# GPU Accelerated Property Graphs in Python

Rick Ratzel – NVIDIA Corporation – July 2022

## What is a Property Graph?

A traditional graph models the relationships between entities in data using vertices and edges, where vertices represent entities, and edges define the relationships. Edges can optionally have a direction and a weight value, but no other information is typically present. However, a property graph can represent data containing multiple, heterogeneous attributes on both vertices and edges.

Tabular information contained in one or more tables - often consisting of heterogeneous data types - can be conveyed with a single property graph, making property graphs useful for modeling real-world data that has complex relationships or interdependencies.

| source | destination | weight |
|---|---|---|
| 33 | 0 | 3.14 |
| 33 | 18 | 1.0 |
| 18 | 0 | 31.14 |
| 0 | 2 | 4.0 |
| 0 | 2 | 11.0 |
| 18 | 2 | 13.14 |

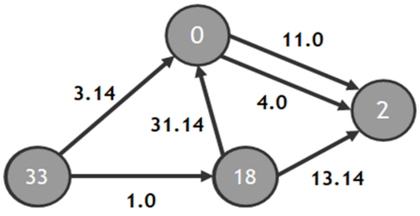


*Figure 1 - A traditional graph with directed and weighted edges*

vertex properties

customers

| Cust_ID | Zip | Card_num |
|---|---|---|
| 18 | 78757 | 23451 |
| 33 | 78750 | 12345 |

stores

| Store_num | Zip | Employees |
|---|---|---|
| 0 | 78750 | 65 |
| 2 | 78757 | 51 |

edge properties

purchases

| Cust_ID | Store_num | Date | Amount |
|---|---|---|---|
| 33 | 0 | 1/1/21 | 3.14 |
| 18 | 2 | 1/3/21 | 13.14 |
| 18 | 0 | 1/4/21 | 31.14 |

referrals

| Cust_ID_1 | Cust_ID_2 | Date | Text |
|---|---|---|---|
| 33 | 18 | 1/3/21 | "store 0 is great!" |

merch xfers

| From | To | Date | SKU | Amount |
|---|---|---|---|---|
| 0 | 2 | 1/7/21 | 123874 | 4 |

employee xfers

| From | To | Date | Employee |
|---|---|---|---|
| 0 | 2 | 1/12/21 | 11 |

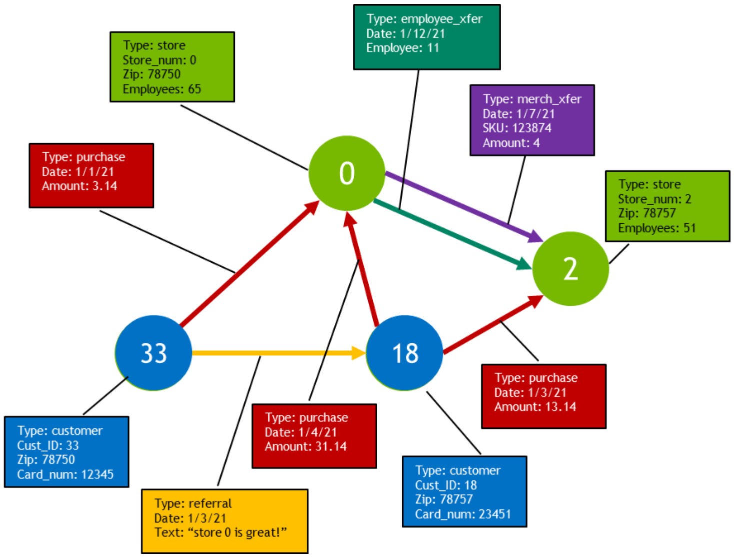


*Figure 2- A property graph containing multiple, heterogeneous attributes on both vertices and edges.*

## Where Are Property Graphs Useful?

### Graph Neural Networks (GNNs)

- Property graphs are obvious choices for graphs that require node and edge features used by GNN frameworks.

### Graph Databases

- Databases which support complex queries involving relationships between entities and their properties are often implemented using property graphs and a graph database query language such as Cypher or GQL.

### Graph Visualization

- Standard graph visualization use cases (maps, social networks, circuit layout, etc.) that require additional information about each element for rendering (color, size, shape, etc.) are easily implemented using property graphs.

## GPU Acceleration

GPU accelerated graph analytics are often implemented using sparse matrix operations, which allow for edge weights (or edge IDs if the algorithm does not require weight values) but not arbitrary properties for vertices and edges. For these implementations, a separate subgraph extraction step is required to perform the analytic. The subgraph extraction step is most efficient if the sparse matrix on the GPU can be created directly from data already present in GPU memory.

- Subgraphs extracted from property graphs can use user-selected edge properties already present on the GPU for edge weights.
- Graph analytic results computed on the GPU can be added back to the property graph as vertex or edge properties efficiently for future queries or computations.

## GPU Accelerated Property Graphs in Python

Python is an excellent choice for working with heterogeneous data that can be represented with a property graph, and (currently experimental) support for property graphs using GPUs has been added to the cuGraph GPU-accelerated python graph analytics library.

Below are two examples demonstrating the cuGraph property graph API.

### Example: use a Property Graph to load various datasets as edges and vertices with attributes, use the Property Graph API to extract different graphs based on attributes to run analysis.

```
pG = cugraph.experimental.PropertyGraph()

pG.add_vertex_data(customers_df,
                   type_name="customers",
                   vertex_col_name="Cust_ID")
...
pG.add_edge_data(purchases_df,
                 type_name="purchases",
                 vertex_col_names=("Cust_ID", "Store_num"))
...
selection = pG.select_vertices(f"{pG.type_col_name}=='stores'")
selection += pG.select_edges(f"{pG.type_col_name}=='merch_xfers'")
G = pG.extract_subgraph(selection=selection, edge_weight_property="Amount")

print(G.view_edge_list())
```

```
(rapids) root@17e55ac97b56:/Projects/cugraph/python/cugraph# python pgdemo4.py
   weights  src  dst
0      4.0    0    2
```

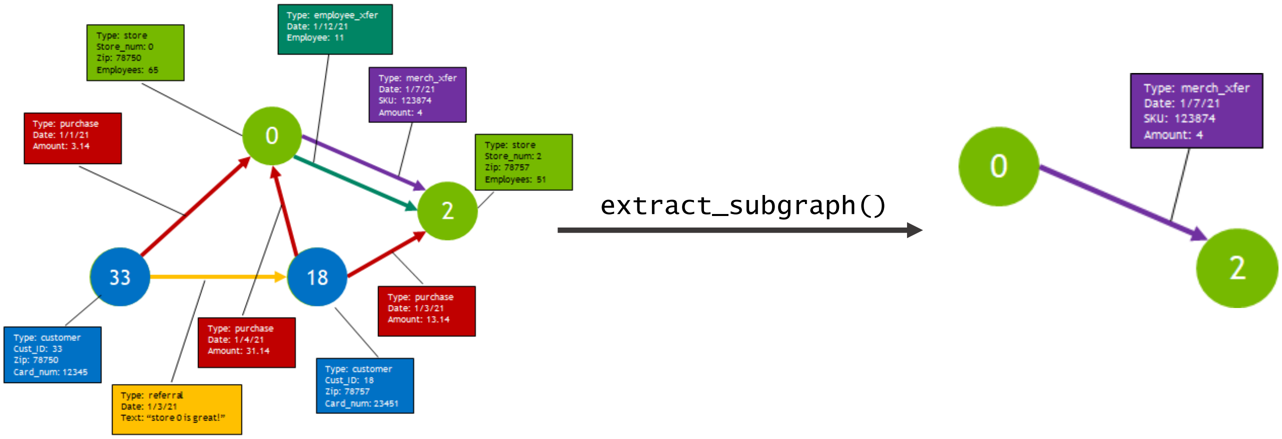
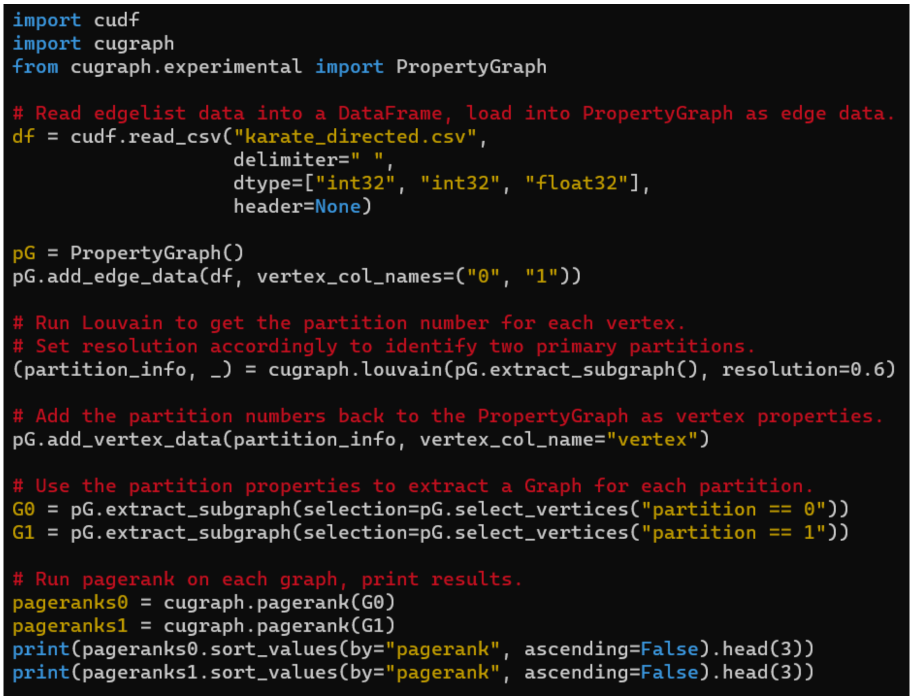


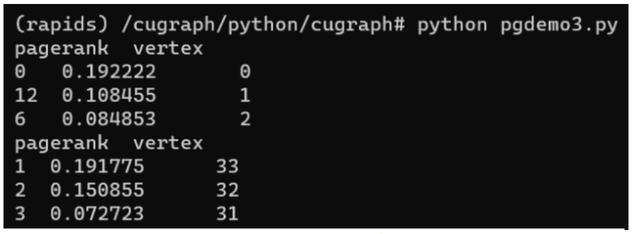
*Figure 3- cuGraph property graph API example that extracts a single vertex pair and edge as a subgraph.*

In the example above, a subgraph is extracted based on specific edges and vertices selected by the user. Because the property graph is GPU-based, only GPU memory is read and written to, and the resulting subgraph can be used directly by other cuGraph GPU-accelerated APIs without an expensive host-to-GPU data copy step.

Example: use a Property Graph to load the Zachary Karate Club dataset, use Louvain to find the two primary partitions, use Pagerank to find the top 3 influential vertices in each partition.



```
import cudf
import cugraph
from cugraph.experimental import PropertyGraph

# Read edgelist data into a DataFrame, load into PropertyGraph as edge data.
df = cudf.read_csv("karate_directed.csv",
                   delimiter=" ",
                   dtype=["int32", "int32", "float32"],
                   header=None)

pG = PropertyGraph()
pG.add_edge_data(df, vertex_col_names=("0", "1"))

# Run Louvain to get the partition number for each vertex.
# Set resolution accordingly to identify two primary partitions.
(partition_info, _) = cugraph.louvain(pG.extract_subgraph(), resolution=0.6)

# Add the partition numbers back to the PropertyGraph as vertex properties.
pG.add_vertex_data(partition_info, vertex_col_name="vertex")

# Use the partition properties to extract a Graph for each partition.
G0 = pG.extract_subgraph(selection=pG.select_vertices("partition == 0"))
G1 = pG.extract_subgraph(selection=pG.select_vertices("partition == 1"))

# Run pagerank on each graph, print results.
pageranks0 = cugraph.pagerank(G0)
pageranks1 = cugraph.pagerank(G1)
print(pageranks0.sort_values(by="pagerank", ascending=False).head(3))
print(pageranks1.sort_values(by="pagerank", ascending=False).head(3))
```



```
(rapids) /cugraph/python/cugraph# python pgdemo3.py
pagerank  vertex
0   0.192222       0
12  0.108455       1
6   0.084853       2
pagerank  vertex
1  0.191775      33
2  0.150855      32
3  0.072723      31
```

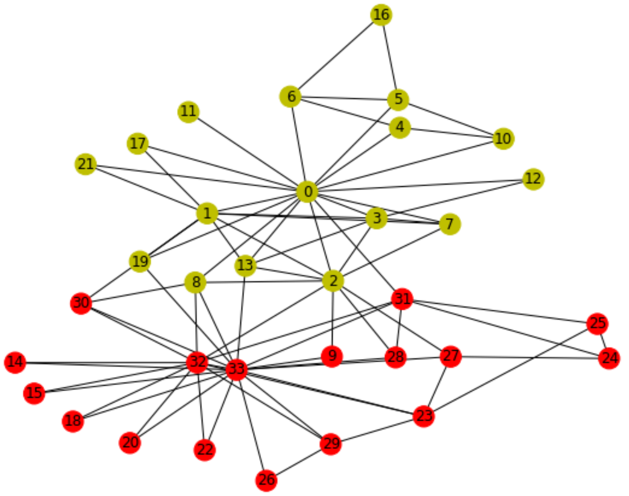


*Figure 4- cuGraph property graph example where algorithm results are used to dynamically add properties to the graph which can be used in future analysis.*

In the example above, graph data with no properties is loaded into a property graph instance where a GPU-accelerated Louvain algorithm is used to find the two primary partitions. The partition information is added back to the property graph instance as vertex properties, which are then used to extract multiple subgraphs containing vertices of only specific partitions. Here once again, since the property graph is GPU-based, the algorithms can run on extracted subgraphs without incurring host-to-device or device-to-host memory copy overhead.