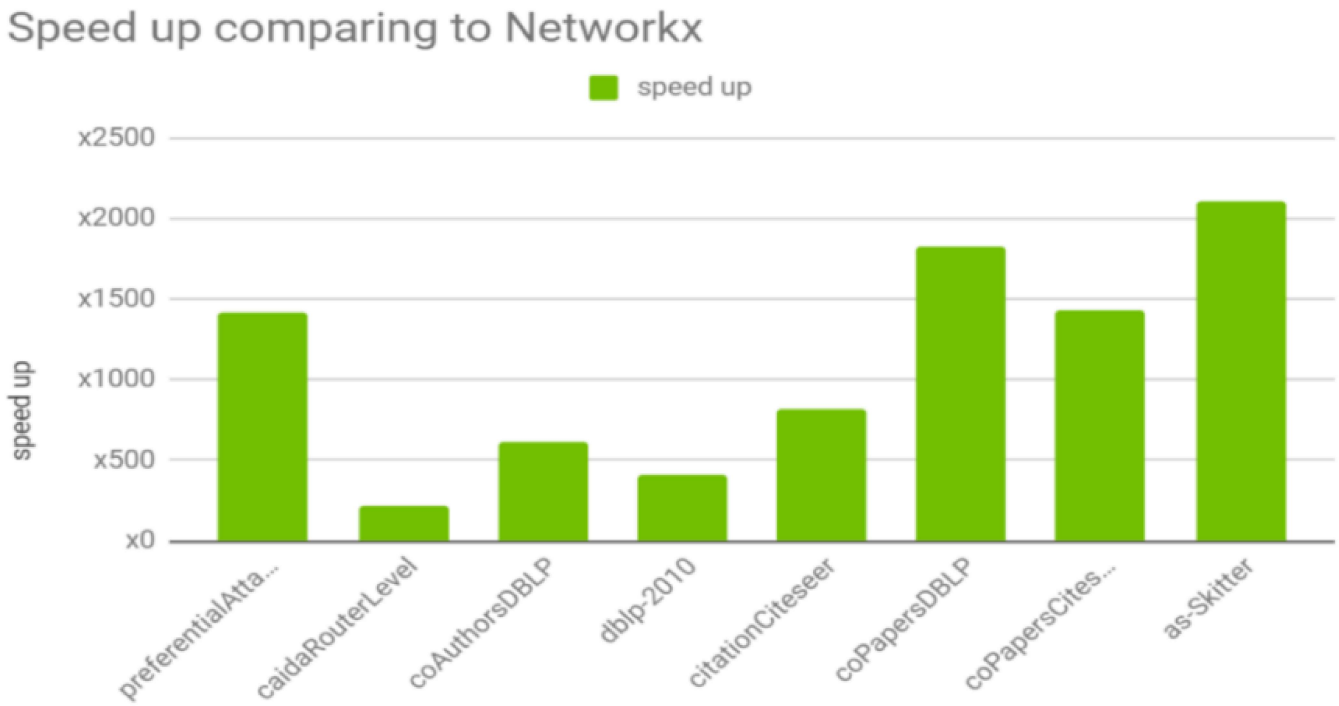




Louvain

Performance

The above listed algorithms are designed for execution on a single-GPU with data sets around 500 million edges or less. When compared against a single-node NetworkX analytic in Python, the data scientist can expect performance improvement of 50–500x on average.



Louvain single-GPU performance compared to NetworkX

As a means of measuring performance, the PageRank websearch benchmark from the HiBench Suite was used. The HiBench website describe the suite as "… *a big data benchmark suite that helps evaluate different big data frameworks in terms of speed, throughput and system resource utilizations*". One feature of HiBench is that it measures the end-to-end performance that a user will experience. That includes, data loading, data prep, and running of an analytic. The PageRank benchmark consist of processing a graph based on webpages (vertices) and the connections between the webpages (edges).

			V100	RTX8000				
HiBench Test Size	Vertices	Edges	cuDF + cuGraph	cuDF + cuGraph	Pandas + cuGraph	NetworkX	Spark 10-nodes	Just PageRank
small	5,000	197,610	0.661	0.822	0.847	1.539		0.23
large	500,000	19,760,120	1.194	1.512	3.762	177.838	109	0.27
huge	5,000,000	197,623,164	7.624	7.851	31.819	2,359.979	213	0.69
10M	10,000,000	395,220,000		15.550				1.2241
13M	13,000,000	513,786,000		19.900				1.5605
gigantic	30,000,000	1,185,769,151			202.610		1,420	3.3709

HiBench Websearch PageRank Results

The biggest challenge faced when executing the benchmark was with the data readers. The CSV reader within cuDF is significantly faster than the one found in Pandas, however we encountered scalability issues with large datasets. To ensure that large data sets were processed, the CSV reader from cuDF was used along with the CSV reader found in Pandas. Additionally, the benchamrk was run on a 10-node Spark cluster and on a high power workstation running NetworkX . Since NetworkX is the most popular graph framework used by data scientists, those results will be used as the baseline for performance evaluation.

RAPIDS cuDF + cuGraph is 300x faster than NetworkX for the huge dataset. If data loading times are removed and just the PageRank performance numbers compared, then cuGraph is 3400x faster than NetworkX. As dataset sizes increase, data reading start to become the limiting factor with data reading contributing to over 92% of the runtime.

Finding external comparable benchmarks for Spark is non-trival since the number of nodes impacts performance. A benchmark result from Mellanox using 6 dual-Xeon nodes and RDMA run the gigantic dataset in 1,087 seconds . Lu Liu reported gigantic results of 9,953 seconds for Spark and 4,078 seconds for Hadoop on a four node cluster. Lastly, Hameeza Ahmed et al, measured just the PageRank portion on 4 nodes against the gigantic dataset with a runtime of 346.85 seconds. In comparison cuGraph PageRank on the gigantic dataset took 3.7 seconds, a speedup of 103x.

Using cuGraph

The cuGraph GitHub repository has detailed instructions on how to access the cuGraph library. See https://github.com/rapidsai/cugraph

Data Science Graph Analytics Rapids Open Source Gpu

Additionally, a collection of sample cuGraph notebooks can be found within the RAPIDS notebook repository under cugraph: https://github.com/rapidsai/notebooks/tree/branch-0.6/cugraph.

The following example runs through how to execute Weighted Jaccard on a graph where the weights are the PageRank scores.

Find the most similar pair of vertices - adjust the vertex ID by adding 1 to match illustration

In [10]:
```
bestEdge = 0
for i in range(len(df)):
    if df['jaccard_coeff'][i] > df['jaccard_coeff'][bestEdge]:
        bestEdge = i

print("Vertices " + str(df['source'][bestEdge] +1) +
      " and " + str(df['destination'][bestEdge] + 1) +
      " are most similar with score: " + str(df['jaccard_coeff'][be
stEdge]))
```

```
Vertices 1 and 2 are most similar with score: 1.0
```

In [ ]:
```
```


What comes next

RAPIDS is a set of open source libraries for GPU accelerating **data preparation, machine learning,** and now **graph analytics.** The roadmap is for additional analytic and constant improvement in performance, with the goal being that every cuGraph algorithms supports multi-GPU. The cuGraph team is eager to hear feedback and impressions of this initial release. A series of blogs on graph analytic is planned, so look for more blogs on cuGraph to come.