Published in Towards Data Science

Vijini Mallawaarachchi Follow
Dec 2, 2020 · 8 min read · Member-only · Listen

Matching of Bipartite Graphs using NetworkX

A simple introduction to matching in bipartite graphs with Python code examples

Graph matching can be applied to solve different problems including scheduling, designing flow networks and modelling bonds in chemistry. In this article, I will give a basic introduction to bipartite graphs and graph matching, along with code examples using the python library **NetworkX**.

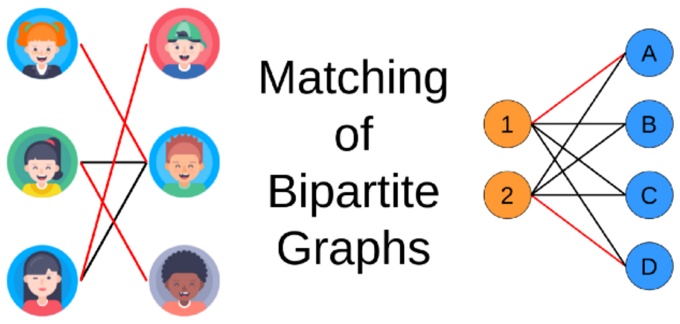

Image by Author

Before moving to the nitty-gritty details of graph matching, let's see what are bipartite graphs.

Bipartite Graphs

According to Wikipedia,

> A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets U and V such that every edge connects a vertex in U to one in V.

In a bipartite graph, we have two sets of vertices **U** and **V** (known as **bipartitions**) and each edge is incident on one vertex in **U** and one vertex in **V**. There will not be any edges connecting two vertices in **U** or two vertices in **V**. Figure 1 denotes an example bipartite graph.

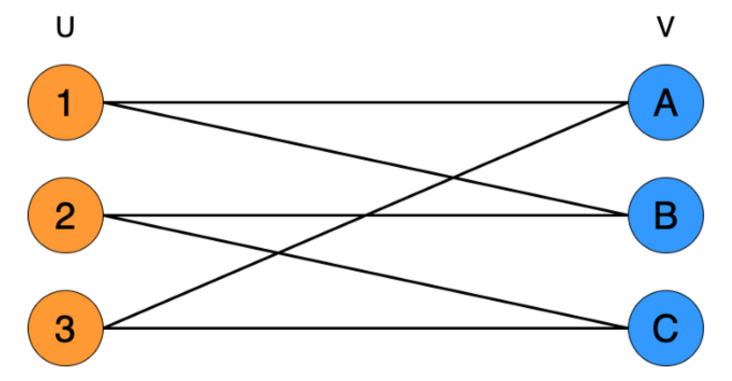

Figure 1: Bipartite graph (Image by Author)

Let’s see how we can represent a bipartite graph using NetworkX. If you do not have NetworkX installed on your machine, you can follow the instructions given here. Now let’s create the bipartite graph shown in Figure 1.

```
# imports
import networkx as nx
from networkx.algorithms import bipartite


# Initialise graph
B = nx.Graph()


# Add nodes with the node attribute "bipartite"
top_nodes = [1, 2, 3]
bottom_nodes = ["A", "B", "C"]
B.add_nodes_from(top_nodes, bipartite=0)
B.add_nodes_from(bottom_nodes, bipartite=1)


# Add edges only between nodes of opposite node sets
B.add_edges_from([(1, "A"), (1, "B"), (2, "B"), (2, "C"), (3, "A"),
(3, "C")])
```

Matching of Bipartite Graphs

According to Wikipedia,

> A matching or independent edge set in an undirected graph is a set of edges without common vertices.

In simple terms, a matching is a graph where each vertex has either zero or one edge incident to it.

If we consider a bipartite graph, the matching will consist of edges connecting one vertex in **U** and one vertex in **V** and each vertex (in **U** and **V**) has either zero or one edge incident to it.

A vertex is considered as **matched** if it connects to one of the edges in the matching. Otherwise, the vertex is considered **unmatched**.

Example Matching Scenario

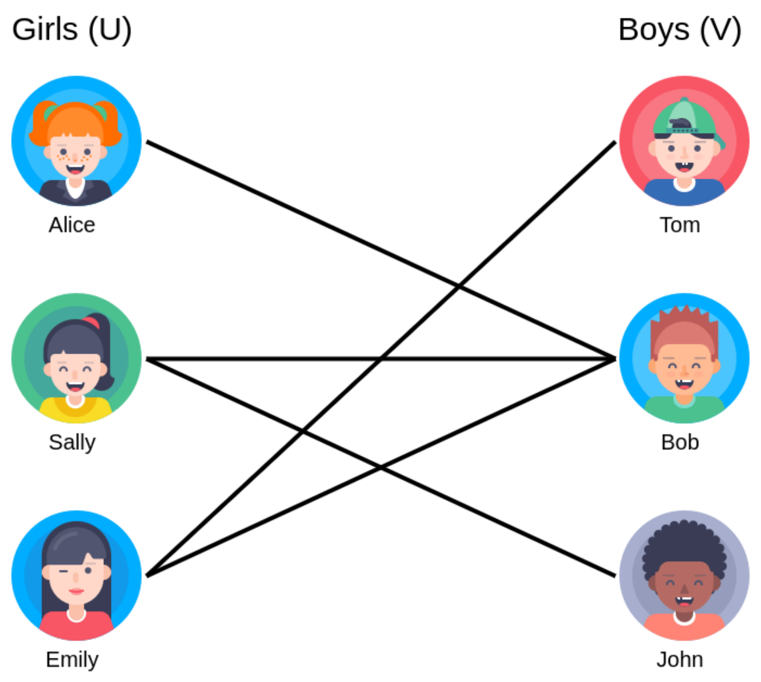

Figure 2: Preferences of girls and boys (Image by Author)

Consider an example scenario as shown in Figure 2, where you have to pair girls with boys and each girl has expressed a preference for one or more boys. You have to pair the girls with boys so that each girl is paired with a unique boy she prefers.

You can easily solve this problem by coming up with a matching for the bipartite graph as shown in Figure 3. You can model the girls and boys as bipartitions. Here we have three girls (Alice, Sally and Emily) and three boys (Tom, Bob and John.)

This example can be solved very easily without an algorithm as shown in Figure 3. We can see that Alice can be paired with Bob, Emily can be paired with Tom, and Sally can be paired with John. However, as the number of girls and boys increases (*i.e.*, the number of vertices in each bipartition increases), it can get complicated to obtain a matching by just looking at the bipartite graph.

Figure 4: No matching of girls and boys (Image by Author)

When we cannot get a Matching?

As shown in Figure 4, two girls can prefer the same boy and no other boys. If this boy is paired with one girl, the other girl will have no boy to be paired with. So there will be no matching. If there were two or more boys those two girls likes, then there can be a matching.

Similarly, if there are three girls who collectively prefer two boys, and we pair up one girl with one boy, then we have our previous scenario of two girls preferring the same boy. We can continue this way with more and more girls and boys.

To express this in a more graph-based manner, let's consider a set of vertices **S⊆U**. Define **N(S)** to be the set of all the neighbours of vertices in set S. If we consider our previous example, N(S) will be the set of boys liked by the girls in set S. The **matching condition** can be defined as follows.

> If a bipartite graph G={U, V} has a matching of U, then |N(S)|≥|S| for all S⊆U.

The converse of this is also true and it is known as the **Hall's Marriage Theorem**.

A matching is considered a **perfect matching** if each node has exactly one edge incident on it. For a perfect matching, we should have **|U|=|V|**.

Figure 5: Perfect matching (Image by Author)

Maximal Matching

A matching is considered as a **maximal matching** if no other edges can be added to the matching because every vertex is matched to another vertex. *i.e.*, we cannot add more edges without increasing the degree of one of the vertices to two.

Figure 6: Maximal matching (Image by Author)

Maximum Cardinality Matching

A matching is considered as a **maximum cardinality matching** if it contains the largest possible number of edges. As each edge will cover exactly two vertices, it is equivalent to finding a matching that covers as many vertices as possible. The ***Hopcroft-Karp algorithm*** can be used to solve this problem.

Figure 7: Maximum matching (Image by Author)

Note: Every maximum matching is a maximal matching, but not every maximal matching is a maximum matching. Moreover, every perfect matching is a maximum matching, but not every maximum matching is a perfect matching.

Maximum Weight Matching

In a weighted bipartite graph, a matching is considered a **maximum weight matching** if the sum of weights of the matching is maximised. This is also known as the *assignment problem*. The ***Hungarian algorithm*** can be used to solve this problem.

Minimum Weight Matching

In a weighted bipartite graph, a matching is considered a **minimum weight matching** if the sum of weights of the matching is minimised. The ***Karp algorithm*** can be used to solve this problem.

Running Examples

It's time to get our hands dirty. Let's dig into some code and see how we can obtain different matchings of bipartite graphs using the python library **NetworkX**.

1. Maximum Cardinality Matching

We will try to obtain a maximum cardinality matching for the bipartite graph shown in Figure 7.

Figure 7: Maximum matching (Image by Author)

We can use the NetworkX function named `hopcroft_karp_matching` to obtain the maximum cardinality matching of a bipartite graph as shown below.

```
# Initialise graph
B = nx.Graph()


# Add nodes with the node attribute "bipartite"
top_nodes = [1, 2]
bottom_nodes = ["A", "B", "C", "D"]
B.add_nodes_from(top_nodes, bipartite=0)
B.add_nodes_from(bottom_nodes, bipartite=1)


# Add edges only between nodes of opposite node sets
B.add_edges_from([(1, "A"), (1, "B"), (1, "C"), (1, "D"), (2, "A"),
(2, "D")])


#Obtain the maximum cardinality matching
my_matching = bipartite.matching.hopcroft_karp_matching(B, top_nodes)
```

`my_matching` will contain a dictionary with the edges of the matching as follows.

```
{1: 'A', 2: 'D', 'A': 1, 'D': 2}
```

You can also use the function `eppstein_matching` to obtain the maximum cardinality matching of a bipartite graph. This function implements David Eppstein's version of the algorithm Hopcroft–Karp algorithm.

2. Minimum Weight Matching

We will try to obtain a minimum weight matching for the bipartite graph shown in Figure 8.

Figure 8: Weighted bipartite graph (Image by Author)

We can use the NetworkX function named `minimum_weight_full_matching` to obtain the minimum weight full matching of the bipartite graph as shown below. The matching is defined as *full* as it follows the Karp algorithm where |U| and |V| can differ (if |U|=|V|, then it will be a minimum weight perfect matching).

```
# Initialise the graph
B = nx.Graph()

# Add nodes with the node attribute "bipartite"
top_nodes = [1, 2]
bottom_nodes = ["A", "B", "C", "D"]
B.add_nodes_from(top_nodes, bipartite=0)
B.add_nodes_from(bottom_nodes, bipartite=1)

# Add edges with weights
B.add_edge(1, "A", weight = 1)
B.add_edge(1, "B", weight = 4)
B.add_edge(1, "C", weight = 2)
B.add_edge(1, "D", weight = 1)
B.add_edge(2, "A", weight = 3)
B.add_edge(2, "B", weight = 1)
B.add_edge(2, "C", weight = 2)
B.add_edge(2, "D", weight = 2)

#Obtain the minimum weight full matching
my_matching = bipartite.matching.minimum_weight_full_matching(B,
top_nodes, "weight")
```

`my_matching` will contain a dictionary with the edges of the matching as follows.

```
{1: 'A', 2: 'B', 'A': 1, 'B': 2}
```

Applications

Applications of bipartite graph matching can be found in different fields including data science and computational biology. Many systems can be modelled as bipartite graphs and matchings can be obtained to identify the most similar pairings.

Matching is used in scheduling problems such as when we want to assign jobs

Chemical bonds are modelled as matching problems in chemistry. For example, a **Kekulé structure** of an aromatic compound consists of a perfect matching of its carbon skeleton [1].

Image by Gerd Altmann from Pixabay

Final Thoughts

Hope you got a brief idea about the matching of bipartite graphs and found this article useful. Feel free to use these code examples in your work.

Thank you for reading!

Cheers!

References

[1] Matching (graph theory) - Wikipedia — https://en.wikipedia.org/wiki/Matching_(graph_theory)

[3] 2.11.7 Bipartite Matching - MIT OpenCourseWare — https://youtu.be/HZLKDC9OSaQ

[4] Matching in Bipartite Graphs — http://discrete.openmathbooks.org/dmoi3/sec_matchings.html

[5] networkx.algorithms.bipartite.matching — NetworkX 2.5 — https://networkx.org/documentation/stable//_modules/networkx/algorithms/bipartite/matching.html

133 1