hipGRAPH

Release 1.0.0b1

Advanced Micro Devices, Inc.

Jan 14, 2026

INSTALL

> **Note**
>
> hipGRAPH is in an early access state. Running production workloads is not recommended. The early access version of the hipGRAPH Python layer is untested and the Python API is unsupported.

hipGRAPH is a graph marshalling library that acts as a wrapper between your application and a worker graph library, such as rocGRAPH.

The hipGRAPH library contains a collection of graph algorithms, enabling you to build, analyze, and manipulate complex graphs or networks. hipGRAPH is derived from the cuGraph library that forms part of the NVIDIA RAPIDS open source project. The hipGRAPH library described here should not be confused with HIP graphs in the HIP runtime API.

The hipGRAPH code is open and hosted at https://github.com/ROCm-DS/hipGraph.

The hipGRAPH documentation is structured as follows:

Installation

- *hipGRAPH installation instructions*

API reference

- *C++ API reference*
- *Python API reference*

To contribute to the documentation refer to Contributing to ROCm-DS.

You can find licensing information on the Licensing page.

CHAPTER

ONE

HIPGRAPH INSTALLATION INSTRUCTIONS

You can install hipGRAPH using the following instructions. There are some prerequisites that should be installed prior to installing the hipGRAPH library, as described in the following steps.

1.1 Prerequisites

hipGRAPH requires a ROCm-enabled platform as an implementation backend. This documentation assumes that you have a system with a compatible AMD GPU. hipGRAPH has been tested with the following system requirements:

- Ubuntu version: 22.04
- ROCm version: 6.4.0
- AMD GPUs: AMD Instinct MI210 and AMD Radeon RX7900 GRE

1.2 Building hipGRAPH from source for the ROCm backend

There currently are no prebuilt packages, and hipGRAPH must be built and installed from source files.

The following compile-time dependencies must be met:

- AMD ROCm 6.4.0
- rocGRAPH
- git
- CMake 3.5 or later
- GoogleTest (optional, for the test suite)
- Python 3.10 (optional for Python modules)

> **Note**
>
> hipGRAPH does not require hipcc and is tested against other compilers like g++. It does require the libraries in a ROCm installation.

1.2.1 Download hipGRAPH

The hipGRAPH source code is available at the hipGRAPH GitHub page. Download the source code using the following commands:

```
$ git clone https://github.com/ROCm-DS/hipGRAPH.git
$ cd hipGRAPH
```

1.2.2 Using `install.sh` to build and install hipGRAPH

It is recommended that you use the `install.sh` script to build and install different packages for the hipGRAPH library, including dependencies needed by hipGRAPH, and clients that use hipGRAPH such as unit tests.

The following table lists common uses of `install.sh` from the hipGRAPH source folder to build and install hipGRAPH and optionally its dependencies and clients.

Command	Description
`./install.sh -h`	Print help information.
`./install.sh`	Build the hipGRAPH library in your local directory. It is assumed that the required dependencies have been previously installed.
`./install.sh -d`	Build the hipGRAPH library and its dependencies in your local directory. The `-d` flag only needs to be used once. For subsequent invocations of install.sh it is not necessary to rebuild the dependencies.
`./install.sh -c`	Build the library and client in your local directory. It is assumed dependencies are available.
`./install.sh -dc`	Build the library, dependencies, and client in your local directory. The `-d` flag only needs to be used once. For subsequent invocations of install.sh it is not necessary to rebuild the dependencies.
`./install.sh -i`	Build the library, then build and install the hipGRAPH package in `/opt/rocm/hipgraph`. You will be prompted for sudo access. This will install for all users.
`./install.sh -idc`	Build the library, dependencies, and client, then build and install the hipGRAPH package in `/opt/rocm/hipgraph`. You will be prompted for sudo access. This will install for all users.
`./install.sh -ic`	Build the library and client, then build and install the hipGRAPH package in `opt/rocm/hipgraph`. You will be prompted for sudo access. This will install for all users.

1.2.3 Building and installing hipGRAPH with individual commands

CMake 3.5 or later is required to build hipGRAPH. GoogleTest is required to build hipGRAPH clients.

Despite its name, *CMAKE_INSTALL_PREFIX* primary usage is pointing CMake towards the package configurations in *lib/cmake* under that prefix. The Python wheels bundle the libraries after running *auditwheel*.

You can build and install hipGRAPH along with its dependencies and clients using the following commands:

```
# Install GoogleTest
$ mkdir -p build/release/deps ; cd build/release/deps
$ cmake ../../../deps
$ make -j$(nproc) install

# Change to build directory
$ cd ..

# Default install path is /opt/rocm, use -DCMAKE_INSTALL_PREFIX=<path> to adjust it
$ cmake ../.. -DBUILD_CLIENTS_TESTS=ON

# Compile hipGRAPH library
$ make -j$(nproc)

# Install hipGRAPH to /opt/rocm
$ make install
```

Alternatively, you can build just hipGRAPH without dependencies or clients using the following commands. This

assumes that the dependencies or clients have been previously installed.

```
# Create and change to build directory
$ mkdir -p build/release ; cd build/release

# Default install path is /opt/rocm, use -DCMAKE_INSTALL_PREFIX=<path> to adjust it
$ cmake ../..

# Compile hipGRAPH library
$ make -j$(nproc)

# Install hipGRAPH into build/release/DESTDIR
$ make install DESTDIR=$(pwd)/build/release/DESTDIR
# Optionally, install hipGRAPH to a system-wide /opt/rocm
$ make install
```

1.2.4 Building the Python modules

The Python modules require much more detailed instructions, which are work in progress. See `clients/python`.

CHAPTER

TWO

HIPGRAPH C++ API DOCUMENTATION

2.1 Centrality Functions

2.1.1 PageRank

hipgraph_error_code_t **hipgraph_pagerank**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *precomputed_vertex_out_weight_vertices, const hipgraph_type_erased_device_array_view_t *precomputed_vertex_out_weight_sums, const hipgraph_type_erased_device_array_view_t *initial_guess_vertices, const hipgraph_type_erased_device_array_view_t *initial_guess_values, double alpha, double epsilon, size_t max_iterations, hipgraph_bool_t do_expensive_check, hipgraph_centrality_result_t **result, hipgraph_error_t **error,
)

Compute pagerank.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **precomputed_vertex_out_weight_vertices** – **[in]** Optionally send in precomputed sum of vertex out weights (a performance optimization). This defines the vertices. Set to NULL if no value is passed.
- **precomputed_vertex_out_weight_sums** – **[in]** Optionally send in precomputed sum of vertex out weights (a performance optimization). Set to NULL if no value is passed.
- **initial_guess_vertices** – **[in]** Optionally send in an initial guess of the pagerank values (a performance optimization). This defines the vertices. Set to NULL if no value is passed. If NULL, initial PageRank values are set to 1.0 divided by the number of vertices in the graph.
- **initial_guess_values** – **[in]** Optionally send in an initial guess of the pagerank values (a performance optimization). Set to NULL if no value is passed. If NULL, initial PageRank values are set to 1.0 divided by the number of vertices in the graph.
- **alpha** – **[in]** PageRank damping factor.
- **epsilon** – **[in]** Error tolerance to check convergence. Convergence is assumed if the sum of the differences in PageRank values between two consecutive iterations is less than the number of vertices in the graph multiplied by `epsilon`.
- **max_iterations** – **[in]** Maximum number of PageRank iterations.

- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to pagerank results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

hipgraph_error_code_t **hipgraph_pagerank_allow_nonconvergence**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *precomputed_vertex_out_weight_vertices, const hipgraph_type_erased_device_array_view_t *precomputed_vertex_out_weight_sums, const hipgraph_type_erased_device_array_view_t *initial_guess_vertices, const hipgraph_type_erased_device_array_view_t *initial_guess_values, double alpha, double epsilon, size_t max_iterations, hipgraph_bool_t do_expensive_check, hipgraph_centrality_result_t **result, hipgraph_error_t **error,
)

Compute pagerank.

Deprecated:

This version of pagerank should be dropped in favor of the hipgraph_pagerank_allow_nonconvergence version. Eventually that version will be renamed to this version.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **precomputed_vertex_out_weight_vertices** – **[in]** Optionally send in precomputed sum of vertex out weights (a performance optimization). This defines the vertices. Set to NULL if no value is passed.
- **precomputed_vertex_out_weight_sums** – **[in]** Optionally send in precomputed sum of vertex out weights (a performance optimization). Set to NULL if no value is passed.
- **initial_guess_vertices** – **[in]** Optionally send in an initial guess of the pagerank values (a performance optimization). This defines the vertices. Set to NULL if no value is passed. If NULL, initial PageRank values are set to 1.0 divided by the number of vertices in the graph.
- **initial_guess_values** – **[in]** Optionally send in an initial guess of the pagerank values (a performance optimization). Set to NULL if no value is passed. If NULL, initial PageRank values are set to 1.0 divided by the number of vertices in the graph.
- **alpha** – **[in]** PageRank damping factor.
- **epsilon** – **[in]** Error tolerance to check convergence. Convergence is assumed if the sum of the differences in PageRank values between two consecutive iterations is less than the number of vertices in the graph multiplied by `epsilon`.
- **max_iterations** – **[in]** Maximum number of PageRank iterations.
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to pagerank results

- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.1.2 Personalized PageRank

hipgraph_error_code_t **hipgraph_personalized_pagerank**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *precomputed_vertex_out_weight_vertices, const hipgraph_type_erased_device_array_view_t *precomputed_vertex_out_weight_sums, const hipgraph_type_erased_device_array_view_t *initial_guess_vertices, const hipgraph_type_erased_device_array_view_t *initial_guess_values, const hipgraph_type_erased_device_array_view_t *personalization_vertices, const hipgraph_type_erased_device_array_view_t *personalization_values, double alpha, double epsilon, size_t max_iterations, hipgraph_bool_t do_expensive_check, hipgraph_centrality_result_t **result, hipgraph_error_t **error,
)

Compute personalized pagerank.

Deprecated:

This version of personalized pagerank should be dropped in favor of the hipgraph_personalized_pagerank_allow_nonconvergence version. Eventually that version will be renamed to this version.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **precomputed_vertex_out_weight_vertices** – **[in]** Optionally send in precomputed sum of vertex out weights (a performance optimization). This defines the vertices. Set to NULL if no value is passed.
- **precomputed_vertex_out_weight_sums** – **[in]** Optionally send in precomputed sum of vertex out weights (a performance optimization). Set to NULL if no value is passed.
- **initial_guess_vertices** – **[in]** Optionally send in an initial guess of the pagerank values (a performance optimization). This defines the vertices. Set to NULL if no value is passed. If NULL, initial PageRank values are set to 1.0 divided by the number of vertices in the graph.
- **initial_guess_values** – **[in]** Optionally send in an initial guess of the pagerank values (a performance optimization). Set to NULL if no value is passed. If NULL, initial PageRank values are set to 1.0 divided by the number of vertices in the graph.
- **personalization_vertices** – **[in]** Pointer to an array storing personalization vertex identifiers (compute personalized PageRank).
- **personalization_values** – **[in]** Pointer to an array storing personalization values for the vertices in the personalization set.
- **alpha** – **[in]** PageRank damping factor.
- **epsilon** – **[in]** Error tolerance to check convergence. Convergence is assumed if the sum of the differences in PageRank values between two consecutive iterations is less than the number of vertices in the graph multiplied by `epsilon`.

- **max_iterations** – **[in]** Maximum number of PageRank iterations.
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to pagerank results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

hipgraph_error_code_t **hipgraph_personalized_pagerank_allow_nonconvergence**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *precomputed_vertex_out_weight_vertices, const hipgraph_type_erased_device_array_view_t *precomputed_vertex_out_weight_sums, const hipgraph_type_erased_device_array_view_t *initial_guess_vertices, const hipgraph_type_erased_device_array_view_t *initial_guess_values, const hipgraph_type_erased_device_array_view_t *personalization_vertices, const hipgraph_type_erased_device_array_view_t *personalization_values, double alpha, double epsilon, size_t max_iterations, hipgraph_bool_t do_expensive_check, hipgraph_centrality_result_t **result, hipgraph_error_t **error,
)

Compute personalized pagerank.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **precomputed_vertex_out_weight_vertices** – **[in]** Optionally send in precomputed sum of vertex out weights (a performance optimization). This defines the vertices. Set to NULL if no value is passed.
- **precomputed_vertex_out_weight_sums** – **[in]** Optionally send in precomputed sum of vertex out weights (a performance optimization). Set to NULL if no value is passed.
- **initial_guess_vertices** – **[in]** Optionally send in an initial guess of the pagerank values (a performance optimization). This defines the vertices. Set to NULL if no value is passed. If NULL, initial PageRank values are set to 1.0 divided by the number of vertices in the graph.
- **initial_guess_values** – **[in]** Optionally send in an initial guess of the pagerank values (a performance optimization). Set to NULL if no value is passed. If NULL, initial PageRank values are set to 1.0 divided by the number of vertices in the graph.
- **personalization_vertices** – **[in]** Pointer to an array storing personalization vertex identifiers (compute personalized PageRank).
- **personalization_values** – **[in]** Pointer to an array storing personalization values for the vertices in the personalization set.
- **alpha** – **[in]** PageRank damping factor.
- **epsilon** – **[in]** Error tolerance to check convergence. Convergence is assumed if the sum of the differences in PageRank values between two consecutive iterations is less than the number of vertices in the graph multiplied by `epsilon`.
- **max_iterations** – **[in]** Maximum number of PageRank iterations.

- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to pagerank results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.1.3 Eigenvector Centrality

hipgraph_error_code_t **hipgraph_eigenvector_centrality**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, double epsilon, size_t max_iterations, hipgraph_bool_t do_expensive_check, hipgraph_centrality_result_t **result, hipgraph_error_t **error,
)

Compute eigenvector centrality.

Computed using the power method.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **epsilon** – **[in]** Error tolerance to check convergence. Convergence is measured comparing the L1 norm until it is less than epsilon
- **max_iterations** – **[in]** Maximum number of power iterations, will not exceed this number of iterations even if we haven't converged
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to eigenvector centrality results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.1.4 Katz Centrality

hipgraph_error_code_t **hipgraph_katz_centrality**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *betas, double alpha, double beta, double epsilon, size_t max_iterations, hipgraph_bool_t do_expensive_check, hipgraph_centrality_result_t **result, hipgraph_error_t **error,
)

Compute katz centrality.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph

- **betas** – **[in]** Optionally send in a device array holding values to be added to each vertex's new Katz Centrality score in every iteration. If set to NULL then `beta` is used for all vertices.
- **alpha** – **[in]** Katz centrality attenuation factor. This should be smaller than the inverse of the maximum eigenvalue of this graph
- **beta** – **[in]** Constant value to be added to each vertex's new Katz Centrality score in every iteration. Relevant only when `betas` is NULL
- **epsilon** – **[in]** Error tolerance to check convergence. Convergence is assumed if the sum of the differences in Katz Centrality values between two consecutive iterations is less than the number of vertices in the graph multiplied by `epsilon`. (L1-norm)
- **max_iterations** – **[in]** Maximum number of Katz Centrality iterations.
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to katz centrality results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.1.5 Betweenness Centrality

hipgraph_error_code_t **hipgraph_betweenness_centrality**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *vertex_list, hipgraph_bool_t normalized, hipgraph_bool_t include_endpoints, hipgraph_bool_t do_expensive_check, hipgraph_centrality_result_t **result, hipgraph_error_t **error,
)

Compute betweenness centrality.

Betweenness can be computed exactly by specifying vertex_list as NULL. This will compute betweenness centrality by doing a traversal from every source vertex.

Approximate betweenness can be computed specifying a list of vertices that should be used as seeds for the traversals. Note that the function hipgraph_select_random_vertices can be used to create a list of seeds.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **vertex_list** – **[in]** Optionally specify a device array containing a list of vertices to use as seeds for betweenness centrality approximation
- **normalized** – **[in]** Normalize
- **include_endpoints** – **[in]** The traditional formulation of betweenness centrality does not include endpoints when considering a vertex to be on a shortest path. Setting this to true will consider the endpoints of a path to be part of the path.
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to betweenness centrality results

- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.1.6 Edge Betweenness Centrality

hipgraph_error_code_t **hipgraph_edge_betweenness_centrality**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *vertex_list, hipgraph_bool_t normalized, hipgraph_bool_t do_expensive_check, hipgraph_edge_centrality_result_t **result, hipgraph_error_t **error,
)

Compute edge betweenness centrality.

Edge betweenness can be computed exactly by specifying vertex_list as NULL. This will compute betweenness centrality by doing a traversal from every vertex and counting the frequency that a edge appears on a shortest path.

Approximate betweenness can be computed specifying a list of vertices that should be used as seeds for the traversals. Note that the function hipgraph_select_random_vertices can be used to create a list of seeds.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **vertex_list** – **[in]** Optionally specify a device array containing a list of vertices to use as seeds for betweenness centrality approximation
- **normalized** – **[in]** Normalize
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to edge betweenness centrality results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.1.7 HITS Centrality

hipgraph_error_code_t **hipgraph_hits**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, double epsilon, size_t max_iterations, const hipgraph_type_erased_device_array_view_t *initial_hubs_guess_vertices, const hipgraph_type_erased_device_array_view_t *initial_hubs_guess_values, hipgraph_bool_t normalize, hipgraph_bool_t do_expensive_check, hipgraph_hits_result_t **result, hipgraph_error_t **error,
)

Compute hits.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph

- **epsilon** – **[in]** Error tolerance to check convergence. Convergence is assumed if the sum of the differences in Hits values between two consecutive iterations is less than the number of vertices in the graph multiplied by `epsilon`.
- **max_iterations** – **[in]** Maximum number of Hits iterations.
- **initial_hubs_guess_vertices** – **[in]** Pointer to optional type erased device array containing the vertex ids for an initial hubs guess. If set to NULL there is no initial guess.
- **initial_hubs_guess_values** – **[in]** Pointer to optional type erased device array containing the values for an initial hubs guess. If set to NULL there is no initial guess. Note that both `initial_hubs_guess_vertices` and `initial_hubs_guess_values` have to be specified (or they both have to be NULL). Otherwise this will be treated as an error.
- **normalize** – **[in]** A flag to normalize the results (if set to `true`)
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to hits results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.1.8 Centrality Support Functions

hipgraph_type_erased_device_array_view_t *hipgraph_centrality_result_get_vertices(
hipgraph_centrality_result_t *result,
)

Get the vertex ids from the centrality result.

Parameters
result – **[in]** The result from a centrality algorithm

Returns
type erased array of vertex ids

hipgraph_type_erased_device_array_view_t *hipgraph_centrality_result_get_values(
hipgraph_centrality_result_t *result,
)

Get the centrality values from a centrality algorithm result.

Parameters
result – **[in]** The result from a centrality algorithm

Returns
type erased array view of centrality values

size_t hipgraph_centrality_result_get_num_iterations(
hipgraph_centrality_result_t *result,
)

Get the number of iterations executed from the algorithm metadata.

Parameters
result – **[in]** The result from a centrality algorithm

Returns
the number of iterations

hipgraph_bool_t **hipgraph_centrality_result_converged**(
    hipgraph_centrality_result_t *result,
)

Returns true if the centrality algorithm converged.

Parameters
result – **[in]** The result from a centrality algorithm

Returns
True if the centrality algorithm converged, false otherwise

void **hipgraph_centrality_result_free**(
    hipgraph_centrality_result_t *result,
)

Free centrality result.

Parameters
result – **[in]** The result from a centrality algorithm

hipgraph_type_erased_device_array_view_t ***hipgraph_edge_centrality_result_get_src_vertices**(
    hipgraph_edge_centrality_result_t *result,
)

Get the src vertex ids from an edge centrality result.

Parameters
result – **[in]** The result from an edge centrality algorithm

Returns
type erased array of src vertex ids

hipgraph_type_erased_device_array_view_t ***hipgraph_edge_centrality_result_get_dst_vertices**(
    hipgraph_edge_centrality_result_t *result,
)

Get the dst vertex ids from an edge centrality result.

Parameters
result – **[in]** The result from an edge centrality algorithm

Returns
type erased array of dst vertex ids

hipgraph_type_erased_device_array_view_t ***hipgraph_edge_centrality_result_get_edge_ids**(
    hipgraph_edge_centrality_result_t *result,
)

Get the edge ids from an edge centrality result.

Parameters
result – **[in]** The result from an edge centrality algorithm

Returns
type erased array of edge ids

hipgraph_type_erased_device_array_view_t ***hipgraph_edge_centrality_result_get_values**(
    hipgraph_edge_centrality_result_t *result,
)

Get the centrality values from an edge centrality algorithm result.

Parameters
result – **[in]** The result from an edge centrality algorithm

Returns
type erased array view of centrality values

void **hipgraph_edge_centrality_result_free**(
hipgraph_edge_centrality_result_t *result,
)

Free centrality result.

Parameters
result – **[in]** The result from a centrality algorithm

hipgraph_type_erased_device_array_view_t ***hipgraph_hits_result_get_vertices**(
hipgraph_hits_result_t *result,
)

Get the vertex ids from the hits result.

Parameters
result – **[in]** The result from hits

Returns
type erased array of vertex ids

hipgraph_type_erased_device_array_view_t ***hipgraph_hits_result_get_hubs**(
hipgraph_hits_result_t *result,
)

Get the hubs values from the hits result.

Parameters
result – **[in]** The result from hits

Returns
type erased array of hubs values

hipgraph_type_erased_device_array_view_t ***hipgraph_hits_result_get_authorities**(
hipgraph_hits_result_t *result,
)

Get the authorities values from the hits result.

Parameters
result – **[in]** The result from hits

Returns
type erased array of authorities values

double **hipgraph_hits_result_get_hub_score_differences**(
hipgraph_hits_result_t *result,
)

Get the score differences between the last two iterations.

Parameters
result – **[in]** The result from hits

Returns
score differences

size_t **hipgraph_hits_result_get_number_of_iterations**(
hipgraph_hits_result_t *result,
)

Get the actual number of iterations.

Parameters
result – **[in]** The result from hits

Returns
actual number of iterations

void **hipgraph_hits_result_free**(
hipgraph_hits_result_t *result,
)

Free hits result.

Parameters
result – **[in]** The result from hits

2.2 Community Functions

2.2.1 Triangle Counting

hipgraph_error_code_t **hipgraph_triangle_count**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *start, hipgraph_bool_t do_expensive_check, hipgraph_triangle_count_result_t **result, hipgraph_error_t **error,
)

Triangle Counting.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **start** – **[in]** Device array of vertices we want to count triangles for. If NULL the entire set of vertices in the graph is processed
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to true)
- **result** – **[out]** Output from the triangle_count call
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.2.2 Louvain

hipgraph_error_code_t **hipgraph_louvain**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, size_t max_level, double threshold, double resolution, hipgraph_bool_t do_expensive_check, hipgraph_hierarchical_clustering_result_t **result, hipgraph_error_t **error,
)

Compute Louvain.

Parameters

- **handle** – **[in]** Handle for accessing resources

- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **max_level** – **[in]** Maximum level in hierarchy
- **threshold** – **[in]** Threshold parameter, defines convergence at each level of hierarchy
- **resolution** – **[in]** Resolution parameter (gamma) in modularity formula. This changes the size of the communities. Higher resolutions lead to more smaller communities, lower resolutions lead to fewer larger communities.
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to true)
- **result** – **[out]** Output from the Louvain call
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.2.3 Leiden

hipgraph_error_code_t **hipgraph_leiden**(
const hipgraph_resource_handle_t *handle, hipgraph_rng_state_t *rng_state, hipgraph_graph_t *graph, size_t max_level, double resolution, double theta, hipgraph_bool_t do_expensive_check, hipgraph_hierarchical_clustering_result_t **result, hipgraph_error_t **error,
)

Compute Leiden.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **rng_state** – **[inout]** State of the random number generator, updated with each call
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **max_level** – **[in]** Maximum level in hierarchy
- **resolution** – **[in]** Resolution parameter (gamma) in modularity formula. This changes the size of the communities. Higher resolutions lead to more smaller communities, lower resolutions lead to fewer larger communities.
- **theta** – **[in]** (optional) The value of the parameter to scale modularity gain in Leiden refinement phase. It is used to compute the probability of joining a random leiden community. Called theta in the Leiden algorithm.
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to true)
- **result** – **[out]** Output from the Leiden call
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.2.4 ECG

hipgraph_error_code_t **hipgraph_ecg**(
const hipgraph_resource_handle_t *handle, hipgraph_rng_state_t *rng_state, hipgraph_graph_t *graph, double min_weight, size_t ensemble_size, size_t max_level, double threshold, double resolution, hipgraph_bool_t do_expensive_check, hipgraph_hierarchical_clustering_result_t **result, hipgraph_error_t **error,
)

Compute ECG clustering of the given graph.

ECG runs truncated Louvain on an ensemble of permutations of the input graph, then uses the ensemble partitions to determine weights for the input graph. The final result is found by running full Louvain on the input graph using the determined weights. See https://arxiv.org/abs/1809.05578 for further information.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **rng_state** – **[inout]** State of the random number generator, updated with each call
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **min_weight** – **[in]** Minimum edge weight in final graph
- **ensemble_size** – **[in]** The number of Louvain iterations to run
- **max_level** – **[in]** Maximum level in hierarchy for final Louvain
- **threshold** – **[in]** Threshold parameter, defines convergence at each level of hierarchy for final Louvain
- **resolution** – **[in]** Resolution parameter (gamma) in modularity formula. This changes the size of the communities. Higher resolutions lead to more smaller communities, lower resolutions lead to fewer larger communities.
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to true)
- **result** – **[out]** Output from the Louvain call
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.2.5 Extract Egonet

hipgraph_error_code_t **hipgraph_extract_ego**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *source_vertices, size_t radius, hipgraph_bool_t do_expensive_check, hipgraph_induced_subgraph_result_t **result, hipgraph_error_t **error,
)

Extract ego graphs.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **source_vertices** – **[in]** Device array of vertices we want to extract egonets for.

- **radius** – **[in]** The number of hops to go out from each source vertex
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to true)
- **result** – **[out]** Opaque object containing the extracted subgraph
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.2.6 Balanced Cut

hipgraph_error_code_t **hipgraph_balanced_cut_clustering**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, size_t n_clusters, size_t n_eigenvectors, double evs_tolerance, int evs_max_iterations, double k_means_tolerance, int k_means_max_iterations, hipgraph_bool_t do_expensive_check, hipgraph_clustering_result_t **result, hipgraph_error_t **error,
)

Balanced cut clustering.

NOTE: This currently wraps the legacy balanced cut clustering implementation and is only available in Single GPU implementation.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **n_clusters** – **[in]** The desired number of clusters
- **n_eigenvectors** – **[in]** The number of eigenvectors to use
- **evs_tolerance** – **[in]** The tolerance to use for the eigenvalue solver
- **evs_max_iterations** – **[in]** The maximum number of iterations of the eigenvalue solver
- **k_means_tolerance** – **[in]** The tolerance to use for the k-means solver
- **k_means_max_iterations** – **[in]** The maximum number of iterations of the k-means solver
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to true)
- **result** – **[out]** Opaque object containing the clustering result
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.2.7 Spectral Clustering - Modularity Maximization

hipgraph_error_code_t **hipgraph_spectral_modularity_maximization**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, size_t n_clusters, size_t n_eigenvectors, double evs_tolerance, int evs_max_iterations, double k_means_tolerance, int k_means_max_iterations, hipgraph_bool_t do_expensive_check, hipgraph_clustering_result_t **result, hipgraph_error_t **error,
)

Spectral clustering.

NOTE: This currently wraps the legacy spectral clustering implementation and is only available in Single GPU implementation.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **n_clusters** – **[in]** The desired number of clusters
- **n_eigenvectors** – **[in]** The number of eigenvectors to use
- **evs_tolerance** – **[in]** The tolerance to use for the eigenvalue solver
- **evs_max_iterations** – **[in]** The maximum number of iterations of the eigenvalue solver
- **k_means_tolerance** – **[in]** The tolerance to use for the k-means solver
- **k_means_max_iterations** – **[in]** The maximum number of iterations of the k-means solver
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to true)
- **result** – **[out]** Opaque object containing the clustering result
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns

error code

hipgraph_error_code_t **hipgraph_analyze_clustering_modularity**(
const hipgraph_resource_handle_t *handle,hipgraph_graph_t *graph,size_t n_clusters,const hipgraph_type_erased_device_array_view_t *vertices,const hipgraph_type_erased_device_array_view_t *clusters,double *score,hipgraph_error_t **error,
)

Compute modularity of the specified clustering.

NOTE: This currently wraps the legacy spectral modularity implementation and is only available in Single GPU implementation.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **n_clusters** – **[in]** The desired number of clusters
- **vertices** – **[in]** Vertex ids from the clustering result
- **clusters** – **[in]** Cluster ids from the clustering result
- **score** – **[out]** The modularity score for this clustering
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns

error code

2.2.8 Spectral Clustering - Edge Cut

hipgraph_error_code_t **hipgraph_analyze_clustering_edge_cut**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, size_t n_clusters, const hipgraph_type_erased_device_array_view_t *vertices, const hipgraph_type_erased_device_array_view_t *clusters, double *score, hipgraph_error_t **error,
)

Compute edge cut of the specified clustering.

NOTE: This currently wraps the legacy spectral edge cut implementation and is only available in Single GPU implementation.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **n_clusters** – **[in]** The desired number of clusters
- **vertices** – **[in]** Vertex ids from the clustering result
- **clusters** – **[in]** Cluster ids from the clustering result
- **score** – **[out]** The edge cut score for this clustering
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

hipgraph_error_code_t **hipgraph_analyze_clustering_ratio_cut**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, size_t n_clusters, const hipgraph_type_erased_device_array_view_t *vertices, const hipgraph_type_erased_device_array_view_t *clusters, double *score, hipgraph_error_t **error,
)

Compute ratio cut of the specified clustering.

NOTE: This currently wraps the legacy spectral ratio cut implementation and is only available in Single GPU implementation.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **n_clusters** – **[in]** The desired number of clusters
- **vertices** – **[in]** Vertex ids from the clustering result
- **clusters** – **[in]** Cluster ids from the clustering result
- **score** – **[out]** The ratio cut score for this clustering
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.2.9 Community Support Functions

hipgraph_type_erased_device_array_view_t *hipgraph_triangle_count_result_get_vertices(
    hipgraph_triangle_count_result_t *result,
)

Get triangle counting vertices.

hipgraph_type_erased_device_array_view_t *hipgraph_triangle_count_result_get_counts(
    hipgraph_triangle_count_result_t *result,
)

Get triangle counting counts.

void hipgraph_triangle_count_result_free(
    hipgraph_triangle_count_result_t *result,
)

Free a triangle count result.

Parameters
result – **[in]** The result from a sampling algorithm

hipgraph_type_erased_device_array_view_t *hipgraph_hierarchical_clustering_result_get_vertices(
    hipgraph_hierarchical_clustering_result_t *result,
)

Get hierarchical clustering vertices.

hipgraph_type_erased_device_array_view_t *hipgraph_hierarchical_clustering_result_get_clusters(
    hipgraph_hierarchical_clustering_result_t *result,
)

Get hierarchical clustering clusters.

double hipgraph_hierarchical_clustering_result_get_modularity(
    hipgraph_hierarchical_clustering_result_t *result,
)

Get modularity.

void hipgraph_hierarchical_clustering_result_free(
    hipgraph_hierarchical_clustering_result_t *result,
)

Free a hierarchical clustering result.

Parameters
result – **[in]** The result from a sampling algorithm

2.3 Core Functions

2.3.1 Core Number

hipgraph_error_code_t hipgraph_core_number(
    const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, *hipgraph_k_core_degree_type_t* degree_type, hipgraph_bool_t do_expensive_check, hipgraph_core_result_t **result, hipgraph_error_t **error,
)

Perform core number.

Parameters

- **handle** – **[in]** Handle for accessing resources

- **graph** – **[in]** Pointer to graph
- **degree_type** – **[in]** Compute core_number using in, out or both in and out edges
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to core number results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.3.2 K-Core

hipgraph_error_code_t **hipgraph_k_core**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, size_t k, *hipgraph_k_core_degree_type_t* degree_type, const hipgraph_core_result_t *core_result, hipgraph_bool_t do_expensive_check, hipgraph_k_core_result_t **result, hipgraph_error_t **error,
)

Perform k_core using output from core_number.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **k** – **[in]** The value of k to use
- **degree_type** – **[in]** Compute core_number using in, out or both in and out edges. Ignored if core_result is specified.
- **core_result** – **[in]** Result from calling hipgraph_core_number, if NULL then call core_number inside this function call.
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to k_core results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.3.3 Core Support Functions

enum **hipgraph_k_core_degree_type_t**

Enumeration for computing core number.

Values:

enumerator **HIPGRAPH_K_CORE_DEGREE_TYPE_IN**

enumerator **HIPGRAPH_K_CORE_DEGREE_TYPE_OUT**

Compute core_number using incoming edges

enumerator **HIPGRAPH_K_CORE_DEGREE_TYPE_INOUT**

Compute core_number using outgoing edges

hipgraph_error_code_t **hipgraph_core_result_create**(
const hipgraph_resource_handle_t *handle, hipgraph_type_erased_device_array_view_t *vertices, hipgraph_type_erased_device_array_view_t *core_numbers, hipgraph_core_result_t **core_result, hipgraph_error_t **error,
)

Create a core_number result (in case it was previously extracted)

Parameters

- **handle** – **[in]** Handle for accessing resources
- **vertices** – **[in]** The result from core number
- **core_numbers** – **[in]** The result from core number
- **core_result** – **[out]** Opaque pointer to core number results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

hipgraph_type_erased_device_array_view_t *__hipgraph_core_result_get_vertices__(
hipgraph_core_result_t *result,
)

Get the vertex ids from the core result.

Parameters
result – **[in]** The result from core number

Returns
type erased array of vertex ids

hipgraph_type_erased_device_array_view_t *__hipgraph_core_result_get_core_numbers__(
hipgraph_core_result_t *result,
)

Get the core numbers from the core result.

Parameters
result – **[in]** The result from core number

Returns
type erased array of core numbers

void **hipgraph_core_result_free**(
hipgraph_core_result_t *result,
)

Free core result.

Parameters
result – **[in]** The result from core number

hipgraph_type_erased_device_array_view_t *hipgraph_k_core_result_get_src_vertices(
    hipgraph_k_core_result_t *result,
)

Get the src vertex ids from the k-core result.

Parameters
`result` – **[in]** The result from k-core

Returns
type erased array of src vertex ids

hipgraph_type_erased_device_array_view_t *hipgraph_k_core_result_get_dst_vertices(
    hipgraph_k_core_result_t *result,
)

Get the dst vertex ids from the k-core result.

Parameters
`result` – **[in]** The result from k-core

Returns
type erased array of dst vertex ids

hipgraph_type_erased_device_array_view_t *hipgraph_k_core_result_get_weights(
    hipgraph_k_core_result_t *result,
)

Get the weights from the k-core result.

Returns NULL if the graph is unweighted

Parameters
`result` – **[in]** The result from k-core

Returns
type erased array of weights

void hipgraph_k_core_result_free(
    hipgraph_k_core_result_t *result,
)

Free k-core result.

Parameters
`result` – **[in]** The result from k-core

2.4 Labeling Functions

2.4.1 Weakly Connected Components

hipgraph_error_code_t hipgraph_weakly_connected_components(
    const hipgraph_resource_handle_t *handle,hipgraph_graph_t *graph,hipgraph_bool_t do_expensive_check,hipgraph_labeling_result_t **result,hipgraph_error_t **error,
)

Labels each vertex in the input graph with its (weakly-connected-)component ID.

The input graph must be symmetric. Component IDs can be arbitrary integers (they can be non-consecutive and are not ordered by component size or any other criterion).

Parameters

- `handle` – **[in]** Handle for accessing resources

- **graph** – **[in]** Pointer to graph
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to labeling results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

2.4.2 Strongly Connected Components

hipgraph_error_code_t **hipgraph_strongly_connected_components**(
const hipgraph_resource_handle_t *handle,hipgraph_graph_t *graph,hipgraph_bool_t do_expensive_check,hipgraph_labeling_result_t **result,hipgraph_error_t **error,
)

Labels each vertex in the input graph with its (strongly-connected-)component ID.

The input graph may be asymmetric. Component IDs can be arbitrary integers (they can be non-consecutive and are not ordered by component size or any other criterion).

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to labeling results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

2.4.3 Labeling Support Functions

hipgraph_type_erased_device_array_view_t *__hipgraph_labeling_result_get_vertices__(
hipgraph_labeling_result_t *result,
)

Get the vertex ids from the labeling result.

Parameters
result – **[in]** The result from a labeling algorithm

Returns
type erased array of vertex ids

hipgraph_type_erased_device_array_view_t *__hipgraph_labeling_result_get_labels__(
hipgraph_labeling_result_t *result,
)

Get the label values from the labeling result.

Parameters
result – **[in]** The result from a labeling algorithm

Returns
type erased array of label values

void **hipgraph_labeling_result_free**(
hipgraph_labeling_result_t *result,
)

Free labeling result.

Parameters
result – **[in]** The result from a labeling algorithm

2.5 Sampling Functions

2.5.1 Uniform Random Walks

hipgraph_error_code_t **hipgraph_uniform_random_walks**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *start_vertices, size_t max_length, hipgraph_random_walk_result_t **result, hipgraph_error_t **error,
)

Compute uniform random walks.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **start_vertices** – **[in]** Array of source vertices
- **max_length** – **[in]** Maximum length of the generated path
- **result** – **[in]** Output from the node2vec call
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.5.2 Biased Random Walks

hipgraph_error_code_t **hipgraph_biased_random_walks**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *start_vertices, size_t max_length, hipgraph_random_walk_result_t **result, hipgraph_error_t **error,
)

Compute biased random walks.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **start_vertices** – **[in]** Array of source vertices
- **max_length** – **[in]** Maximum length of the generated path
- **result** – **[in]** Output from the node2vec call
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.5.3 Random Walks via Node2Vec

hipgraph_error_code_t **hipgraph_node2vec_random_walks**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *start_vertices, size_t max_length, double p, double q, hipgraph_random_walk_result_t **result, hipgraph_error_t **error,
)

Compute random walks using the node2vec framework.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **start_vertices** – **[in]** Array of source vertices
- **max_length** – **[in]** Maximum length of the generated path
- **p** – **[in]** The return parameter
- **q** – **[in]** The inout parameter
- **result** – **[in]** Output from the node2vec call
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.5.4 Node2Vec

hipgraph_error_code_t **hipgraph_node2vec**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *sources, size_t max_depth, hipgraph_bool_t compress_result, double p, double q, hipgraph_random_walk_result_t **result, hipgraph_error_t **error,
)

Compute random walks using the node2vec framework.

Deprecated:
This call should be replaced with hipgraph_node2vec_random_walks

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **sources** – **[in]** Array of source vertices
- **max_depth** – **[in]** Maximum length of the generated path
- **compress_result** – **[in]** If true, return the paths as a compressed sparse row matrix, otherwise return as a dense matrix

- **p** – **[in]** The return parameter
- **q** – **[in]** The inout parameter
- **result** – **[in]** Output from the node2vec call
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.5.5 Uniform Neighbor Sampling

hipgraph_error_code_t **hipgraph_uniform_neighbor_sample**(
const hipgraph_resource_handle_t *handle,hipgraph_graph_t *graph,const hipgraph_type_erased_device_array_view_t *start_vertices,const hipgraph_type_erased_device_array_view_t *start_vertex_labels,const hipgraph_type_erased_device_array_view_t *label_list,const hipgraph_type_erased_device_array_view_t *label_to_comm_rank,const hipgraph_type_erased_device_array_view_t *label_offsets,const hipgraph_type_erased_host_array_view_t *fan_out,hipgraph_rng_state_t *rng_state,const hipgraph_sampling_options_t *options,hipgraph_bool_t do_expensive_check,hipgraph_sample_result_t **result,hipgraph_error_t **error,
)

Uniform Neighborhood Sampling.

Returns a sample of the neighborhood around specified start vertices. Optionally, each start vertex can be associated with a label, allowing the caller to specify multiple batches of sampling requests in the same function call - which should improve GPU utilization.

If label is NULL then all start vertices will be considered part of the same batch and the return value will not have a label column.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **start_vertices** – **[in]** Device array of start vertices for the sampling
- **start_vertex_labels** – **[in]** Device array of start vertex labels for the sampling. The labels associated with each start vertex will be included in the output associated with results that were derived from that start vertex. We only support label of type INT32. If label is NULL, the return data will not be labeled.
- **label_list** – **[in]** Device array of the labels included in `start_vertex_labels`. If `label_to_comm_rank` is not specified this parameter is ignored. If specified, label_list must be sorted in ascending order.
- **label_to_comm_rank** – **[in]** Device array identifying which comm rank the output for a particular label should be shuffled in the output. If not specifed the data is not organized in output. If specified then the all data from `label_list`[i] will be shuffled to rank. This cannot be specified unless `start_vertex_labels` is also specified label_to_comm_rank[i]. If not specified then the output data will not be shuffled between ranks.
- **label_offsets** – **[in]** tbd
- **fan_out** – **[in]** Host array defining the fan out at each step in the sampling algorithm. We only support fanout values of type INT32
- **rng_state** – **[inout]** State of the random number generator, updated with each call

- **options** – **[in]** Opaque pointer defining the sampling options.
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to true)
- **result** – **[in]** Output from the uniform_neighbor_sample call
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.5.6 Sampling Support Functions

size_t **hipgraph_random_walk_result_get_max_path_length**(
hipgraph_random_walk_result_t *result,
)

Get the max path length from random walk result.

Parameters
result – **[in]** The result from random walks

Returns
maximum path length

hipgraph_type_erased_device_array_view_t ***hipgraph_random_walk_result_get_paths**(
hipgraph_random_walk_result_t *result,
)

Get the matrix (row major order) of vertices in the paths.

Parameters
result – **[in]** The result from a random walk algorithm

Returns
type erased array pointing to the path matrix in device memory

hipgraph_type_erased_device_array_view_t ***hipgraph_random_walk_result_get_weights**(
hipgraph_random_walk_result_t *result,
)

Get the matrix (row major order) of edge weights in the paths.

Parameters
result – **[in]** The result from a random walk algorithm

Returns
type erased array pointing to the path edge weights in device memory

hipgraph_type_erased_device_array_view_t ***hipgraph_random_walk_result_get_path_sizes**(
hipgraph_random_walk_result_t *result,
)

If the random walk result is compressed, get the path sizes.

Deprecated:

This call will no longer be relevant once the new node2vec are called

Parameters
result – **[in]** The result from a random walk algorithm

Returns
type erased array pointing to the path sizes in device memory

void **hipgraph_random_walk_result_free**(
hipgraph_random_walk_result_t *result,
)

Free random walks result.

Parameters
result – **[in]** The result from random walks

hipgraph_error_code_t **hipgraph_sampling_options_create**(
hipgraph_sampling_options_t **options,hipgraph_error_t **error,
)

Create sampling options object.

All sampling options set to FALSE

Parameters

- **options** – **[out]** Opaque pointer to the sampling options
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

void **hipgraph_sampling_set_renumber_results**(
hipgraph_sampling_options_t *options,hipgraph_bool_t value,
)

Set flag to renumber results.

Parameters

- **options** – - opaque pointer to the sampling options
- **value** – - Boolean value to assign to the option

void **hipgraph_sampling_set_compress_per_hop**(
hipgraph_sampling_options_t *options,hipgraph_bool_t value,
)

Set whether to compress per-hop (True) or globally (False)

Parameters

- **options** – - opaque pointer to the sampling options
- **value** – - Boolean value to assign to the option

void **hipgraph_sampling_set_with_replacement**(
hipgraph_sampling_options_t *options,hipgraph_bool_t value,
)

Set flag to sample with_replacement.

Parameters

- **options** – - opaque pointer to the sampling options
- **value** – - Boolean value to assign to the option

void **hipgraph_sampling_set_return_hops**(
hipgraph_sampling_options_t *options,hipgraph_bool_t value,
)

Set flag to sample return_hops.

Parameters

- **options** – - opaque pointer to the sampling options
- **value** – - Boolean value to assign to the option

void **hipgraph_sampling_set_compression_type**(
hipgraph_sampling_options_t *options, hipgraph_compression_type_t value,
)

Set compression type.

Parameters

- **options** – - opaque pointer to the sampling options
- **value** – - Enum defining the compresion type

void **hipgraph_sampling_set_prior_sources_behavior**(
hipgraph_sampling_options_t *options, hipgraph_prior_sources_behavior_t value,
)

Set prior sources behavior.

Parameters

- **options** – - opaque pointer to the sampling options
- **value** – - Enum defining prior sources behavior

void **hipgraph_sampling_set_dedupe_sources**(
hipgraph_sampling_options_t *options, hipgraph_bool_t value,
)

Set flag to sample dedupe_sources prior to sampling.

Parameters

- **options** – - opaque pointer to the sampling options
- **value** – - Boolean value to assign to the option

void **hipgraph_sampling_options_free**(
hipgraph_sampling_options_t *options,
)

Free sampling options object.

Parameters
options – **[in]** Opaque pointer to sampling object

hipgraph_type_erased_device_array_view_t ***hipgraph_sample_result_get_majors**(
const hipgraph_sample_result_t *result,
)

Get the major vertices from the sampling algorithm result.

Parameters
result – **[in]** The result from a sampling algorithm

Returns
type erased array pointing to the major vertices in device memory

hipgraph_type_erased_device_array_view_t ***hipgraph_sample_result_get_minors**(
const hipgraph_sample_result_t *result,
)

Get the minor vertices from the sampling algorithm result.

Parameters
result – **[in]** The result from a sampling algorithm

Returns
type erased array pointing to the minor vertices in device memory

hipgraph_type_erased_device_array_view_t *hipgraph_sample_result_get_major_offsets(
const hipgraph_sample_result_t *result,
)

Get the major offsets from the sampling algorithm result.

Parameters
result – **[in]** The result from a sampling algorithm

Returns
type erased array pointing to the major offsets in device memory

hipgraph_type_erased_device_array_view_t *hipgraph_sample_result_get_start_labels(
const hipgraph_sample_result_t *result,
)

Get the start labels from the sampling algorithm result.

Parameters
result – **[in]** The result from a sampling algorithm

Returns
type erased array pointing to the start labels

hipgraph_type_erased_device_array_view_t *hipgraph_sample_result_get_edge_id(
const hipgraph_sample_result_t *result,
)

Get the edge_id from the sampling algorithm result.

Parameters
result – **[in]** The result from a sampling algorithm

Returns
type erased array pointing to the edge_id

hipgraph_type_erased_device_array_view_t *hipgraph_sample_result_get_edge_type(
const hipgraph_sample_result_t *result,
)

Get the edge_type from the sampling algorithm result.

Parameters
result – **[in]** The result from a sampling algorithm

Returns
type erased array pointing to the edge_type

hipgraph_type_erased_device_array_view_t *hipgraph_sample_result_get_edge_weight(
const hipgraph_sample_result_t *result,
)

Get the edge_weight from the sampling algorithm result.

Parameters
result – **[in]** The result from a sampling algorithm

Returns
type erased array pointing to the edge_weight

hipgraph_type_erased_device_array_view_t *hipgraph_sample_result_get_hop(
    const hipgraph_sample_result_t *result,
)

Get the hop from the sampling algorithm result.

Parameters
`result` – **[in]** The result from a sampling algorithm

Returns
type erased array pointing to the hop

hipgraph_type_erased_device_array_view_t *hipgraph_sample_result_get_label_hop_offsets(
    const hipgraph_sample_result_t *result,
)

Get the label-hop offsets from the sampling algorithm result.

Parameters
`result` – **[in]** The result from a sampling algorithm

Returns
type erased array pointing to the label-hop offsets

hipgraph_type_erased_device_array_view_t *hipgraph_sample_result_get_index(
    const hipgraph_sample_result_t *result,
)

Get the index from the sampling algorithm result.

Parameters
`result` – **[in]** The result from a sampling algorithm

Returns
type erased array pointing to the index

hipgraph_type_erased_device_array_view_t *hipgraph_sample_result_get_renumber_map(
    const hipgraph_sample_result_t *result,
)

Get the renumber map.

Parameters
`result` – **[in]** The result from a sampling algorithm

Returns
type erased array pointing to the renumber map

hipgraph_type_erased_device_array_view_t *hipgraph_sample_result_get_renumber_map_offsets(
    const hipgraph_sample_result_t *result,
)

Get the renumber map offsets.

Parameters
`result` – **[in]** The result from a sampling algorithm

Returns
type erased array pointing to the renumber map offsets

void hipgraph_sample_result_free(
    hipgraph_sample_result_t *result,
)

Free a sampling result.

Parameters
result – **[in]** The result from a sampling algorithm

hipgraph_error_code_t **hipgraph_test_sample_result_create**(
const hipgraph_resource_handle_t *handle, const hipgraph_type_erased_device_array_view_t *srcs, const hipgraph_type_erased_device_array_view_t *dsts, const hipgraph_type_erased_device_array_view_t *edge_id, const hipgraph_type_erased_device_array_view_t *edge_type, const hipgraph_type_erased_device_array_view_t *wgt, const hipgraph_type_erased_device_array_view_t *hop, const hipgraph_type_erased_device_array_view_t *label, hipgraph_sample_result_t **result, hipgraph_error_t **error,
)

Create a sampling result (testing API)

Parameters

- **handle** – **[in]** Handle for accessing resources
- **srcs** – **[in]** Device array view to populate srcs
- **dsts** – **[in]** Device array view to populate dsts
- **edge_id** – **[in]** Device array view to populate edge_id (can be NULL)
- **edge_type** – **[in]** Device array view to populate edge_type (can be NULL)
- **wgt** – **[in]** Device array view to populate wgt (can be NULL)
- **hop** – **[in]** Device array view to populate hop
- **label** – **[in]** Device array view to populate label (can be NULL)
- **result** – **[out]** Pointer to the location to store the hipgraph_sample_result_t*
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

hipgraph_error_code_t **hipgraph_test_uniform_neighborhood_sample_result_create**(
const hipgraph_resource_handle_t *handle, const hipgraph_type_erased_device_array_view_t *srcs, const hipgraph_type_erased_device_array_view_t *dsts, const hipgraph_type_erased_device_array_view_t *edge_id, const hipgraph_type_erased_device_array_view_t *edge_type, const hipgraph_type_erased_device_array_view_t *weight, const hipgraph_type_erased_device_array_view_t *hop, const hipgraph_type_erased_device_array_view_t *label, hipgraph_sample_result_t **result, hipgraph_error_t **error,
)

Create a sampling result (testing API)

Parameters

- **handle** – **[in]** Handle for accessing resources
- **srcs** – **[in]** Device array view to populate srcs
- **dsts** – **[in]** Device array view to populate dsts
- **edge_id** – **[in]** Device array view to populate edge_id
- **edge_type** – **[in]** Device array view to populate edge_type
- **weight** – **[in]** Device array view to populate weight
- **hop** – **[in]** Device array view to populate hop

- **label** – **[in]** Device array view to populate label
- **result** – **[out]** Pointer to the location to store the hipgraph_sample_result_t*
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

hipgraph_error_code_t **hipgraph_select_random_vertices**(
const hipgraph_resource_handle_t *handle,const hipgraph_graph_t *graph,hipgraph_rng_state_t *rng_state,size_t num_vertices,hipgraph_type_erased_device_array_t **vertices,hipgraph_error_t **error,
)

Select random vertices from the graph.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **rng_state** – **[inout]** State of the random number generator, updated with each call
- **num_vertices** – **[in]** Number of vertices to sample
- **vertices** – **[out]** Device array view to populate label
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.6 Similarity Functions

2.6.1 Jaccard

hipgraph_error_code_t **hipgraph_jaccard_coefficients**(
const hipgraph_resource_handle_t *handle,hipgraph_graph_t *graph,const hipgraph_vertex_pairs_t *vertex_pairs,hipgraph_bool_t use_weight,hipgraph_bool_t do_expensive_check,hipgraph_similarity_result_t **result,hipgraph_error_t **error,
)

Perform Jaccard similarity computation.

Compute the similarity for the specified vertex_pairs

Note that Jaccard similarity must run on a symmetric graph.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **vertex_pairs** – **[in]** Vertex pair for input
- **use_weight** – **[in]** If true consider the edge weight in the graph, if false use an edge weight of 1
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).

- **result** – **[out]** Opaque pointer to similarity results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.6.2 Sorensen

hipgraph_error_code_t **hipgraph_sorensen_coefficients**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_vertex_pairs_t *vertex_pairs, hipgraph_bool_t use_weight, hipgraph_bool_t do_expensive_check, hipgraph_similarity_result_t **result, hipgraph_error_t **error,
)

Perform Sorensen similarity computation.

Compute the similarity for the specified vertex_pairs

Note that Sorensen similarity must run on a symmetric graph.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **vertex_pairs** – **[in]** Vertex pair for input
- **use_weight** – **[in]** If true consider the edge weight in the graph, if false use an edge weight of 1
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to similarity results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.6.3 Overlap

hipgraph_error_code_t **hipgraph_overlap_coefficients**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_vertex_pairs_t *vertex_pairs, hipgraph_bool_t use_weight, hipgraph_bool_t do_expensive_check, hipgraph_similarity_result_t **result, hipgraph_error_t **error,
)

Perform overlap similarity computation.

Compute the similarity for the specified vertex_pairs

Note that overlap similarity must run on a symmetric graph.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **vertex_pairs** – **[in]** Vertex pair for input

- **use_weight** – **[in]** If true consider the edge weight in the graph, if false use an edge weight of 1
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to similarity results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.6.4 Similarty Support Functions

hipgraph_vertex_pairs_t ***hipgraph_similarity_result_get_vertex_pairs**(
hipgraph_similarity_result_t *result,
)

Get vertex pair from the similarity result.

Parameters
result – **[in]** The result from a similarity algorithm

Returns
vertex pairs

hipgraph_type_erased_device_array_view_t ***hipgraph_similarity_result_get_similarity**(
hipgraph_similarity_result_t *result,
)

Get the similarity coefficient array.

Parameters
result – **[in]** The result from a similarity algorithm

Returns
type erased array of similarity coefficients

void **hipgraph_similarity_result_free**(
hipgraph_similarity_result_t *result,
)

Free similarity result.

Parameters
result – **[in]** The result from a similarity algorithm

2.7 Traversal Functions

2.7.1 Breadth First Search (BFS)

hipgraph_error_code_t **hipgraph_bfs**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, hipgraph_type_erased_device_array_view_t *sources, hipgraph_bool_t direction_optimizing, size_t depth_limit, hipgraph_bool_t compute_predecessors, hipgraph_bool_t do_expensive_check, hipgraph_paths_result_t **result, hipgraph_error_t **error,
)

Perform a breadth first search from a set of seed vertices.

This function computes the distances (minimum number of hops to reach the vertex) from the source vertex. If `predecessors` is not NULL, this function calculates the predecessor of each vertex (parent vertex in the breadth-first search tree) as well.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph FIXME: Make this just [in], copy it if I need to temporarily modify internally
- **sources** – **[inout]** Array of source vertices. NOTE: Array might be modified if renumbering is enabled for the graph
- **direction_optimizing** – **[in]** If set to true, this algorithm switches between the push based breadth-first search and pull based breadth-first search depending on the size of the breadth-first search frontier (currently unsupported). This option is valid only for symmetric input graphs.
- **depth_limit** – Sets the maximum number of breadth-first search iterations. Any vertices farther than `depth_limit` hops from `source_vertex` will be marked as unreachable.
- **compute_predecessors** – **[in]** A flag to indicate whether to compute the predecessors in the result
- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to paths results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns

error code

2.7.2 Single-Source Shortest-Path (SSSP)

hipgraph_error_code_t **hipgraph_sssp**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, size_t source, double cutoff, hipgraph_bool_t compute_predecessors, hipgraph_bool_t do_expensive_check, hipgraph_paths_result_t **result, hipgraph_error_t **error,
)

Perform single-source shortest-path to compute the minimum distances (and predecessors) from the source vertex.

This function computes the distances (minimum edge weight sums) from the source vertex. If `predecessors` is not NULL, this function calculates the predecessor of each vertex (parent vertex in the breadth-first search tree) as well.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph
- **source** – **[in]** Source vertex id
- **cutoff** – **[in]** Maximum edge weight sum to consider
- **compute_predecessors** – **[in]** A flag to indicate whether to compute the predecessors in the result

- **do_expensive_check** – **[in]** A flag to run expensive checks for input arguments (if set to `true`).
- **result** – **[out]** Opaque pointer to paths results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.7.3 Path Extraction

hipgraph_error_code_t **hipgraph_extract_paths**(
const hipgraph_resource_handle_t *handle, hipgraph_graph_t *graph, const hipgraph_type_erased_device_array_view_t *sources, const hipgraph_paths_result_t *paths_result, const hipgraph_type_erased_device_array_view_t *destinations, hipgraph_extract_paths_result_t **result, hipgraph_error_t **error,
)

Extract BFS or SSSP paths from a hipgraph_paths_result_t.

This function extracts paths from the BFS or SSSP output. BFS and SSSP output distances and predecessors. The path from a vertex v back to the original source vertex can be extracted by recursively looking up the predecessor vertex until you arrive back at the original source vertex.

Parameters

- **handle** – **[in]** Handle for accessing resources
- **graph** – **[in]** Pointer to graph. NOTE: Graph might be modified if the storage needs to be transposed
- **sources** – **[in]** Array of source vertices
- **paths_result** – **[in]** Output from the BFS call
- **destinations** – **[in]** Array of destination vertices.
- **result** – **[out]** Opaque pointer to extract_paths results
- **error** – **[out]** Pointer to an error object storing details of any error. Will be populated if error code is not HIPGRAPH_SUCCESS

Returns
error code

2.7.4 Extract Max Path Length

size_t **hipgraph_extract_paths_result_get_max_path_length**(
hipgraph_extract_paths_result_t *result,
)

Get the max path length from extract_paths result.

Parameters
result – **[in]** The result from extract_paths

Returns
maximum path length

2.7.5 Traversal Support Functions

hipgraph_type_erased_device_array_view_t *hipgraph_paths_result_get_vertices(
    hipgraph_paths_result_t *result,
)

Get the vertex ids from the paths result.

Parameters
result – **[in]** The result from bfs or sssp

Returns
type erased array of vertex ids

hipgraph_type_erased_device_array_view_t *hipgraph_paths_result_get_distances(
    hipgraph_paths_result_t *result,
)

Get the distances from the paths result.

Parameters
result – **[in]** The result from bfs or sssp

Returns
type erased array of distances

hipgraph_type_erased_device_array_view_t *hipgraph_paths_result_get_predecessors(
    hipgraph_paths_result_t *result,
)

Get the predecessors from the paths result.

Parameters
result – **[in]** The result from bfs or sssp

Returns
type erased array of predecessors. Value will be NULL if compute_predecessors was FALSE in the call to bfs or sssp that produced this result.

void hipgraph_paths_result_free(
    hipgraph_paths_result_t *result,
)

Free paths result.

Parameters
result – **[in]** The result from bfs or sssp

hipgraph_type_erased_device_array_view_t *hipgraph_extract_paths_result_get_paths(
    hipgraph_extract_paths_result_t *result,
)

Get the matrix (row major order) of paths.

Parameters
result – **[in]** The result from extract_paths

Returns
type erased array pointing to the matrix in device memory

void hipgraph_extract_paths_result_free(
    hipgraph_extract_paths_result_t *result,
)

Free extract_paths result.

Parameters

`result` – **[in]** The result from extract_paths

CHAPTER

THREE

HIPGRAPH PYTHON API DOCUMENTATION

Note

The early access version of the hipGRAPH Python layer is untested and the Python API is unsupported.

This directory contains the sources to the *pylibhipgraph* package. The sources are primarily cython files which are built using the *setup.py* file in the parent directory and depend on the *libhipgraph_c* and *libhipgraph* libraries and headers.

3.1 pylibhipgraph.all_pairs_cosine_coefficients

all_pairs_cosine_coefficients (*ResourceHandle resource_handle, _GPUGraph graph, vertices, bool_t use_weight, topk, bool_t do_expensive_check*)

Perform All-Pairs Cosine similarity computation.

Note

Cosine similarity must run on a symmetric graph.

3.1.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

vertices
: [cudf.Series or None]

  Vertex list to compute all-pairs. If None, then compute based
  on all vertices in the graph.

use_weight
: [bool, optional] If set to True, then compute the weighted cosine_coefficients (the input graph must be weighted in that case). Otherwise, compute the non-weighted cosine_coefficients.

topk
: [size_t] Specify the number of answers to return otherwise will return all values.

do_expensive_check
[bool] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.1.2 Returns

A tuple of device arrays containing the vertex pairs with their corresponding Cosine coefficient scores.

3.2 pylibhipgraph.all_pairs_jaccard_coefficients

all_pairs_jaccard_coefficients (*ResourceHandle resource_handle, _GPUGraph graph, vertices, bool_t use_weight, topk, bool_t do_expensive_check*)

Perform All-Pairs Jaccard similarity computation.

> **Note**
>
> Jaccard similarity must run on a symmetric graph.

3.2.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

vertices
[cudf.Series or None]

Vertex list to compute all-pairs. If None, then compute based
on all vertices in the graph.

use_weight
[bool, optional] If set to True, then compute the weighted jaccard_coefficients (the input graph must be weighted in that case). Otherwise, compute the non-weighted jaccard_coefficients.

topk
[size_t] Specify the number of answers to return otherwise will return all values.

do_expensive_check
[bool] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.2.2 Returns

A tuple of device arrays containing the vertex pairs with their corresponding Jaccard coefficient scores.

3.3 pylibhipgraph.all_pairs_overlap_coefficients

all_pairs_overlap_coefficients (*ResourceHandle resource_handle, _GPUGraph graph, vertices, bool_t use_weight, topk, bool_t do_expensive_check*)

Perform All-Pairs Overlap similarity computation.

Note

Overlap similarity must run on a symmetric graph.

3.3.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

vertices
[cudf.Series or None]

Vertex list to compute all-pairs. If None, then compute based
on all vertices in the graph.

use_weight
[bool, optional] If set to True, then compute the weighted overlap_coefficients (the input graph must be weighted in that case). Otherwise, compute the non-weighted overlap_coefficients.

topk
[size_t] Specify the number of answers to return otherwise will return all values.

do_expensive_check
[bool] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.3.2 Returns

A tuple of device arrays containing the vertex pairs with their corresponding Overlap coefficient scores.

3.4 pylibhipgraph.all_pairs_sorensen_coefficients

all_pairs_sorensen_coefficients (*ResourceHandle resource_handle, _GPUGraph graph, vertices, bool_t use_weight, topk, bool_t do_expensive_check*)

Perform All-Pairs Sorensen similarity computation.

Note

Sorensen similarity must run on a symmetric graph.

3.4.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

vertices
[cudf.Series or None]

Vertex list to compute all-pairs. If None, then compute based
on all vertices in the graph.

use_weight
[bool, optional] If set to True, then compute the weighted sorensen_coefficients (the input graph must be weighted in that case). Otherwise, compute the non-weighted sorensen_coefficients.

topk
[size_t] Specify the number of answers to return otherwise will return all values.

do_expensive_check
[bool] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.4.2 Returns

A tuple of device arrays containing the vertex pairs with their corresponding Sorensen coefficient scores.

3.5 pylibhipgraph.analyze_clustering_edge_cut

analyze_clustering_edge_cut (*ResourceHandle resource_handle, _GPUGraph graph, size_t num_clusters, vertex, cluster*)

Compute edge cut score of the specified clustering.

3.5.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph] The input graph.

num_clusters
[size_t] Specifies the number of clusters to find, must be greater than 1.

vertex
[device array type] Vertex ids from the clustering to analyze.

cluster
[device array type] Cluster ids from the clustering to analyze.

3.5.2 Returns

The edge cut score of the specified clustering.

3.5.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 0], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=True, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
```

(continues on next page)

(continued from previous page)

```
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertex, cluster) = pylibhipgraph.spectral_modularity_maximization(
...     resource_handle, G, num_clusters=5, num_eigen_vects=2, evs_tolerance=0.00001
...     evs_max_iter=100, kmean_tolerance=0.00001, kmean_max_iter=100)
    # FIXME: Fix docstring result.
>>> vertices
############
>>> clusters
############
>>> score = pylibhipgraph.analyze_clustering_edge_cut(
...     resource_handle, G, num_clusters=5, vertex=vertex, cluster=cluster)
>>> score
############
```

3.6 pylibhipgraph.analyze_clustering_modularity

analyze_clustering_modularity (*ResourceHandle resource_handle, _GPUGraph graph, size_t num_clusters, vertex, cluster*)

Compute modularity score of the specified clustering.

3.6.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph] The input graph.

num_clusters
: [size_t] Specifies the number of clusters to find, must be greater than 1.

vertex
: [device array type] Vertex ids from the clustering to analyze.

cluster
: [device array type] Cluster ids from the clustering to analyze.

3.6.2 Returns

The modularity score of the specified clustering.

3.6.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 0], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=True, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
```

(continues on next page)

(continued from previous page)

```
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertex, cluster) = pylibhipgraph.spectral_modularity_maximization(
...     resource_handle, G, num_clusters=5, num_eigen_vects=2, evs_tolerance=0.00001
...     evs_max_iter=100, kmean_tolerance=0.00001, kmean_max_iter=100)
    # FIXME: Fix docstring result.
>>> vertices
############
>>> clusters
############
>>> score = pylibhipgraph.analyze_clustering_modularity(
...     resource_handle, G, num_clusters=5, vertex=vertex, cluster=cluster)
>>> score
############
```

3.7 pylibhipgraph.analyze_clustering_ratio_cut

analyze_clustering_ratio_cut (*ResourceHandle resource_handle, _GPUGraph graph, size_t num_clusters, vertex, cluster*)

Compute ratio cut score of the specified clustering.

3.7.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph] The input graph.

num_clusters
: [size_t] Specifies the number of clusters to find, must be greater than 1.

vertex
: [device array type] Vertex ids from the clustering to analyze.

cluster
: [device array type] Cluster ids from the clustering to analyze.

3.7.2 Returns

The ratio cut score of the specified clustering.

3.7.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 0], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=True, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
```

(continues on next page)

(continued from previous page)

```
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertex, cluster) = pylibhipgraph.spectral_modularity_maximization(
...     resource_handle, G, num_clusters=5, num_eigen_vects=2, evs_tolerance=0.00001
...     evs_max_iter=100, kmean_tolerance=0.00001, kmean_max_iter=100)
    # FIXME: Fix docstring result.
>>> vertices
############
>>> clusters
############
>>> score = pylibhipgraph.analyze_clustering_ratio_cut(
...     resource_handle, G, num_clusters=5, vertex=vertex, cluster=cluster)
>>> score
############
```

3.8 pylibhipgraph.balanced_cut_clustering

balanced_cut_clustering (*ResourceHandle resource_handle, _GPUGraph graph, num_clusters, num_eigen_vects, evs_tolerance, evs_max_iter, kmean_tolerance, kmean_max_iter, bool_t do_expensive_check*)

Compute a clustering/partitioning of the given graph using the spectral balanced cut method.

3.8.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph] The input graph.

num_clusters
: [size_t] Specifies the number of clusters to find, must be greater than 1.

num_eigen_vects
: [size_t] Specifies the number of eigenvectors to use. Must be lower or equal to num_clusters.

evs_tolerance: double
: Specifies the tolerance to use in the eigensolver.

evs_max_iter: size_t
: Specifies the maximum number of iterations for the eigensolver.

kmean_tolerance: double
: Specifies the tolerance to use in the k-means solver.

kmean_max_iter: size_t
: Specifies the maximum number of iterations for the k-means solver.

do_expensive_check
: [bool_t] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.8.2 Returns

A tuple containing the clustering vertices, clusters

3.8.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 0], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=True, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertices, clusters) = pylibhipgraph.balanced_cut_clustering(
...     resource_handle, G, num_clusters=5, num_eigen_vects=2, evs_tolerance=0.00001
...     evs_max_iter=100, kmean_tolerance=0.00001, kmean_max_iter=100)
# FIXME: Fix docstring results.
>>> vertices
############
>>> clusters
############
```

3.9 pylibhipgraph.betweenness_centrality

betweenness_centrality (*ResourceHandle resource_handle, _GPUGraph graph, k, random_state, bool_t normalized bool_t include_endpoints, bool_t do_expensive_check*)

Compute the betweenness centrality for all vertices of the graph G. Betweenness centrality is a measure of the number of shortest paths that pass through a vertex. A vertex with a high betweenness centrality score has more paths passing through it and is therefore believed to be more important.

3.9.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

k
: [int or device array type or None, optional (default=None)] If k is not None, use k node samples to estimate betweenness. Higher values give better approximation. If k is a device array type, use the content of the list for estimation: the list should contain vertex identifiers. If k is None (the default), all the vertices are used to estimate betweenness. Vertices obtained through sampling or defined as a list will be used as sources for traversals inside the algorithm.

random_state
: [int, optional (default=None)] if k is specified and k is an integer, use random_state to initialize the random number generator. Using None defaults to a hash of process id, time, and hostname If k is either None or list or cudf objects: random_state parameter is ignored.

normalized
: [bool_t] Normalization will ensure that values are in [0, 1].

include_endpoints
: [bool_t] If true, include the endpoints in the shortest path counts.

do_expensive_check
[bool_t] A flag to run expensive checks for input arguments if True.

3.10 pylibhipgraph.bfs

bfs (*ResourceHandle handle, _GPUGraph graph, sources, bool_t direction_optimizing, int32_t depth_limit, bool_t compute_predecessors, bool_t do_expensive_check*)

Performs a Breadth-first search starting from the provided sources. Returns the distances, and predecessors if requested.

3.10.1 Parameters

handle: ResourceHandle
The resource handle responsible for managing device resources that this algorithm will use

graph: SGGraph or MGGraph
The graph to operate upon

sources: cudf.Series
The vertices to start the breadth-first search from. Should match the numbering of the provided graph. All workers must have a unique set of sources. Empty sets are allowed as long as at least one worker has a source.

direction_optimizing: bool_t
Whether to treat the graph as undirected (should only be called on a symmetric graph)

depth_limit: int32_t
The depth limit at which the traversal will be stopped. If this is a negative number, the traversal will run without a depth limit.

compute_predecessors: bool_t
Whether to compute the predecessors. If left blank, -1 will be returned instead of the correct predecessor of each vertex.

do_expensive_check
[bool_t] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.10.2 Returns

A tuple of device arrays (cupy arrays) of the form (distances, predecessors, vertices)

3.10.3 Examples

```
>>> M = cudf.read_csv(datasets_path / 'karate.csv', delimiter=' ',
>>>                   dtype=['int32', 'int32', 'float32'], header=None)
>>> G = hipgraph.Graph()
>>>  G.from_cudf_edgelist(M, source='0', destination='1', edge_attr='2')
>>>
>>> handle = ResourceHandle()
>>>
>>> srcs = G.edgelist.edgelist_df['src']
>>> dsts = G.edgelist.edgelist_df['dst']
>>> weights = G.edgelist.edgelist_df['weights']
>>>
>>> sg = SGGraph(
>>>     resource_handle = handle,
```

(continues on next page)

(continued from previous page)

```
>>>     graph_properties = GraphProperties(is_multigraph=G.is_multigraph()),
>>>     src_array = srcs,
>>>     dst_array = dsts,
>>>     weight_array = weights,
>>>     store_transposed=False,
>>>     renumber=False,
>>>     do_expensive_check=do_expensive_check
>>> )
>>>
>>> res = pylibhipgraph_bfs(
>>>         handle,
>>>         sg,
>>>         cudf.Series([0], dtype='int32'),
>>>         False,
>>>         10,
>>>         True,
>>>         False
>>> )
>>>
>>> distances, predecessors, vertices = res
>>>
f>>> inal_results = cudf.DataFrame({
>>>     'distance': cudf.Series(distances),
>>>     'vertex': cudf.Series(vertices),
>>>     'predecessor': cudf.Series(predecessors),
>>> })
```

3.11 pylibhipgraph.core_number

core_number (*ResourceHandle resource_handle, _GPUGraph graph, degree_type, bool_t do_expensive_check*)

Computes core number.

3.11.1 Parameters

resource_handle: ResourceHandle
Handle to the underlying device and host resource needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

degree_type: str
This option determines if the core number computation should be based on input, output, or both directed edges, with valid values being “incoming”, “outgoing”, and “bidirectional” respectively. This option is currently ignored in this release, and setting it will result in a warning.

do_expensive_check: bool
If True, performs more extensive tests on the inputs to ensure validity, at the expense of increased run time.

3.11.2 Returns

A tuple of device arrays, where the first item in the tuple is a device array containing the vertices and the second item in the tuple is a device array containing the core numbers for the corresponding vertices.

3.12 pylibhipgraph.cosine_coefficients

cosine_coefficients (*ResourceHandle resource_handle, _GPUGraph graph, first, second, bool_t use_weight, bool_t do_expensive_check*)

Compute the Cosine coefficients for the specified vertex_pairs.

> **Note**
>
> Cosine similarity must run on a symmetric graph.

3.12.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

first :
: Source of the vertex pair.

second :
: Destination of the vertex pair.

use_weight
: [bool, optional] If set to True, compute the weighted cosine_coefficients (the input graph must be weighted in that case). Otherwise, computed the un-weighted cosine_coefficients.

do_expensive_check
: [bool] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.12.2 Returns

A tuple of device arrays containing the vertex pairs with their corresponding Cosine coefficient scores.

3.13 pylibhipgraph.count_multi_edges

count_multi_edges (*ResourceHandle resource_handle, _GPUGraph graph, bool_t do_expensive_check*)

Count the number of multi-edges in the graph. This returns the number of duplicates. If the edge (u, v) appears k times in the graph, then that edge will contribute (k-1) toward the total number of duplicates.

3.13.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

do_expensive_check
[bool_t] A flag to run expensive checks for input arguments if True.

3.13.2 Returns

Total count of duplicate edges in the graph

3.13.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 0, 0], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 1, 1], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> count = pylibhipgraph.count_multi_edges(resource_handle, G, False)
```

3.14 pylibhipgraph.degrees

3.14.1 pylibhipgraph.in_degrees

in_degrees (*ResourceHandle resource_handle, _GPUGraph graph, source_vertices, bool_t do_expensive_check*)

Compute the in degrees for the nodes of the graph.

Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

source_vertices
[cupy array] The nodes for which we will compute degrees.

do_expensive_check
[bool_t] A flag to run expensive checks for input arguments if True.

Returns

A tuple of device arrays, where the first item in the tuple is a device array containing the vertices, the second item in the tuple is a device array containing the in degrees for the vertices.

Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 3], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
```

(continues on next page)

(continued from previous page)

```
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertices, in_degrees) = pylibhipgraph.in_degrees(
                               resource_handle, G, None, False)
```

3.14.2 pylibhipgraph.out_degrees

out_degrees (*ResourceHandle resource_handle _GPUGraph graph, source_vertices, bool_t do_expensive_check*)

Compute the out degrees for the nodes of the graph.

Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

source_vertices
: [cupy array] The nodes for which we will compute degrees.

do_expensive_check
: [bool_t] A flag to run expensive checks for input arguments if True.

Returns

A tuple of device arrays, where the first item in the tuple is a device array containing the vertices, the second item in the tuple is a device array containing the out degrees for the vertices.

Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 3], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertices, out_degrees) = pylibhipgraph.out_degrees(
                                resource_handle, G, None, False)
```

3.14.3 pylibhipgraph.degrees

degrees (*ResourceHandle resource_handle, _GPUGraph graph, source_vertices, bool_t do_expensive_check*)

Compute the degrees for the nodes of the graph.

Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

source_vertices
[cupy array] The nodes for which we will compute degrees.

do_expensive_check
[bool_t] A flag to run expensive checks for input arguments if True.

Returns

A tuple of device arrays, where the first item in the tuple is a device array containing the vertices, the second item in the tuple is a device array containing the in degrees for the vertices, the third item in the tuple is a device array containing the out degrees for the vertices.

Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 3], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertices, in_degrees, out_degrees) = pylibhipgraph.degrees(
                                          resource_handle, G, None, False)
```

3.15 pylibhipgraph.ecg

ecg (*ResourceHandle resource_handle, random_state, _GPUGraph graph, double min_weight, size_t ensemble_size, size_t max_level, double threshold, double resolution, bool_t do_expensive_check*)

Compute the Ensemble Clustering for Graphs (ECG) partition of the input graph. ECG runs truncated Louvain on an ensemble of permutations of the input graph, then uses the ensemble partitions to determine weights for the input graph. The final result is found by running full Louvain on the input graph using the determined weights.

See https://arxiv.org/abs/1809.05578 for further information.

3.15.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

random_state
[int , optional] Random state to use when generating samples. Optional argument, defaults to a hash of process id, time, and hostname. (See pylibhipgraph.random.HipGraphRandomState)

graph
[SGGraph] The input graph.

min_weight
: [double, optional (default=0.5)] The minimum value to assign as an edgeweight in the ECG algorithm. It should be a value in the range [0,1] usually left as the default value of .05

ensemble_size
: [size_t, optional (default=16)] The number of graph permutations to use for the ensemble. The default value is 16, larger values may produce higher quality partitions for some graphs.

max_level: size_t
: This controls the maximum number of levels/iterations of the leiden algorithm. When specified the algorithm will terminate after no more than the specified number of iterations. No error occurs when the algorithm terminates early in this manner.

threshold: float
: Modularity gain threshold for each level. If the gain of modularity between 2 levels of the algorithm is less than the given threshold then the algorithm stops and returns the resulting communities.

resolution: double
: Called gamma in the modularity formula, this changes the size of the communities. Higher resolutions lead to more smaller communities, lower resolutions lead to fewer larger communities. Defaults to 1.

do_expensive_check
: [bool_t] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.15.2 Returns

A tuple containing the hierarchical clustering vertices, clusters

3.15.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 0], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=True, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertices, clusters) = pylibhipgraph.ecg(resource_handle, G)
# FIXME: Check this docstring example
>>> vertices
[0, 1, 2]
>>> clusters
[0, 0, 0]
```

3.16 pylibhipgraph.edge_betweenness_centrality

edge_betweenness_centrality (*ResourceHandle resource_handle _GPUGraph graph, k random_state, bool_t normalized, bool_t do_expensive_check*)

Compute the edge betweenness centrality for all edges of the graph G. Betweenness centrality is a measure of the number of shortest paths that pass over an edge. An edge with a high betweenness centrality score has more paths passing over it and is therefore believed to be more important.

3.16.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

k
[int or device array type or None, optional (default=None)] If k is not None, use k node samples to estimate the edge betweenness. Higher values give better approximation. If k is a device array type, the contents are assumed to be vertex identifiers to be used for estimation. If k is None (the default), all the vertices are used to estimate the edge betweenness. Vertices obtained through sampling or defined as a list will be used as sources for traversals inside the algorithm.

random_state
[int, optional (default=None)] if k is specified and k is an integer, use random_state to initialize the random number generator. Using None defaults to a hash of process id, time, and hostname If k is either None or list or cudf objects: random_state parameter is ignored.

normalized
[bool_t] Normalization will ensure that values are in [0, 1].

do_expensive_check
[bool_t] A flag to run expensive checks for input arguments if True.

3.16.2 Returns

A tuple of device arrays corresponding to the sources, destinations, edge betweenness centrality scores and edge ids (if provided).

array containing the vertices and the second item in the tuple is a device array containing the eigenvector centrality scores for the corresponding vertices.

3.16.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 1, 2, 2, 2, 3, 4, 1, 3, 4, 0, 1, 3, 5, 5],
...     dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 3, 4, 0, 1, 3, 5, 5, 0, 1, 1, 2, 2, 2, 3, 4],
...     dtype=numpy.int32)
>>> edge_ids = cupy.asarray(
...     [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],
...     dtype=numpy.int32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, store_transposed=False,
...     renumber=False, do_expensive_check=False, edge_id_array=edge_ids)
>>> (srcs, dsts, values, edge_ids) = pylibhipgraph.edge_betweenness_centrality(
        resource_handle, G, None, None, True, False)
>>> srcs
[0 0 1 1 1 1 2 2 2 3 3 3 4 4 5 5]
>>> dsts
[1 2 0 2 3 4 0 1 3 1 2 5 1 5 3 4]
```

(continues on next page)

(continued from previous page)

```
>>> values
[0.10555556 0.06111111 0.10555556 0.06666667 0.09444445 0.14444445
 0.06111111 0.06666667 0.09444445 0.09444445 0.09444445 0.12222222
 0.14444445 0.07777778 0.12222222 0.07777778]
>>> edge_ids
[ 0 11  8 12  1  2  3  4  5  9 13  6 10  7 14 15]
```

3.17 pylibhipgraph.ego_graph

ego_graph (*ResourceHandle resource_handle, _GPUGraph graph, source_vertices, size_t radius, bool_t do_expensive_check*)

Compute the induced subgraph of neighbors centered at nodes source_vertices, within a given radius.

3.17.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph.

source_vertices
: [cupy array] The centered nodes from which the induced subgraph will be extracted

radius: size_t
: The number of hops to go out from each source vertex

do_expensive_check
: [bool_t] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.17.2 Returns

A tuple of device arrays containing the sources, destinations, edge_weights and the subgraph_offsets(if there are more than one seeds)

3.17.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 1, 2, 2, 2, 3, 3, 4], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 3, 4, 0, 1, 3, 4, 5, 5], dtype=numpy.int32)
>>> weights = cupy.asarray(
...     [0.1, 2.1, 1.1, 5.1, 3.1, 4.1, 7.2, 3.2, 6.1], dtype=numpy.float32)
>>> source_vertices = cupy.asarray([0, 1], dtype=numpy.int32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=False, renumber=False, do_expensive_check=False)
>>> (sources, destinations, edge_weights, subgraph_offsets) =
...     pylibhipgraph.ego_graph(resource_handle, G, source_vertices, 2, False)
```

(continues on next page)

(continued from previous page)

```
# FIXME: update results
>>> sources
[0, 1, 1, 3, 1, 1, 3, 3, 4]
>>> destinations
[1, 3, 4, 4, 3, 4, 4, 5, 5]
>>> edge_weights
[0.1, 2.1, 1.1, 7.2, 2.1, 1.1, 7.2, 3.2, 6.1]
>>> subgraph_offsets
[0, 4, 9]
```

3.18 pylibhipgraph.eigenvector_centrality

eigenvector_centrality (*ResourceHandle resource_handle, _GPUGraph graph, double epsilon, size_t max_iterations, bool_t do_expensive_check*)

Compute the Eigenvector centrality for the nodes of the graph.

3.18.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

epsilon
: [double] Error tolerance to check convergence

max_iterations: size_t
: Maximum number of Eignevector Centrality iterations

do_expensive_check
: [bool_t] A flag to run expensive checks for input arguments if True.

3.18.2 Returns

A tuple of device arrays, where the first item in the tuple is a device array containing the vertices and the second item in the tuple is a device array containing the eigenvector centrality scores for the corresponding vertices.

3.18.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 3], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertices, values) = pylibhipgraph.eigenvector_centrality(
                            resource_handle, G, 1e-6, 1000, False)
```

3.19 pylibhipgraph.generate_rmat_edgelist

generate_rmat_edgelist (*ResourceHandle resource_handle, random_state, size_t scale, size_t num_edges, double a, double b, double c, bool_t clip_and_flip, bool_t scramble_vertex_ids, bool_t include_edge_weights, minimum_weight, maximum_weight dtype, bool_t include_edge_ids, bool_t include_edge_types, min_edge_type_value, max_edge_type_value, bool_t multi_gpu*)

Generate RMAT edge list

3.19.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

random_state
: [int , optional] Random state to use when generating samples. Optional argument, defaults to a hash of process id, time, and hostname. (See pylibhipgraph.random.HipGraphRandomState)

scale
: [size_t] Scale factor to set the number of vertices in the graph Vertex IDs have values in [0, V), where V = 1 << `scale`

num_edges
: [size_t] Number of edges to generate

a
: [double] Probability of the edge being in the first partition The Graph 500 spec sets this value to 0.57

b
: [double] Probability of the edge being in the second partition The Graph 500 spec sets this value to 0.19

c
: [double] Probability of the edge being in the third partition The Graph 500 spec sets this value to 0.19

clip_and_flip
: [bool] Flag controlling whether to generate edges only in the lower triangular part (including the diagonal) of the graph adjacency matrix (if set to `true`) or not (if set to `false`).

scramble_vertex_ids
: [bool] Flag controlling whether to scramble vertex ID bits (if set to `true`) or not (if set to `false`); scrambling vertex ID bits breaks correlation between vertex ID values and vertex degrees.

include_edge_weights
: [bool] Flag controlling whether to generate edges with weights (if set to `true`) or not (if set to `false`).

minimum_weight
: [double] Minimum weight value to generate (if `include_edge_weights` is `true`)

maximum_weight
: [double] Maximum weight value to generate (if `include_edge_weights` is `true`)

dtype
: [string] The type of weight to generate (`FLOAT32` or `FLOAT64`), ignored unless `include_weights` is `true`

include_edge_ids
: [bool] Flag controlling whether to generate edges with ids (if set to `true`) or not (if set to `false`).

include_edge_types
: [bool] Flag controlling whether to generate edges with types (if set to `true`) or not (if set to `false`).

min_edge_type_value
: [int] Minimum edge type to generate if `include_edge_types` is `true` otherwise, this parameter is ignored.

max_edge_type_value
[int] Maximum edge type to generate if `include_edge_types` is `true` otherwise, this parameter is ignored.

multi_gpu
[bool] Flag if the COO is being created on multiple GPUs

3.19.2 Returns

Return a tuple containing the sources and destinations with their corresponding weights, ids and types if the flags `include_edge_weights`, `include_edge_ids` and `include_edge_types` are respectively set to `true`

3.20 pylibhipgraph.generate_rmat_edgelists

generate_rmat_edgelists (*ResourceHandle resource_handle, random_state, size_t n_edgelists, size_t min_scale, size_t max_scale, size_t edge_factor, size_distribution, edge_distribution, bool_t clip_and_flip, bool_t scramble_vertex_ids, bool_t include_edge_weights, minimum_weight, maximum_weight, dtype, bool_t include_edge_ids, bool_t include_edge_types, min_edge_type_value, max_edge_type_value, bool_t multi_gpu*)

Generate multiple RMAT edge lists

3.20.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

random_state
[int , optional] Random state to use when generating samples. Optional argument, defaults to a hash of process id, time, and hostname. (See pylibhipgraph.random.HipGraphRandomState)

n_edgelists
[size_t] Number of edge lists (graphs) to generate

min_scale
[size_t] Scale factor to set the minimum number of vertices in the graph

max_scale
[size_t] Scale factor to set the maximum number of vertices in the graph

edge_factor
[size_t] Average number of edges per vertex to generate

size_distribution
[int] Distribution of the graph sizes, impacts the scale parameter of the R-MAT generator. '0' for POWER_LAW distribution and '1' for UNIFORM distribution

edge_distribution
[int] Edges distribution for each graph, impacts how R-MAT parameters a,b,c,d, are set. '0' for POWER_LAW distribution and '1' for UNIFORM distribution

clip_and_flip
[bool] Flag controlling whether to generate edges only in the lower triangular part (including the diagonal) of the graph adjacency matrix (if set to 'true') or not (if set to 'false')

scramble_vertex_ids
[bool] Flag controlling whether to scramble vertex ID bits (if set to *true*) or not (if set to *false*); scrambling vertex ID bits breaks correlation between vertex ID values and vertex degrees.

include_edge_weights
[bool] Flag controlling whether to generate edges with weights (if set to 'true') or not (if set to 'false').

minimum_weight
[double] Minimum weight value to generate (if 'include_edge_weights' is 'true')

maximum_weight
[double] Maximum weight value to generate (if 'include_edge_weights' is 'true')

dtype
[string] The type of weight to generate ("FLOAT32" or "FLOAT64"), ignored unless include_weights is true

include_edge_ids
[bool] Flag controlling whether to generate edges with ids (if set to 'true') or not (if set to 'false').

include_edge_types
[bool] Flag controlling whether to generate edges with types (if set to 'true') or not (if set to 'false').

min_edge_type_value
[int] Minimum edge type to generate if 'include_edge_types' is 'true' otherwise, this parameter is ignored.

max_edge_type_value
[int] Maximum edge type to generate if 'include_edge_types' is 'true' otherwise, this parameter is ignored.

3.20.2 Returns

return a list of tuple containing the sources and destinations with their corresponding weights, ids and types if the flags 'include_edge_weights', 'include_edge_ids' and 'include_edge_types' are respectively set to 'true'

3.21 pylibhipgraph.GraphProperties

class GraphProperties

Class wrapper around C hipgraph_graph_properties_t struct

3.22 pylibhipgraph.SGGraph

class SGGraph (*_GPUGraph*)

RAII-stye Graph class for use with single-GPU APIs that manages the individual create/free calls and the corresponding hipgraph_graph_t pointer.

3.22.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph_properties
[GraphProperties] Object defining intended properties for the graph.

src_or_offset_array
[device array type] Device array containing either the vertex identifiers of the source of each directed edge if represented in COO format or the offset if CSR format. In the case of a COO, the order of the array corresponds to the ordering of the dst_or_index_array, where the ith item in src_offset_array and the ith item in dst_index_array define the ith edge of the graph.

dst_or_index_array
[device array type] Device array containing the vertex identifiers of the destination of each directed edge if represented in COO format or the index if CSR format. In the case of a COO, The order of the array corresponds to the ordering of the src_offset_array, where the ith item in src_offset_array and the ith item in dst_index_array define the ith edge of the graph.

vertices_array
: [device array type] Device array containing all vertices of the graph. This array is optional, but must be used if the graph contains isolated vertices which cannot be represented in the src_or_offset_array and dst_index_array arrays. If specified, this array must contain every vertex identifier, including vertex identifiers that are already included in the src_or_offset_array and dst_index_array arrays.

weight_array
: [device array type] Device array containing the weight values of each directed edge. The order of the array corresponds to the ordering of the src_array and dst_array arrays, where the ith item in weight_array is the weight value of the ith edge of the graph.

store_transposed
: [bool] Set to True if the graph should be transposed. This is required for some algorithms, such as pagerank.

renumber
: [bool] Set to True to indicate the vertices used in src_array and dst_array are not appropriate for use as internal array indices, and should be mapped to continuous integers starting from 0.

do_expensive_check
: [bool] If True, performs more extensive tests on the inputs to ensure validity, at the expense of increased run time.

edge_id_array
: [device array type] Device array containing the edge ids of each directed edge. Must match the ordering of the src/dst arrays. Optional (may be null). If provided, edge_type_array must also be provided.

edge_type_array
: [device array type] Device array containing the edge types of each directed edge. Must match the ordering of the src/dst/edge_id arrays. Optional (may be null). If provided, edge_id_array must be provided.

input_array_format: str, optional (default='COO')

: **Input representation used to construct a graph**
  COO: arrays represent src_array and dst_array CSR: arrays represent offset_array and index_array

drop_self_loops
: [bool, optional (default='False')] If true, drop any self loops that exist in the provided edge list.

drop_multi_edges: bool, optional (default='False')
: If true, drop any multi edges that exist in the provided edge list

3.22.2 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 3], dtype=numpy.int32)
>>> seeds = cupy.asarray([0, 0, 1], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=False, renumber=False, do_expensive_check=False)
```

3.23 pylibhipgraph.hits

hits (*ResourceHandle resource_handle, _GPUGraph graph, double tol, size_t max_iter, initial_hubs_guess_vertices, initial_hubs_guess_values, bool_t normalized, bool_t do_expensive_check*)

Compute HITS hubs and authorities values for each vertex

The HITS algorithm computes two numbers for a node. Authorities estimates the node value based on the incoming links. Hubs estimates the node value based on outgoing links.

3.23.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

tol
: [float, optional (default=1.0e-5)] Set the tolerance the approximation, this parameter should be a small magnitude value. This parameter is not currently supported.

max_iter
: [int, optional (default=100)] The maximum number of iterations before an answer is returned.

initial_hubs_guess_vertices
: [device array type, optional (default=None)] Device array containing the pointer to the array of initial hub guess vertices

initial_hubs_guess_values
: [device array type, optional (default=None)] Device array containing the pointer to the array of initial hub guess values

normalized : bool, optional (default=True)

do_expensive_check
: [bool] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.23.2 Returns

A tuple of device arrays, where the third item in the tuple is a device array containing the vertex identifiers, the first and second items are device arrays containing respectively the hubs and authorities values for the corresponding vertices

3.24 pylibhipgraph.induced_subgraph

induced_subgraph (*ResourceHandle resource_handle, _GPUGraph graph, subgraph_vertices, subgraph_offsets, bool_t do_expensive_check*)

extract a list of edges that represent the subgraph containing only the specified vertex ids.

3.24.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph.

subgraph_vertices
[cupy array] array of vertices to include in extracted subgraph.

subgraph_offsets
[cupy array] array of subgraph offsets into subgraph_vertices.

do_expensive_check
[bool_t] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.24.2 Returns

A tuple of device arrays containing the sources, destinations, edge_weights and the subgraph_offsets(if there are more than one seeds)

3.24.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 1, 2, 2, 2, 3, 4], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 3, 4, 0, 1, 3, 5, 5], dtype=numpy.int32)
>>> weights = cupy.asarray(
...     [0.1, 2.1, 1.1, 5.1, 3.1, 4.1, 7.2, 3.2], dtype=numpy.float32)
>>> subgraph_vertices = cupy.asarray([0, 1, 2, 3], dtype=numpy.int32)
>>> subgraph_offsets = cupy.asarray([0, 4], dtype=numpy.int32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=False, renumber=False, do_expensive_check=False)
>>> (sources, destinations, edge_weights, subgraph_offsets) =
...     pylibhipgraph.induced_subgraph(
...         resource_handle, G, subgraph_vertices, subgraph_offsets, False)
>>> sources
[0, 1, 2, 2, 2]
>>> destinations
[1, 3, 0, 1, 3]
>>> edge_weights
[0.1, 2.1, 5.1, 3.1, 4.1]
>>> subgraph_offsets
[0, 5]
```

3.25 pylibhipgraph.jaccard_coefficients

jaccard_coefficients (*ResourceHandle resource_handle, _GPUGraph graph, first, second, bool_t use_weight, bool_t do_expensive_check*)

Compute the Jaccard coefficients for the specified vertex_pairs.

> **Note**
>
> Jaccard similarity must run on a symmetric graph.

3.25.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

first :
Source of the vertex pair.

second :
Destination of the vertex pair.

use_weight
[bool, optional] If set to True, compute the weighted jaccard_coefficients (the input graph must be weighted in that case). Otherwise, compute the un-weighted jaccard_coefficients.

do_expensive_check
[bool] If True, performs more extensive tests on the inputs to ensure validity, at the expense of increased run time.

3.25.2 Returns

A tuple of device arrays containing the vertex pairs with their corresponding Jaccard coefficient scores.

3.26 pylibhipgraph.k_core

k_core (*ResourceHandle resource_handle, _GPUGraph graph, size_t k, degree_type, core_result, bool_t do_expensive_check*)

Compute the k-core of the graph G A k-core of a graph is a maximal subgraph that contains nodes of degree k or more. This call does not support a graph with self-loops and parallel edges.

3.26.1 Parameters

resource_handle: ResourceHandle
Handle to the underlying device and host resource needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

k
[size_t (default=None)] Order of the core. This value must not be negative. If set to None the main core is returned.

degree_type: str
This option determines if the core number computation should be based on input, output, or both directed edges, with valid values being “incoming”, “outgoing”, and “bidirectional” respectively. This option is currently ignored in this release, and setting it will result in a warning.

core_result
[device array type] Precomputed core number of the nodes of the graph G If set to None, the core numbers of the nodes are calculated internally.

do_expensive_check: bool
If True, performs more extensive tests on the inputs to ensure validity, at the expense of increased run time.

3.26.2 Returns

A tuple of device arrays contaning the sources, destinations vertices and the weights.

3.27 pylibhipgraph.k_truss_subgraph

k_truss_subgraph (*ResourceHandle resource_handle, _GPUGraph graph, size_t k, bool_t do_expensive_check*)

Extract k truss of a graph for a specific k.

3.27.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph.

k: size_t
: The desired k to be used for extracting the k-truss subgraph.

do_expensive_check
: [bool_t] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.27.2 Returns

A tuple of device arrays containing the sources, destinations, edge_weights and edge_offsets.

3.27.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 1, 3, 1, 4, 2, 0, 2, 1, 2,
...     3, 3, 4, 3, 5, 4, 5], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 0, 3, 1, 4, 1, 0, 2, 1, 2, 3,
...     2, 4, 3, 5, 3, 5, 4], dtype=numpy.int32)
>>> weights = cupy.asarray(
...     [0.1, 0.1, 2.1, 2.1, 1.1, 1.1, 7.2, 7.2, 2.1, 2.1,
...     1.1, 1.1, 7.2, 7.2, 3.2, 3.2, 6.1, 6.1]
...     ,dtype=numpy.float32)
>>> k = 2
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=True, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=False, renumber=False, do_expensive_check=False)
>>> (sources, destinations, edge_weights, subgraph_offsets) =
...     pylibhipgraph.k_truss_subgraph(resource_handle, G, k, False)
>>> sources
[0 0 1 1 1 1 2 2 2 3 3 3 3 4 4 4 5 5]
>>> destinations
[1 2 0 2 3 4 0 1 3 1 2 4 5 1 3 5 3 4]
>>> edge_weights
[0.1 7.2 0.1 2.1 2.1 1.1 7.2 2.1 1.1 2.1 1.1 7.2 3.2 1.1 7.2 6.1 3.2 6.1]
```

(continues on next page)

(continued from previous page)

```
>>> subgraph_offsets
[0 18]
```

3.28 pylibhipgraph.katz_centrality

katz_centrality (*ResourceHandle resource_handle, _GPUGraph graph, betas, double alpha, double beta, double epsilon, size_t max_iterations, bool_t do_expensive_check*)

Compute the Katz centrality for the nodes of the graph. This implementation is based on a relaxed version of Katz defined by Foster with a reduced computational complexity of O(n+m)

3.28.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

betas
: [device array type] Device array containing the values to be added to each vertex's new Katz Centrality score in every iteration. If set to None then beta is used for all vertices.

alpha
: [double] The attenuation factor, should be smaller than the inverse of the maximum eigenvalue of the graph

beta
: [double] Constant value to be added to each vertex's new Katz Centrality score in every iteration. Relevant only when betas is None

epsilon
: [double] Error tolerance to check convergence

max_iterations: size_t
: Maximum number of Katz Centrality iterations

do_expensive_check
: [bool_t] A flag to run expensive checks for input arguments if True.

3.29 pylibhipgraph.leiden

leiden (*ResourceHandle resource_handle, random_state, _GPUGraph graph, size_t max_level, double resolution, double theta, bool_t do_expensive_check*)

Compute the modularity optimizing partition of the input graph using the Leiden method.

3.29.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

random_state
: [int , optional] Random state to use when generating samples. Optional argument, defaults to a hash of process id, time, and hostname. (See pylibhipgraph.random.HipGraphRandomState)

graph
[SGGraph or MGGraph] The input graph.

max_level: size_t
This controls the maximum number of levels/iterations of the leiden algorithm. When specified the algorithm will terminate after no more than the specified number of iterations. No error occurs when the algorithm terminates early in this manner.

resolution: double
Called gamma in the modularity formula, this changes the size of the communities. Higher resolutions lead to more smaller communities, lower resolutions lead to fewer larger communities. Defaults to 1.

theta: double
Called theta in the Leiden algorithm, this is used to scale modularity gain in Leiden refinement phase, to compute the probability of joining a random leiden community.

do_expensive_check
[bool_t] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.29.2 Returns

A tuple containing the hierarchical clustering vertices, clusters and modularity score

3.29.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 0], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=True, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertices, clusters, modularity) = pylibhipgraph.Leiden(
                            resource_handle, G, 100, 1., False)
>>> vertices
[0, 1, 2]
>>> clusters
[0, 0, 0]
>>> modularity
0.0
```

3.30 pylibhipgraph.louvain

louvain (*ResourceHandle resource_handle, _GPUGraph graph, size_t max_level, float threshold, float resolution, bool_t do_expensive_check*)

Compute the modularity optimizing partition of the input graph using the Louvain method.

3.30.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph.

max_level: size_t
This controls the maximum number of levels/iterations of the Louvain algorithm. When specified the algorithm will terminate after no more than the specified number of iterations. No error occurs when the algorithm terminates early in this manner.

threshold: float
Modularity gain threshold for each level. If the gain of modularity between 2 levels of the algorithm is less than the given threshold then the algorithm stops and returns the resulting communities.

resolution: float
Called gamma in the modularity formula, this changes the size of the communities. Higher resolutions lead to more smaller communities, lower resolutions lead to fewer larger communities.

do_expensive_check
[bool_t] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.30.2 Returns

A tuple containing the hierarchical clustering vertices, clusters and modularity score

3.30.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 0], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=True, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertices, clusters, modularity) = pylibhipgraph.louvain(
                            resource_handle, G, 100, 1e-7, 1., False)
>>> vertices
[0, 1, 2]
>>> clusters
[0, 0, 0]
>>> modularity
0.0
```

3.31 pylibhipgraph.node2vec

node2vec (*ResourceHandle resource_handle, _GPUGraph graph, seed_array, size_t max_depth, bool_t compress_result, double p, double q*)

Computes random walks under node2vec sampling procedure.

3.31.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph] The input graph.

seed_array: device array type
Device array containing the pointer to the array of seed vertices.

max_depth
[size_t] Maximum number of vertices in generated path

compress_result
[bool_t] If true, the paths are unpadded and a third return device array contains the sizes for each path, otherwise the paths are padded and the third return device array is empty.

p
[double] The return factor p represents the likelihood of backtracking to a node in the walk. A higher value (> max(q, 1)) makes it less likely to sample a previously visited node, while a lower value (< min(q, 1)) would make it more likely to backtrack, making the walk more "local".

q
[double] The in-out factor q represents the likelihood of visiting nodes closer or further from the outgoing node. If q > 1, the random walk is likelier to visit nodes closer to the outgoing node. If q < 1, the random walk is likelier to visit nodes further from the outgoing node.

3.31.2 Returns

A tuple of device arrays, where the first item in the tuple is a device array containing the compressed paths, the second item is a device array containing the corresponding weights for each edge traversed in each path, and the third item is a device array containing the sizes for each of the compressed paths, if compress_result is True.

3.31.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 3], dtype=numpy.int32)
>>> seeds = cupy.asarray([0, 0, 1], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=False, renumber=False, do_expensive_check=False)
>>> (paths, weights, sizes) = pylibhipgraph.node2vec(
...                               resource_handle, G, seeds, 3, True, 1.0, 1.0)
```

3.32 pylibhipgraph.overlap_coefficients

overlap_coefficients (*ResourceHandle resource_handle, _GPUGraph graph, first, second, bool_t use_weight, bool_t do_expensive_check*)

Compute the Overlap coefficients for the specified vertex_pairs.

> **Note**
>
> Overlap similarity must run on a symmetric graph.

3.32.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

first :
: Source of the vertex pair.

second :
: Destination of the vertex pair.

use_weight
: [bool, optional] If set to True, compute the weighted jaccard_coefficients (the input graph must be weighted in that case). Otherwise, compute the un-weighted jaccard_coefficients.

do_expensive_check
: [bool] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.32.2 Returns

A tuple of device arrays containing the vertex pairs with their corresponding Overlap coefficient scores.

3.33 pylibhipgraph.pagerank

pagerank (*ResourceHandle resource_handle, _GPUGraph graph, precomputed_vertex_out_weight_vertices, precomputed_vertex_out_weight_sums, initial_guess_vertices, initial_guess_values, double alpha, double epsilon, size_t max_iterations, bool_t do_expensive_check, fail_on_nonconvergence=True*)

Find the PageRank score for every vertex in a graph by computing an approximation of the Pagerank eigenvector using the power method. The number of iterations depends on the properties of the network itself; it increases when the tolerance descreases and/or alpha increases toward the limiting value of 1.

3.33.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

precomputed_vertex_out_weight_vertices: device array type
: Subset of vertices of graph for precomputed_vertex_out_weight (a performance optimization)

precomputed_vertex_out_weight_sums
: [device array type] Corresponding precomputed sum of outgoing vertices weight (a performance optimization)

initial_guess_vertices
: [device array type] Subset of vertices of graph for initial guess for pagerank values (a performance optimization)

initial_guess_values
[device array type] Pagerank values for vertices (a performance optimization)

alpha
[double] The damping factor alpha represents the probability to follow an outgoing edge, standard value is 0.85. Thus, 1.0-alpha is the probability to "teleport" to a random vertex. Alpha should be greater than 0.0 and strictly lower than 1.0.

epsilon
[double] Set the tolerance the approximation, this parameter should be a small magnitude value. The lower the tolerance the better the approximation. If this value is 0.0f, hipGRAPH will use the default value which is 1.0E-5. Setting too small a tolerance can lead to non-convergence due to numerical roundoff. Usually values between 0.01 and 0.00001 are acceptable.

max_iterations
[size_t] The maximum number of iterations before an answer is returned. This can be used to limit the execution time and do an early exit before the solver reaches the convergence tolerance. If this value is lower or equal to 0 hipGRAPH will use the default value, which is 100.

do_expensive_check
[bool_t] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

fail_on_nonconvergence
[bool (default=True)] If the solver does not reach convergence, raise an exception if fail_on_nonconvergence is True. If fail_on_nonconvergence is False, the return value is a tuple of (pagerank, converged) where pagerank is a cudf.DataFrame as described below, and converged is a boolean indicating if the solver converged (True) or not (False).

3.33.2 Returns

The return value varies based on the value of the fail_on_nonconvergence paramter. If fail_on_nonconvergence is True:

> A tuple of device arrays, where the first item in the tuple is a device array containing the vertex identifiers, and the second item is a device array containing the pagerank values for the corresponding vertices. For example, the vertex identifier at the ith element of the vertex array has the pagerank value of the ith element in the pagerank array.

If fail_on_nonconvergence is False:

> A three-tuple where the first two items are the device arrays described above, and the third is a bool indicating if the solver converged (True) or not (False).

3.33.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 3], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertices, pageranks) = pylibhipgraph.pagerank(
...     resource_handle, G, None, None, None, None, alpha=0.85,
```

(continues on next page)

(continued from previous page)

```
...     epsilon=1.0e-6, max_iterations=500, do_expensive_check=False)
>>> vertices
array([0, 1, 2, 3], dtype=int32)
>>> pageranks
array([0.11615585, 0.21488841, 0.2988108 , 0.3701449 ], dtype=float32)
```

3.34 pylibhipgraph.personalized_pagerank

personalized_pagerank (*ResourceHandle resource_handle, _GPUGraph graph, precomputed_vertex_out_weight_vertices, precomputed_vertex_out_weight_sums, initial_guess_vertices, initial_guess_values, personalization_vertices, personalization_values, double alpha, double epsilon, size_t max_iterations, bool_t do_expensive_check, fail_on_nonconvergence=True*)

Find the PageRank score for every vertex in a graph by computing an approximation of the Pagerank eigenvector using the power method. The number of iterations depends on the properties of the network itself; it increases when the tolerance descreases and/or alpha increases toward the limiting value of 1.

3.34.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

precomputed_vertex_out_weight_vertices: device array type
: Subset of vertices of graph for precomputed_vertex_out_weight

precomputed_vertex_out_weight_sums
: [device array type] Corresponding precomputed sum of outgoing vertices weight

initial_guess_vertices
: [device array type] Subset of vertices of graph for initial guess for pagerank values

initial_guess_values
: [device array type] Pagerank values for vertices

personalization_vertices
: [device array type] Subset of vertices of graph for personalization

personalization_values
: [device array type] Personalization values for vertices

alpha
: [double] The damping factor alpha represents the probability to follow an outgoing edge, standard value is 0.85. Thus, 1.0-alpha is the probability to "teleport" to a random vertex. Alpha should be greater than 0.0 and strictly lower than 1.0.

epsilon
: [double] Set the tolerance the approximation, this parameter should be a small magnitude value. The lower the tolerance the better the approximation. If this value is 0.0f, hipGRAPH will use the default value which is 1.0E-5. Setting too small a tolerance can lead to non-convergence due to numerical roundoff. Usually values between 0.01 and 0.00001 are acceptable.

max_iterations
: [size_t] The maximum number of iterations before an answer is returned. This can be used to limit the execution

time and do an early exit before the solver reaches the convergence tolerance. If this value is lower or equal to 0 hipGRAPH will use the default value, which is 100.

do_expensive_check
[bool_t] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

fail_on_nonconvergence
[bool (default=True)] If the solver does not reach convergence, raise an exception if fail_on_nonconvergence is True. If fail_on_nonconvergence is False, the return value is a tuple of (pagerank, converged) where pagerank is a cudf.DataFrame as described below, and converged is a boolean indicating if the solver converged (True) or not (False).

3.34.2 Returns

The return value varies based on the value of the fail_on_nonconvergence paramter. If fail_on_nonconvergence is True:

A tuple of device arrays, where the first item in the tuple is a device array containing the vertex identifiers, and the second item is a device array containing the pagerank values for the corresponding vertices. For example, the vertex identifier at the ith element of the vertex array has the pagerank value of the ith element in the pagerank array.

If fail_on_nonconvergence is False:

A three-tuple where the first two items are the device arrays described above, and the third is a bool indicating if the solver converged (True) or not (False).

3.34.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 3], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> personalization_vertices = cupy.asarray([0, 2], dtype=numpy.int32)
>>> personalization_values = cupy.asarray(
...     [0.008309, 0.991691], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertices, pageranks) = pylibhipgraph.personalized_pagerank(
...     resource_handle, G, None, None, None, None, alpha=0.85,
...     personalization_vertices=personalization_vertices,
...     personalization_values=personalization_values, epsilon=1.0e-6,
...     max_iterations=500,
...     do_expensive_check=False)
>>> vertices
array([0, 1, 2, 3], dtype=int32)
>>> pageranks
array([0.00446455, 0.00379487, 0.53607565, 0.45566472 ], dtype=float32)
```

3.35 pylibhipgraph.HipGraphRandomState

class HipGraphRandomState

This class wraps a `hipgraph_rng_state_t` instance, which represents a random state.

__cinit__ (*self, ResourceHandle resource_handle, seed=None*)

Constructs a new HipGraphRandomState instance.

3.35.1 Parameters

resource_handle: pylibhipgraph.ResourceHandle (Required)
The hipgraph resource handle for this process.

seed: int (Optional)
The random seed of this random state object. Defaults to the hash of the hostname, pid, and time.

__dealloc__(*self*)

Destroys this HipGraphRandomState instance. Properly calls free to destroy the underlying C++ object.

3.36 pylibhipgraph.replicate_edgelist

replicate_edgelist (*ResourceHandle resource_handle, src_array, dst_array, weight_array, edge_id_array, edge_type_id_array*)

Replicate edges across all GPUs

3.36.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

src_array
[device array type, optional] Device array containing the vertex identifiers of the source of each directed edge. The order of the array corresponds to the ordering of the `dst_array`, where the ith item in `src_array` and the ith item in `dst_array` define the ith edge of the graph.

dst_array
[device array type, optional] Device array containing the vertex identifiers of the destination of each directed edge. The order of the array corresponds to the ordering of the `src_array`, where the ith item in `src_array` and the ith item in `dst_array` define the ith edge of the graph.

weight_array
[device array type, optional] Device array containing the weight values of each directed edge. The order of the array corresponds to the ordering of the `src_array` and `dst_array` arrays, where the ith item in `weight_array` is the weight value of the ith edge of the graph.

edge_id_array
[device array type, optional] Device array containing the edge id values of each directed edge. The order of the array corresponds to the ordering of the `src_array` and `dst_array` arrays, where the ith item in `edge_id_array` is the id value of the ith edge of the graph.

edge_type_id_array
[device array type, optional] Device array containing the edge type id values of each directed edge. The order of the array corresponds to the ordering of the `src_array` and `dst_array` arrays, where the ith item in `edge_type_id_array` is the type id value of the ith edge of the graph.

3.36.2 Returns

return cupy arrays of `src` and/or `dst` and/or `weight` and/or `edge_id` and/or `edge_type_id`.

3.37 pylibhipgraph.ResourceHandle

class ResourceHandle

RAII-stye resource handle class to manage individual create/free calls and the corresponding pointer to a hipgraph_resource_handle_t

__cinit__(self, handle=None)

cdef void* handle_ptr = NULL cdef size_t handle_size_t if handle is not None:

> # FIXME: rather than assume a RAFT handle here, consider something # like a factory function in hipgraph (which already has a RAFT # dependency and makes RAFT assumptions) that takes a RAFT handle # and constructs/returns a ResourceHandle handle_size_t = <size_t>handle handle_ptr = <void *> handle_size_t

self.c_resource_handle_ptr[0] = hipgraph_create_resource_handle(handle_ptr)[0] # FIXME: check for error

def __dealloc__(self)

FIXME: free only if handle is a valid pointer hipgraph_free_resource_handle(self.c_resource_handle_ptr)

3.38 pylibhipgraph.select_random_vertices

select_random_vertices (*ResourceHandle resource_handle, _GPUGraph graph, random_state, size_t num_vertices*)

Select random vertices from the graph

3.38.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

random_state
: [int , optional] Random state to use when generating samples. Optional argument, defaults to a hash of process id, time, and hostname. (See pylibhipgraph.random.HipGraphRandomState)

num_vertices
: [size_t , optional] Number of vertices to sample. Optional argument, defaults to the total number of vertices.

3.38.2 Returns

return random vertices from the graph

3.39 pylibhipgraph.sorensen_coefficients

sorensen_coefficients (*ResourceHandle resource_handle, _GPUGraph graph, first, second, bool_t use_weight, bool_t do_expensive_check*)

Compute the Sorensen coefficients for the specified vertex_pairs.

Note

Sorensen similarity must run on a symmetric graph.

3.39.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

first :
Source of the vertex pair.

second :
Destination of the vertex pair.

use_weight
[bool, optional] If set to True, compute the weighted jaccard_coefficients (the input graph must be weighted in that case). Otherwise, compute the un-weighted jaccard_coefficients.

do_expensive_check
[bool] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.39.2 Returns

A tuple of device arrays containing the vertex pairs with their corresponding Sorensen coefficient scores.

3.40 pylibhipgraph.spectral_modularity_maximization

spectral_modularity_maximization (*ResourceHandle resource_handle, _GPUGraph graph, num_clusters, num_eigen_vects, evs_tolerance, evs_max_iter, kmean_tolerance, kmean_max_iter, bool_t do_expensive_check*)

Compute a clustering/partitioning of the given graph using the spectral modularity maximization method.

3.40.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph] The input graph.

num_clusters
[size_t] Specifies the number of clusters to find, must be greater than 1

num_eigen_vects
[size_t] Specifies the number of eigenvectors to use. Must be lower or equal to num_clusters.

evs_tolerance: double
Specifies the tolerance to use in the eigensolver.

evs_max_iter: size_t
Specifies the maximum number of iterations for the eigensolver.

kmean_tolerance: double
Specifies the tolerance to use in the k-means solver.

kmean_max_iter: size_t
Specifies the maximum number of iterations for the k-means solver.

do_expensive_check
[bool_t] If True, performs more extensive tests on the inputs to ensure validity, at the expense of increased run time.

3.40.2 Returns

A tuple containing the clustering vertices, clusters

3.40.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 0], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=True, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=True, renumber=False, do_expensive_check=False)
>>> (vertices, clusters) = pylibhipgraph.spectral_modularity_maximization(
...     resource_handle, G, num_clusters=5, num_eigen_vects=2, evs_tolerance=0.00001
...     evs_max_iter=100, kmean_tolerance=0.00001, kmean_max_iter=100)
# FIXME: Fix docstring result.
>>> vertices
############
>>> clusters
############
```

3.41 pylibhipgraph.sssp

sssp (*ResourceHandle resource_handle, _GPUGraph graph, size_t source, double cutoff, bool_t compute_predecessors, bool_t do_expensive_check*)

Compute the distance and predecessors for shortest paths from the specified source to all the vertices in the graph. The returned distances array will contain the distance from the source to each vertex in the returned vertex array at the same index. The returned predecessors array will contain the previous vertex in the shortest path for each vertex in the vertex array at the same index. Vertices that are unreachable will have a distance of infinity denoted by the maximum value of the data type and the predecessor set as -1. The source vertex predecessor will be set to -1. Graphs with negative weight cycles are not supported.

3.41.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

source :
The vertex identifier of the source vertex.

cutoff :
Maximum edge weight sum to consider.

compute_predecessors
[bool] This parameter must be set to True for this release.

do_expensive_check
[bool] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.41.2 Returns

A 3-tuple, where the first item in the tuple is a device array containing the vertex identifiers, the second item is a device array containing the distance for each vertex from the source vertex, and the third item is a device array containing the vertex identifier of the preceding vertex in the path for that vertex. For example, the vertex identifier at the ith element of the vertex array has a distance from the source vertex of the ith element in the distance array, and the preceding vertex in the path is the ith element in the predecessor array.

3.41.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> srcs = cupy.asarray([0, 1, 2], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 2, 3], dtype=numpy.int32)
>>> weights = cupy.asarray([1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...     is_symmetric=False, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=False, renumber=False, do_expensive_check=False)
>>> (vertices, distances, predecessors) = pylibhipgraph.sssp(
...     resource_handle, G, source=1, cutoff=999,
...     compute_predecessors=True, do_expensive_check=False)
>>> vertices
array([0, 1, 2, 3], dtype=int32)
>>> distances
array([3.4028235e+38, 0.0000000e+00, 1.0000000e+00, 2.0000000e+00],
      dtype=float32)
>>> predecessors
array([-1, -1,  1,  2], dtype=int32)
```

3.42 pylibhipgraph.triangle_count

triangle_count (*ResourceHandle resource_handle, _GPUGraph graph, start_list, bool_t do_expensive_check*)

Computes the number of triangles (cycles of length three) and the number per vertex in the input graph.

3.42.1 Parameters

resource_handle: ResourceHandle
Handle to the underlying device and host resources needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

start_list: device array type
Device array containing the list of vertices for triangle counting. If 'None' the entire set of vertices in the graph is processed

do_expensive_check: bool
If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.42.2 Returns

A tuple of device arrays, where the first item in the tuple is a device array containing the vertex identifiers and the second item contains the triangle counting counts

3.42.3 Examples

FIXME: No example yet

3.43 pylibhipgraph.get_two_hop_neighbors

get_two_hop_neighbors (*ResourceHandle resource_handle, _GPUGraph graph, start_vertices, bool_t do_expensive_check*)

Compute vertex pairs that are two hops apart. The resulting pairs are sorted before returning.

3.43.1 Parameters

resource_handle
[ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

start_vertices
[Optional array of starting vertices] If None use all, if specified compute two-hop neighbors for these starting vertices

3.43.2 Returns

return a cupy arrays of 'first' and 'second' or a 'hipgraph_vertex_pairs_t' which can be directly passed to the similarity algorithm?

3.44 pylibhipgraph.uniform_neighbor_sample

uniform_neighbor_sample (ResourceHandle resource_handle, _GPUGraph input_graph, start_list, h_fan_out, *, bool_t with_replacement, bool_t do_expensive_check, with_edge_properties=False, batch_id_list=None, label_list=None, label_to_output_comm_rank=None, label_offsets=None, prior_sources_behavior=None, deduplicate_sources=False, return_hops=False, renumber=False, retain_seeds=False, compression='COO', compress_per_hop=False, random_state=None, return_dict=False)

Does neighborhood sampling, which samples nodes from a graph based on the current node's neighbors, with a corresponding fanout value at each hop.

3.44.1 Parameters

resource_handle: ResourceHandle
Handle to the underlying device and host resources needed for referencing data and running algorithms.

input_graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

start_list: device array type
Device array containing the list of starting vertices for sampling.

h_fan_out: numpy array type
Device array containing the brancing out (fan-out) degrees per starting vertex for each hop level.

with_replacement: bool
If true, sampling procedure is done with replacement (the same vertex can be selected multiple times in the same step).

do_expensive_check: bool
If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

with_edge_properties: bool
If True, returns the edge properties of each edges along with the edges themselves. Will result in an error if the provided graph does not have edge properties.

batch_id_list: list[int32] (Optional)
List of int32 batch ids that is returned with each edge. Optional argument, defaults to NULL, returning nothing.

label_list: list[int32] (Optional)
List of unique int32 batch ids. Required if also passing the label_to_output_comm_rank flag. Default to NULL (does nothing)

label_to_output_comm_rank: list[int32] (Optional)
Maps the unique batch ids in label_list to the rank of the worker that should hold results for that batch id. Defaults to NULL (does nothing)

label_offsets: list[int] (Optional)
Offsets of each label within the start vertex list.

prior_sources_behavior: str (Optional)
Options are "carryover", and "exclude". Default will leave the source list as-is. Carryover will carry over sources from previous hops to the current hop. Exclude will exclude sources from previous hops from reappearing as sources in future hops.

deduplicate_sources: bool (Optional)
If True, will deduplicate the source list before sampling. Defaults to False.

renumber: bool (Optional)
If True, will renumber the sources and destinations on a per-batch basis and return the renumber map and batch offsets in additional to the standard returns.

retain_seeds: bool (Optional)
If True, will retain the original seeds (original source vertices) in the output even if they do not have outgoing neighbors. Defaults to False.

compression: str (Optional)
Options: COO (default), CSR, CSC, DCSR, DCSR Sets the compression format for the returned samples.

compress_per_hop: bool (Optional)
If False (default), will create a compressed edgelist for the entire batch. If True, will create a separate compressed edgelist per hop within a batch.

random_state: int (Optional)
Random state to use when generating samples. Optional argument, defaults to a hash of process id, time, and hostname. (See pylibhipgraph.random.HipGraphRandomState)

return_dict: bool (Optional)
Whether to return a dictionary instead of a tuple. Optional argument, defaults to False, returning a tuple. This argument will eventually be deprecated in favor of always returning a dictionary.

3.44.2 Returns

A tuple of device arrays, where the first and second items in the tuple are device arrays containing the starting and ending vertices of each walk respectively, the third item in the tuple is a device array containing the start labels, and the fourth item in the tuple is a device array containing the indices for reconstructing paths.

If renumber was set to True, then the fifth item in the tuple is a device array containing the renumber map, and the sixth item in the tuple is a device array containing the renumber map offsets (which delineate where the renumber map for each batch starts).

3.45 pylibhipgraph.uniform_random_walks

uniform_random_walks (*ResourceHandle resource_handle, _GPUGraph input_graph, start_vertices, size_t max_length*)

Compute uniform random walks for each nodes in 'start_vertices'

3.45.1 Parameters

resource_handle: ResourceHandle
Handle to the underlying device and host resources needed for referencing data and running algorithms.

input_graph
[SGGraph or MGGraph] The input graph, for either Single or Multi-GPU operations.

start_vertices: device array type
Device array containing the list of starting vertices from which to run the uniform random walk

max_length: size_t
The maximum depth of the uniform random walks

3.45.2 Returns

A tuple containing two device arrays and an size_t which are respectively the vertices path, the edge path weights and the maximum path length

3.46 pylibhipgraph.copy_to_cupy_array

copy_to_cupy_array (hipgraph_resource_handle_t* c_resource_handle_ptr, hipgraph_type_erased_device_array_view_t* device_array_view_ptr)

Copy the contents from a device array view as returned by various hipgraph_* APIs to a new cupy device array, typically intended to be used as a return value from pylibhipgraph APIs.

copy_to_cupy_array_ids (hipgraph_resource_handle_t* c_resource_handle_ptr, hipgraph_type_erased_device_array_view_t* device_array_view_ptr)

Copy the contents from a device array view as returned by various hipgraph_* APIs to a new cupy device array, typically intended to be used as a return value from pylibhipgraph APIs then convert float to int

3.47 pylibhipgraph.weakly_connected_components

weakly_connected_components (*ResourceHandle resource_handle, _GPUGraph graph, offsets, indices, weights, labels, bool_t do_expensive_check*)

Generate the Weakly Connected Components from either an input graph or or CSR arrays('offsets', 'indices', 'weights') and attach a component label to each vertex.

3.47.1 Parameters

resource_handle
: [ResourceHandle] Handle to the underlying device resources needed for referencing data and running algorithms.

graph
: [SGGraph or MGGraph] The input graph.

offsets
: [object supporting a __cuda_array_interface__ interface] Array containing the offsets values of a Compressed Sparse Row matrix that represents the graph.

indices
: [object supporting a __cuda_array_interface__ interface] Array containing the indices values of a Compressed Sparse Row matrix that represents the graph.

weights
: [object supporting a __cuda_array_interface__ interface] Array containing the weights values of a Compressed Sparse Row matrix that represents the graph

do_expensive_check
: [bool_t] If True, performs more extensive tests on the inputs to ensure validitity, at the expense of increased run time.

3.47.2 Returns

A tuple containing containing two device arrays which are respectively vertices and their corresponding labels

3.47.3 Examples

```
>>> import pylibhipgraph, cupy, numpy
>>> from pylibhipgraph import weakly_connected_components
>>> srcs = cupy.asarray([0, 1, 1, 2, 2, 0], dtype=numpy.int32)
>>> dsts = cupy.asarray([1, 0, 2, 1, 0, 2], dtype=numpy.int32)
>>> weights = cupy.asarray(
...     [1.0, 1.0, 1.0, 1.0, 1.0, 1.0], dtype=numpy.float32)
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> graph_props = pylibhipgraph.GraphProperties(
...      is_symmetric=True, is_multigraph=False)
>>> G = pylibhipgraph.SGGraph(
...     resource_handle, graph_props, srcs, dsts, weight_array=weights,
...     store_transposed=False, renumber=True, do_expensive_check=False)
```

(continues on next page)

(continued from previous page)

```
>>> (vertices, labels) = weakly_connected_components(
...     resource_handle, G, None, None, None, None, False)

>>> vertices
[0, 1, 2]
>>> labels
[2, 2, 2]

>>> import cupy as cp
>>> import numpy as np
>>> from scipy.sparse import csr_matrix
>>>
>>> graph = [
... [0, 1, 1, 0, 0],
... [0, 0, 1, 0, 0],
... [0, 0, 0, 0, 0],
... [0, 0, 0, 0, 1],
... [0, 0, 0, 0, 0],
... ]
>>> scipy_csr = csr_matrix(graph)
>>> rows, cols = scipy_csr.nonzero()
>>> scipy_csr[cols, rows] = scipy_csr[rows, cols]
>>>
>>> cp_offsets = cp.asarray(scipy_csr.indptr)
>>> cp_indices = cp.asarray(scipy_csr.indices, dtype=np.int32)
>>>
>>> resource_handle = pylibhipgraph.ResourceHandle()
>>> weakly_connected_components(resource_handle=resource_handle,
                                graph=None,
...                             offsets=cp_offsets,
...                             indices=cp_indices,
...                             weights=None,
...                             False)
>>> print(f"{len(set(cp_labels.tolist()))} - {cp_labels}")
2 - [2 2 2 4 4]
```

CHAPTER

FOUR

LICENSE

SPDX-FileCopyrightText: Modifications Copyright (c) 2025, Advanced Micro Devices, Inc. SPDX-License-Identifier: MIT SPDX-FileCopyrightText: Copyright (c) 2022-2024, NVIDIA CORPORATION. SPDX-License-Identifier: Apache-2.0

In the source code, license texts reside in the LICENSES subdirectory following the SPDX/REUSE convention. They are appended here for convenience.

MIT License

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Apache License Version 2.0, January 2004 http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

    (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

    (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

    (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

    (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that

You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

INDEX

H