1. Data overview

1.2. Basic info

`file_path`: “../data/full_graph.gpickle”

`Number of nodes`: 35906

`Number of edges`: 75111

1.3. Graph schema

```
node type: Company
   Company -- ACQUIRED -> Company
   Company -- ASSETS_BAND -> Quant_Metric
   Company -- CEO -> Person
   Company -- CHAIRMAN -> Person
   Company -- EMPLOYEE_COUNT_BAND -> Quant_Metric
   Company -- EQUITY_BAND -> Quant_Metric
   Company -- GROUPED_IN -> Group
   Company -- HEADQUARTERS -> Location
   Company -- INDUSTRY -> Industry
   Company -- MARKET_CAP_BAND -> Quant_Metric
   Company -- PROFIT_BAND -> Quant_Metric
   Company -- RESIDES_IN -> Location
node type: Person
   Person -- CEO -> Company
   Person -- CHAIRMAN -> Company
node type: Quant_Metric
   Quant_Metric -- ASSETS_BAND -> Company
   Quant_Metric -- EMPLOYEE_COUNT_BAND -> Company
   Quant_Metric -- EQUITY_BAND -> Company
   Quant_Metric -- MARKET_CAP_BAND -> Company
   Quant_Metric -- PROFIT_BAND -> Company
node type: Industry
   Industry -- INDUSTRY -> Company
node type: Group
   Group -- GROUPED_IN -> Company
```

```
node type: Location
   Location -- HEADQUARTERS -> Company
   Location -- RESIDES_IN -> Company
```

2.Modeling

2.1. Parameters

`test_pred_target_edge_count` = .3

- how many of the target edges we drop to later use as test data to evaluate prediction accuracy

`train_pred_target_edge_count` = .99999

- how many of the target edges we drop to use in training, aka finding optima via SGD and backprop

2.2. Train test split

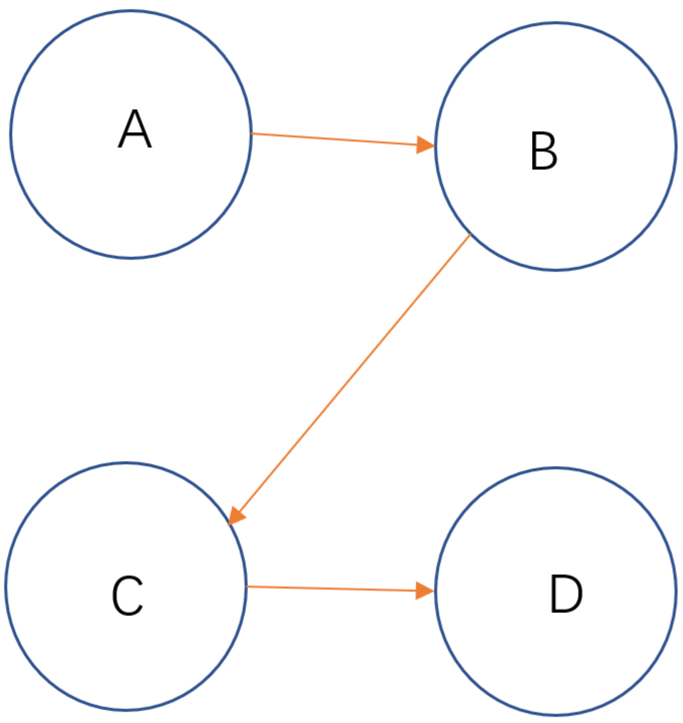
The intuition is to use **negative sampling**

Original data:

For example, lets say we have node A, B, C, D. The only two edges are A -> B, and B -> C, which means A acquired B, B acquired C, and C acquired D.

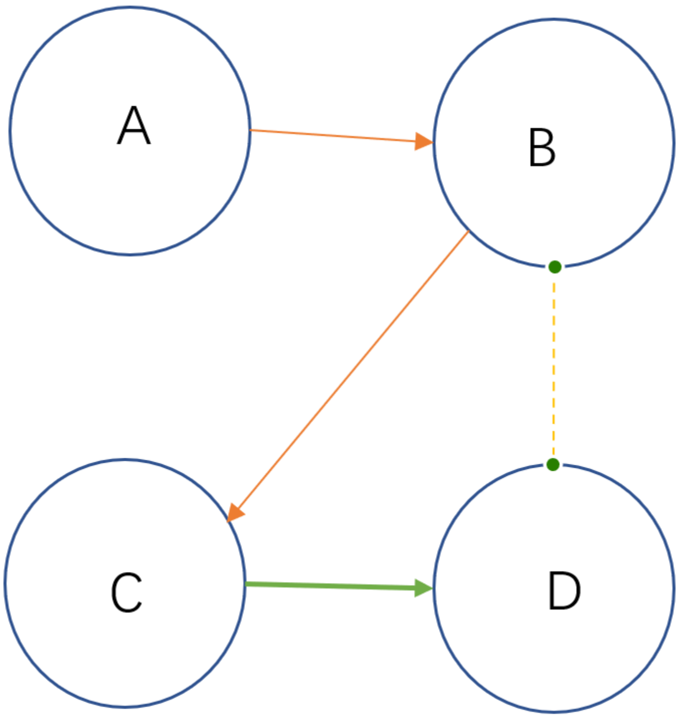
`Number of nodes`: 4

`Number of edges`: 2



Test data:

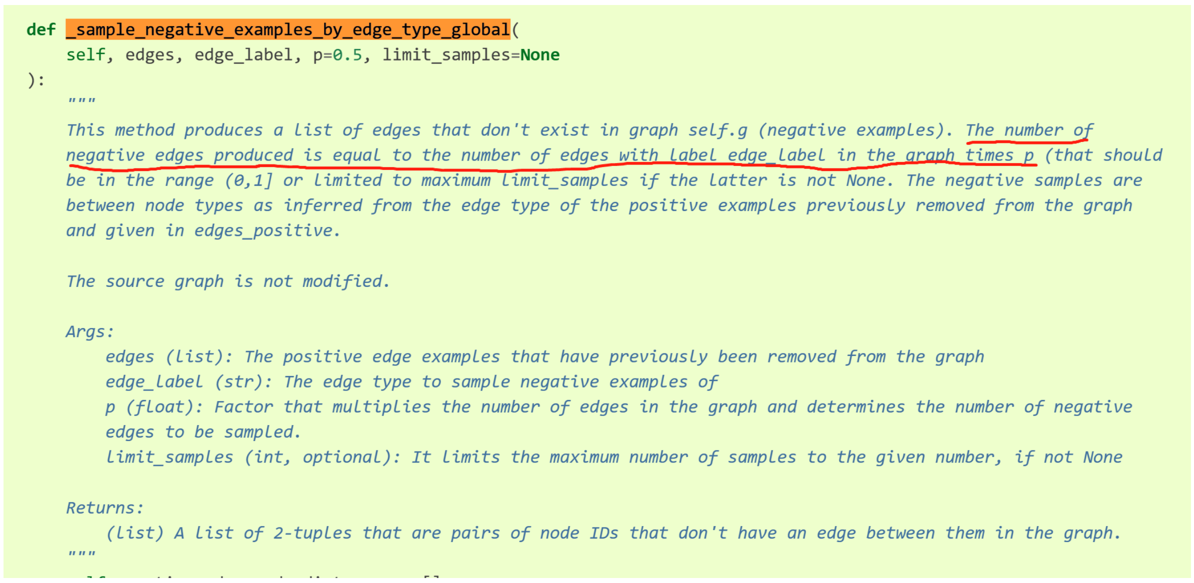
Lets say `test_pred_target_edge_count` = 0.33 right now, so we are going to randomly sample two nodes, until we get **3 * 33% = 1 of the total edges**. But at the sample time, it uses negative sampling, which meanings get the **same number** of **'pseudo edges'**, that does not exist in the graph.



For example, randomly select `C -> D` , then it would be positive sample `1`. The `train_test_split` automatically use negative sampling to get the **same number** of **‘pseudo edges’**. For example, let it be `B -> D`. There is not really connection here it is just used to balance the data.

So the **test set** includes:

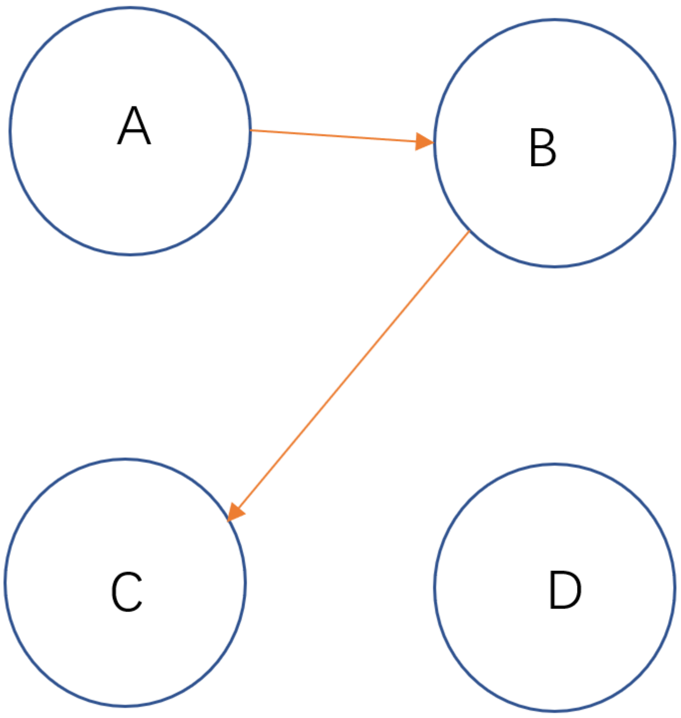
id	**label**
(C, D)	1
(B, D)	0



```python
def _sample_negative_examples_by_edge_type_global(
    self, edges, edge_label, p=0.5, limit_samples=None
):
    """
    This method produces a list of edges that don't exist in graph self.g (negative examples). The number of
    negative edges produced is equal to the number of edges with label edge_label in the graph times p (that should
    be in the range (0,1] or limited to maximum limit_samples if the latter is not None. The negative samples are
    between node types as inferred from the edge type of the positive examples previously removed from the graph
    and given in edges_positive.

    The source graph is not modified.

    Args:
        edges (list): The positive edge examples that have previously been removed from the graph
        edge_label (str): The edge type to sample negative examples of
        p (float): Factor that multiplies the number of edges in the graph and determines the number of negative
        edges to be sampled.
        limit_samples (int, optional): It limits the maximum number of samples to the given number, if not None

    Returns:
        (list) A list of 2-tuples that are pairs of node IDs that don't have an edge between them in the graph.
    """
```

```python
edge_splitter_test = EdgeSplitter(st_g)
G_test, edge_ids_test, edge_labels_test =
edge_splitter_test.train_test_split(
    p=test_pred_target_edge_count, method="global",
edge_label='ACQUIRED'
)
```

Take a closer look at the function, here `edge_ids_test` and `edge_labels_test` are the test table discussed above. `G_test` is the reduced graph (positive edges removed) as follow:

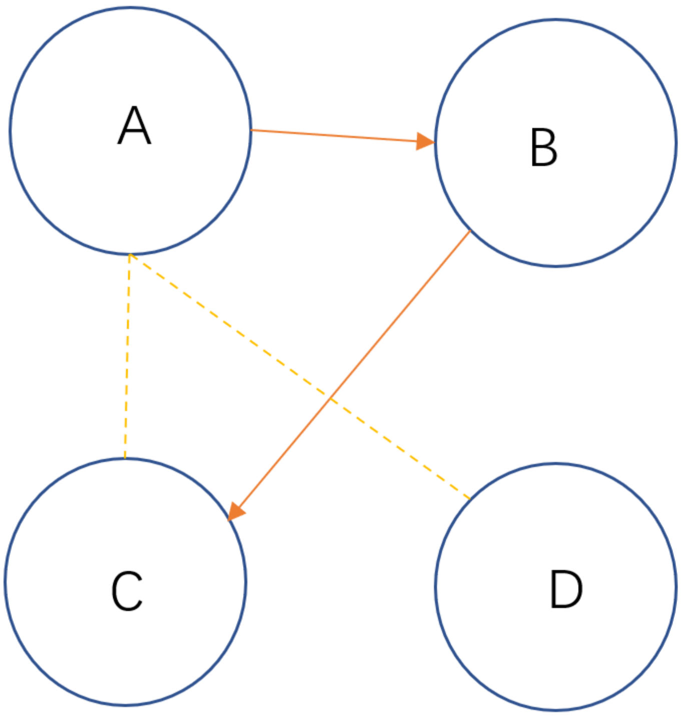


Train data:

The same process. By setting `train_pred_target_edge_count` = .99999. It basically **samples all edges remaining** here. That is:

id	label
(A, B)	1
(B, C)	1

And it also do **negative sampling** to get the sample amount of negative samples. So it turns to be :



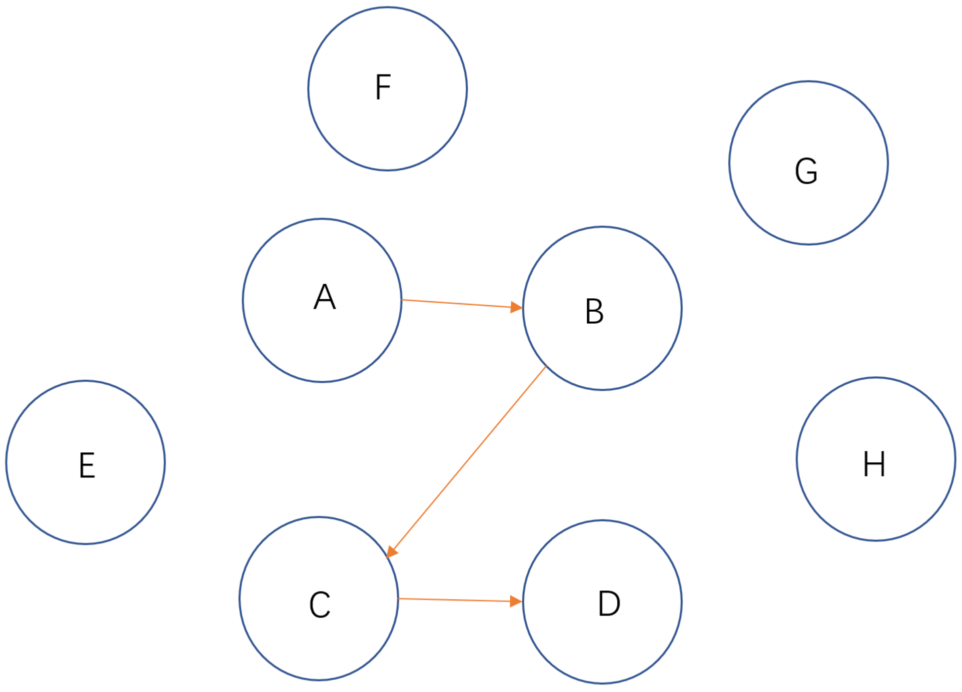
id	label
(A, B)	1
(B, C)	1
(A, C)	**0**
(A, D)	**0**

3. Imbalance problem

So if it is `positive:negative=1:1` within both training and testing data, **why still we say the data is imbalanced?** Personally I think there is a problem when we construct test data using this **current `1:1` negative sampling method.**

The test data is not an reasonable benchmark reflecting the **real world distribution**. The graph is actually extremely **sparse**!

It should be something like:



The prediction we are going to make is a set connecting each two possible nodes. That is $A_8^2 - 3 = 53$. The **sparsity** means that **only a few of them** are supposed to be connected! Let’s say only `A->D` should be positive. So the `positive:negative=1:52`. In my opinion, that is why the problem is imbalanced.