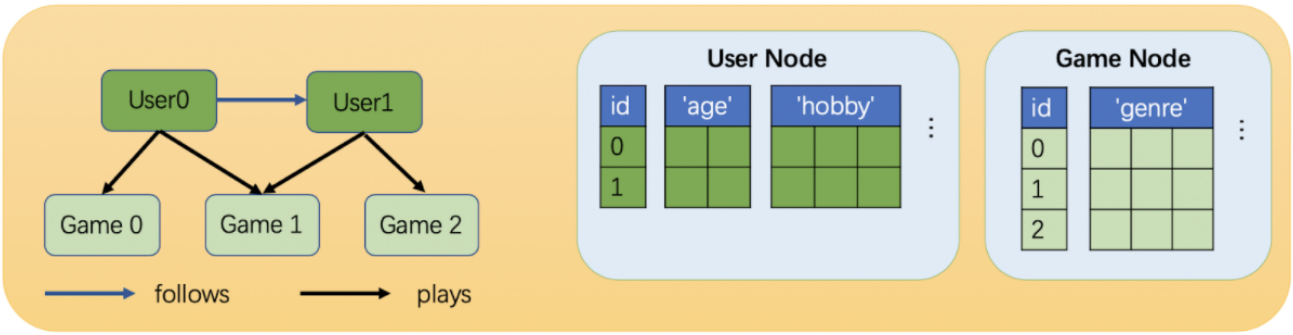
1.5 异构图

(English Version)

相比同构图，异构图里可以有不同类型的节点和边。这些不同类型的节点和边具有独立的ID空间和特征。例如在下图中，”用户”和”游戏”节点的ID都是从0开始的，而且两种节点具有不同的特征。



一个异构图示例。该图具有两种类型的节点(“用户”和”游戏”)和两种类型的边(“关注”和”玩”)。

创建异构图

在DGL中，一个异构图由一系列子图构成，一个子图对应一种关系。每个关系由一个字符串三元组 定义 `(源节点类型，边类型，目标节点类型)` 。由于这里的关系定义消除了边类型的歧义，DGL称它们为规范边类型。

下面的代码是一个在DGL中创建异构图的示例。

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

>>> # 创建一个具有3种节点类型和3种边类型的异构图
>>> 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')]
```

注意，同构图和二分图只是一种特殊的异构图，它们只包括一种关系。

```
>>> # 一个同构图
>>> dgl.heterograph({('node_type', 'edge_type', 'node_type'): (u, v)})
>>> # 一个二分图
>>> dgl.heterograph({('source_type', 'edge_type', 'destination_type'): (u, v)})
```

与异构图相关联的 *metagraph* 就是图的模式。它指定节点集和节点之间的边的类型约束。*metagraph* 中的一个节点 u 对应于相关异构图中的一个节点类型。*metagraph* 中的边 (u, v) 表示在相关异构图中存在从 u 型节点到 v 型节点的边。

```
>>> 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')])
```

相关API: `dgl.heterograph()`、`ntypes`、`etypes`、`canonical_etypes`、`metagraph`。

使用多种类型

当引入多种节点和边类型后，用户在调用DGLGraph API以获取特定类型的信息时，需要指定具体的节点和边类型。此外，不同类型的节点和边具有单独的ID。

```
>>> # 获取图中所有节点的数量
>>> g.num_nodes()
10
>>> # 获取drug节点的数量
>>> g.num_nodes('drug')
3
>>> # 不同类型的节点有单独的ID。因此，没有指定节点类型就没有明确的返回值。
>>> g.nodes()
DGLError: Node type name must be specified if there are more than one node types.
>>> g.nodes('drug')
tensor([0, 1, 2])
```

为了设置/获取特定节点和边类型的特征，DGL提供了两种新类型的语法：*g.nodes['node_type'].data['feat_name']* 和 *g.edges['edge_type'].data['feat_name']* 。

```
>>> # 设置/获取"drug"类型的节点的"hv"特征
>>> g.nodes['drug'].data['hv'] = th.ones(3, 1)
>>> g.nodes['drug'].data['hv']
tensor([[1.],
        [1.],
        [1.]])
>>> # 设置/获取"treats"类型的边的"he"特征
>>> g.edges['treats'].data['he'] = th.zeros(1, 1)
>>> g.edges['treats'].data['he']
tensor([[0.]])
```

如果图里只有一种节点或边类型，则不需要指定节点或边的类型。

```
>>> 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])
>>> # 设置/获取单一类型的节点或边特征，不必使用新的语法
>>> g.ndata['hv'] = th.ones(4, 1)
```

Note

当边类型唯一地确定了源节点和目标节点的类型时，用户可以只使用一个字符串而不是字符串三元组来指定边类型。例如， 对于具有两个关系 `('user', 'plays', 'game')` 和 `('user', 'likes', 'game')` 的异构图， 只使用 `'plays'` 或 `'like'` 来指代这两个关系是可以的。

从磁盘加载异构图

逗号分隔值（CSV）

一种存储异构图的常见方法是在不同的CSV文件中存储不同类型的节点和边。下面是一个例子。

```
# 数据文件夹
data/
|-- drug.csv         # drug节点
|-- gene.csv         # gene节点
|-- disease.csv      # disease节点
|-- drug-interact-drug.csv   # drug-drug相互作用边
|-- drug-interact-gene.csv   # drug-gene相互作用边
|-- drug-treat-disease.csv   # drug-disease治疗边
```

与同构图的情况类似，用户可以使用像Pandas这样的包先将CSV文件解析为numpy数组或框架张量，再构建一个关系字典，并用它构造一个异构图。 这种方法也适用于其他流行的文件格式，比如GML或JSON。

DGL二进制格式

DGL提供了 `dgl.save_graphs()` 和 `dgl.load_graphs()` 函数，分别用于以二进制格式保存异构图和加载它们。

边类型子图

用户可以通过指定要保留的关系来创建异构图的子图，相关的特征也会被拷贝。

```
>>> 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)

>>> # 保留关系 ('drug', 'interacts', 'drug') 和 ('drug', 'treats', 'disease') 。
>>> # 'drug' 和 'disease' 类型的节点也会被保留
>>> 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')])
>>> # 相关的特征也会被拷贝
>>> eg.nodes['drug'].data['hv']
tensor([[1.],
        [1.],
        [1.]])
```

将异构图转化为同构图

异构图为管理不同类型的节点和边及其相关特征提供了一个清晰的接口。这在以下情况下尤其有用:

1. 不同类型的节点和边的特征具有不同的数据类型或大小。
2. 用户希望对不同类型的节点和边应用不同的操作。

如果上述情况不适用，并且用户不希望在建模中区分节点和边的类型，则DGL允许使用 `dgl.DGLGraph.to_homogeneous()` API将异构图转换为同构图。 具体行为如下:

1. 用从0开始的连续整数重新标记所有类型的节点和边。
2. 对所有的节点和边合并用户指定的特征。

```
>>> 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)

>>> # 默认情况下不进行特征合并
>>> hg = dgl.to_homogeneous(g)
>>> 'hv' in hg.ndata
False

>>> # 拷贝边的特征
>>> # 对于要拷贝的特征，DGL假定不同类型的节点或边的需要合并的特征具有相同的大小和数据类型
>>> 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)

>>> # 拷贝节点特征
>>> hg = dgl.to_homogeneous(g, ndata=['hv'])
>>> hg.ndata['hv']
tensor([[1.],
        [1.],
        [1.],
        [0.],
        [0.],
        [0.]])
```

原始的节点或边的类型和对应的ID被存储在 `ndata` 和 `edata` 中。

```
>>> # 异构图中节点类型的顺序
>>> g.ntypes
['disease', 'drug']
>>> # 原始节点类型
>>> hg.ndata[dgl.NTYPE]
tensor([0, 0, 0, 1, 1, 1])
>>> # 原始的特定类型节点ID
>>> hg.ndata[dgl.NID]
tensor([0, 1, 2, 0, 1, 2])

>>> # 异构图中边类型的顺序
>>> g.etypes
['interacts', 'treats']
>>> # 原始边类型
>>> hg.edata[dgl.ETYPE]
tensor([0, 0, 1])
>>> # 原始的特定类型边ID
>>> hg.edata[dgl.EID]
tensor([0, 1, 0])
```

出于建模的目的，用户可能需要将一些关系合并，并对它们应用相同的操作。为了实现这一目的，可以先抽取异构图的边类型子图，然后将该子图转换为同构图。

```
>>> 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,
      ...)
```