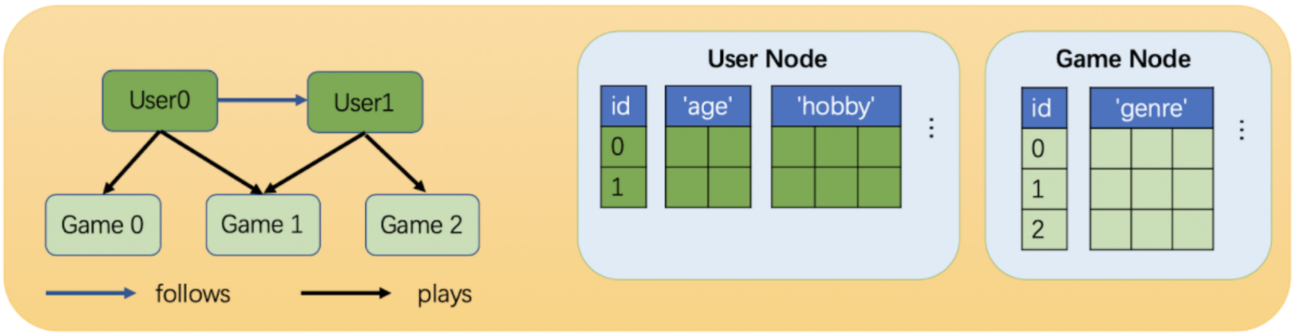
1.5 Heterogeneous Graphs

(中文版)

A heterogeneous graph can have nodes and edges of different types. Nodes/Edges of different types have independent ID space and feature storage. For example in the figure below, the user and game node IDs both start from zero and they have different features.



An example heterogeneous graph with two types of nodes (user and game) and two types of edges (follows and plays).

Creating a Heterogeneous Graph

In DGL, a heterogeneous graph (heterograph for short) is specified with a series of graphs as below, one per relation. Each relation is a string triplet `(source node type, edge type, destination node type)`. Since relations disambiguate the edge types, DGL calls them canonical edge types.

The following code snippet is an example for creating a heterogeneous graph in DGL.

```
>>> import dgl
>>> import torch as th

>>> # Create a heterograph with 3 node types and 3 edges types.
>>> graph_data = {
...    ('drug', 'interacts', 'drug'): (th.tensor([0, 1]), th.tensor([1, 2])),
...    ('drug', 'interacts', 'gene'): (th.tensor([0, 1]), th.tensor([2, 3])),
...    ('drug', 'treats', 'disease'): (th.tensor([1]), th.tensor([2]))
... }
>>> g = dgl.heterograph(graph_data)
>>> g.ntypes
['disease', 'drug', 'gene']
>>> g.etypes
['interacts', 'interacts', 'treats']
>>> g.canonical_etypes
[('drug', 'interacts', 'drug'),
 ('drug', 'interacts', 'gene'),
 ('drug', 'treats', 'disease')]
```

Note that homogeneous and bipartite graphs are just special heterogeneous graphs with one relation.

```
>>> # A homogeneous graph
>>> dgl.heterograph({('node_type', 'edge_type', 'node_type'): (u, v)})
>>> # A bipartite graph
>>> dgl.heterograph({('source_type', 'edge_type', 'destination_type'): (u, v)})
```

The *metagraph* associated with a heterogeneous graph is the schema of the graph. It specifies type constraints on the sets of nodes and edges between the nodes. A node u in a metagraph corresponds to a node type in the associated heterograph. An edge (u, v) in a metagraph indicates that there are edges from nodes of type u to nodes of type v in the associated heterograph.

```
>>> g
Graph(num_nodes={'disease': 3, 'drug': 3, 'gene': 4},
      num_edges={('drug', 'interacts', 'drug'): 2,
                 ('drug', 'interacts', 'gene'): 2,
                 ('drug', 'treats', 'disease'): 1},
      metagraph=[('drug', 'drug', 'interacts'),
                 ('drug', 'gene', 'interacts'),
                 ('drug', 'disease', 'treats')])
>>> g.metagraph().edges()
OutMultiEdgeDataView([('drug', 'drug'), ('drug', 'gene'), ('drug', 'disease')])
```

See APIs: `dgl.heterograph()`, `ntypes`, `etypes`, `canonical_etypes`, `metagraph`.

Working with Multiple Types

When multiple node/edge types are introduced, users need to specify the particular node/edge type when invoking a DGLGraph API for type-specific information. In addition, nodes/edges of different types have separate IDs.

```
>>> # Get the number of all nodes in the graph
>>> g.num_nodes()
10
>>> # Get the number of drug nodes
>>> g.num_nodes('drug')
3
>>> # Nodes of different types have separate IDs,
>>> # hence not well-defined without a type specified
>>> g.nodes()
DGLError: Node type name must be specified if there are more than one node types.
>>> g.nodes('drug')
tensor([0, 1, 2])
```

To set/get features for a specific node/edge type, DGL provides two new types of syntax – *g.nodes['node_type'].data['feat_name']* and *g.edges['edge_type'].data['feat_name']*.

```
>>> # Set/get feature 'hv' for nodes of type 'drug'
>>> g.nodes['drug'].data['hv'] = th.ones(3, 1)
>>> g.nodes['drug'].data['hv']
tensor([[1.],
        [1.],
        [1.]])
>>> # Set/get feature 'he' for edge of type 'treats'
>>> g.edges['treats'].data['he'] = th.zeros(1, 1)
>>> g.edges['treats'].data['he']
tensor([[0.]])
```

If the graph only has one node/edge type, there is no need to specify the node/edge type.

```
>>> g = dgl.heterograph({
...     ('drug', 'interacts', 'drug'): (th.tensor([0, 1]), th.tensor([1, 2])),
...     ('drug', 'is similar', 'drug'): (th.tensor([0, 1]), th.tensor([2, 3]))
... })
>>> g.nodes()
tensor([0, 1, 2, 3])
>>> # To set/get feature with a single type, no need to use the new syntax
>>> g.ndata['hv'] = th.ones(4, 1)
```

Note

When the edge type uniquely determines the types of source and destination nodes, one can just use one string instead of a string triplet to specify the edge type. For example, for a heterograph with two relations `('user', 'plays', 'game')` and `('user', 'likes', 'game')`, it is safe to just use `'plays'` or `'likes'` to refer to the two relations.

Loading Heterographs from Disk

Comma Separated Values (CSV)

A common way to store a heterograph is to store nodes and edges of different types in different CSV files. An example is as follows.

```
# data folder
data/
|-- drug.csv        # drug nodes
|-- gene.csv        # gene nodes
|-- disease.csv     # disease nodes
|-- drug-interact-drug.csv  # drug-drug interaction edges
|-- drug-interact-gene.csv  # drug-gene interaction edges
|-- drug-treat-disease.csv  # drug-treat-disease edges
```

Similar to the case of homogeneous graphs, one can use packages like Pandas to parse CSV files into numpy arrays or framework tensors, build a relation dictionary and construct a heterograph from that. The approach also applies to other popular formats like GML/JSON.

DGL Binary Format

DGL provides `dgl.save_graphs()` and `dgl.load_graphs()` respectively for saving heterogeneous graphs in binary format and loading them from binary format.

Edge Type Subgraph

One can create a subgraph of a heterogeneous graph by specifying the relations to retain, with features copied if any.

```
>>> g = dgl.heterograph({
...     ('drug', 'interacts', 'drug'): (th.tensor([0, 1]), th.tensor([1, 2])),
...     ('drug', 'interacts', 'gene'): (th.tensor([0, 1]), th.tensor([2, 3])),
...     ('drug', 'treats', 'disease'): (th.tensor([1]), th.tensor([2]))
... })
>>> g.nodes['drug'].data['hv'] = th.ones(3, 1)

>>> # Retain relations ('drug', 'interacts', 'drug') and ('drug', 'treats', 'disease')
>>> # All nodes for 'drug' and 'disease' will be retained
>>> eg = dgl.edge_type_subgraph(g, [('drug', 'interacts', 'drug'),
...                                 ('drug', 'treats', 'disease')])
>>> eg
Graph(num_nodes={'disease': 3, 'drug': 3},
      num_edges={('drug', 'interacts', 'drug'): 2, ('drug', 'treats', 'disease'): 1},
      metagraph=[('drug', 'drug', 'interacts'), ('drug', 'disease', 'treats')])
>>> # The associated features will be copied as well
>>> eg.nodes['drug'].data['hv']
tensor([[1.],
        [1.],
        [1.]])
```

Converting Heterogeneous Graphs to Homogeneous Graphs

Heterographs provide a clean interface for managing nodes/edges of different types and their associated features. This is particularly helpful when:

1. The features for nodes/edges of different types have different data types or sizes.
2. We want to apply different operations to nodes/edges of different types.

If the above conditions do not hold and one does not want to distinguish node/edge types in modeling, then DGL allows converting a heterogeneous graph to a homogeneous graph with `dgl.DGLGraph.to_homogeneous()` API. It proceeds as follows:

1. Relabels nodes/edges of all types using consecutive integers starting from 0
2. Merges the features across node/edge types specified by the user.

```
>>> g = dgl.heterograph({
...     ('drug', 'interacts', 'drug'): (th.tensor([0, 1]), th.tensor([1, 2])),
...     ('drug', 'treats', 'disease'): (th.tensor([1]), th.tensor([2]))})
>>> g.nodes['drug'].data['hv'] = th.zeros(3, 1)
>>> g.nodes['disease'].data['hv'] = th.ones(3, 1)
>>> g.edges['interacts'].data['he'] = th.zeros(2, 1)
>>> g.edges['treats'].data['he'] = th.zeros(1, 2)

>>> # By default, it does not merge any features
>>> hg = dgl.to_homogeneous(g)
>>> 'hv' in hg.ndata
False

>>> # Copy edge features
>>> # For feature copy, it expects features to have
>>> # the same size and dtype across node/edge types
>>> hg = dgl.to_homogeneous(g, edata=['he'])
DGLError: Cannot concatenate column ‘he’ with shape Scheme(shape=(2,), dtype=torch.float32) and
shape Scheme(shape=(1,), dtype=torch.float32)

>>> # Copy node features
>>> hg = dgl.to_homogeneous(g, ndata=['hv'])
>>> hg.ndata['hv']
tensor([[1.],
        [1.],
        [1.],
        [0.],
        [0.],
        [0.]])
```

The original node/edge types and type-specific IDs are stored in `ndata` and `edata`.

```
>>> # Order of node types in the heterograph
>>> g.ntypes
['disease', 'drug']
>>> # Original node types
>>> hg.ndata[dgl.NTYPE]
tensor([0, 0, 0, 1, 1, 1])
>>> # Original type-specific node IDs
>>> hg.ndata[dgl.NID]
tensor([0, 1, 2, 0, 1, 2])

>>> # Order of edge types in the heterograph
>>> g.etypes
['interacts', 'treats']
>>> # Original edge types
>>> hg.edata[dgl.ETYPE]
tensor([0, 0, 1])
>>> # Original type-specific edge IDs
>>> hg.edata[dgl.EID]
tensor([0, 1, 0])
```

For modeling purposes, one may want to group some relations together and apply the same operation to them. To address this need, one can first take an edge type subgraph of the heterograph and then convert the subgraph to a homogeneous graph.

```
>>> g = dgl.heterograph({
...     ('drug', 'interacts', 'drug'): (th.tensor([0, 1]), th.tensor([1, 2])),
...     ('drug', 'interacts', 'gene'): (th.tensor([0, 1]), th.tensor([2, 3])),
...     ('drug', 'treats', 'disease'): (th.tensor([1]), th.tensor([2]))
... })
>>> sub_g = dgl.edge_type_subgraph(g, [('drug', 'interacts', 'drug'),
...                                     ('drug', 'interacts', 'gene')])
>>> h_sub_g = dgl.to_homogeneous(sub_g)
>>> h_sub_g
Graph(num_nodes=7, num_edges=4,
      ...)
```