1 Question 1

We will distinguish the 3 vertex sets of the graph G:

- A: the vertices in the connected component that forms a complete graph. There are 20 vertices in A.
- B: the vertices in one part of the bipartite component. There are 10 vertices in B.
- C: the vertices in the other part of the bipartite component. There are 10 vertices in C.

In the original graph G, every pair of vertices in A is connected, and every pair consisting of one vertex from B and one from C is also connected.

On the other hand, $\overline{G}$ contains all edges where one vertex is from A and the other is from B, all edges where one vertex is from A and the other is from C, and all edges between pairs of vertices that are both in B or both in C. Therefore, the possible vertices triplets to form a triangle are:

- One vertex from A and two vertices from B: We can choose the vertex from A in $\binom{20}{1}$ ways and the two vertices from B in $\binom{10}{2}$ ways, thus, there are

$$\binom{20}{1}\binom{10}{2} = 20 \cdot 45 = 900 \text{ triangles.}$$

- One vertex from A and two vertices from C: Similarly, there are

$$\binom{20}{1}\binom{10}{2} == 20 \cdot 45 = 900 \text{ triangles.}$$

- Three vertices from B: We can choose three vertices from B in $\binom{10}{3}$ ways, thus, there are

$$\binom{10}{3} = 120 \text{ triangles.}$$

- Three vertices from C: Similarly, there are

$$\binom{10}{3} = 120 \text{ triangles.}$$

Summing everything together, we have that there are $900 + 900 + 120 + 120 = 2040$ triangles in graph $\overline{G}$.

2 Question 2

We begin by computing the gradient of $R(A, x) = \dfrac{x^\top Ax}{x^\top x}, x \neq 0$. We first define functions $f(x) = x^\top Ax$ and $g(x) = x^\top x$, so that $R(A, x) = \dfrac{f(x)}{g(x)}$. Since $\nabla f(x) = Ax + A^\top x = (A + A^\top)x = 2Ax$ (because A is an adjacency matrix, hence $A = A^\top$) and $\nabla g(x) = 2x$, we have:

$$\nabla R(x) = \frac{g(x)\nabla f(x) - f(x)\nabla g(x)}{g(x)^2} = 2\frac{(x^\top x)Ax - (x^\top Ax)x}{(x^\top x)^2}.$$

We will now show both directions of equivalence:

- x is a stationary point $\implies$ x is an eigenvector of A.
  A stationary point of a function is a point in which the gradient is zero. Therefore, we have:

$$\begin{aligned}
\nabla R(x) = 0 &\Leftrightarrow (x^\top x)Ax = (x^\top Ax)x \\
&\Leftrightarrow Ax = \frac{x^\top Ax}{x^\top x}x, x \neq 0 \\
&\Leftrightarrow Ax = R(A, x)\, x,
\end{aligned}$$

  thus, x is an eigenvector of A with eigenvalue of $R(A, x)$.

- x is an eigenvector of $A \implies x$ is a stationary point.
  Based on the assumption, we have that $Ax = \lambda x$, where λ is an eigenvalue. We also then have $x^\top Ax = x^\top \lambda x = \lambda x^\top x$.
  If we now look at the numerator in the formula for the gradient of function $R(A, x)$, we have:

$$(x^\top x)Ax - (x^\top Ax)x = (x^\top x)\lambda x - \lambda(x^\top x)x = \lambda(x^\top x)x - \lambda(x^\top x)x = 0,$$

  so $\nabla R(x) = 0$, and therefore, x is a stationary point.

Finally, from the previous implications, we can conclude that a non-zero vector x is a stationary point of $R(A, \cdot)$ iff x is an eigenvector of A.

3 Question 3

We will compute the modularity of the two clusterings using the formula:

$$Q = \sum^{n_c} \left[\frac{l_c}{m} - \left(\frac{d_c}{2m} \right)^2 \right].$$

Both clusterings contain 13 edges and 2 communities, thus $m = 13$, $n_c = 2$. The degrees of nodes in clusterings are the same and as follows:

$$\deg(1) = 3,\ \deg(2) = 3,\ \deg(3) = 3,\ \deg(4) = 2,\ \deg(5) = 4,\ \deg(6) = 4,\ \deg(7) = 2,\ \deg(8) = 3,\ \deg(9) = 2.$$

Clustering (a)

The blue community contains following edges: $(1,2), (1,3), (1,5), (2,5), (2,3), (3,4), (4,5)$, thus $l_1 = 7$. The sum of the degrees of nodes in it is $d_1 = deg(1) + deg(2) + deg(3) + deg(4) + deg(5) = 15$.
The orange community contains following edges: $(6,7), (6,8), (6,9), (7,8), (8,9)$, thus $l_2 = 5$. The sum of the degrees of nodes in it is $d_2 = deg(6) + deg(7) + deg(8) + deg(9) = 11$.
We can now compute the modularity:

$$Q = \left[\frac{l_1}{m} - \left(\frac{d_1}{2m} \right)^2 \right] + \left[\frac{l_2}{m} - \left(\frac{d_2}{2m} \right)^2 \right] = \left[\frac{7}{13} - \left(\frac{15}{26} \right)^2 \right] + \left[\frac{5}{13} - \left(\frac{11}{26} \right)^2 \right] = 0.205 + 0.205 = 0.41.$$

Clustering (b)

The blue community contains following edges: $(2,3), (3,4)$, thus $l_1 = 2$. The sum of the degrees of nodes in it is $d_1 = deg(2) + deg(3) + deg(4) = 8$.
The orange community contains following edges: $(1,5), (5,6), (6,7), (6,8), (6,9), (7,8), (8,9)$, thus $l_2 = 7$. The sum of the degrees of nodes in it is $d_2 = deg(1) + deg(5) + deg(6) + deg(7) + deg(8) + deg(9) = 18$.
We can now compute the modularity:

$$Q = \left[\frac{l_1}{m} - \left(\frac{d_1}{2m} \right)^2 \right] + \left[\frac{l_2}{m} - \left(\frac{d_2}{2m} \right)^2 \right] = \left[\frac{2}{13} - \left(\frac{8}{26} \right)^2 \right] + \left[\frac{7}{13} - \left(\frac{18}{26} \right)^2 \right] = 0.059 + 0.059 = 0.118.$$

Comparison

Since higher modularity values indicate a better community structure, we can observe that the partition in (a) is more optimal than the partition in (b). Indeed, if we look at partition (a), only a single edge, (5-6), connects the 2 communities, while all other edges stay within their respective community. On the other hand, in partition (b), there are 4 edges that connect two communities which lowers the modularity value.

4 Question 4

In Figure 1 we show two graphs, G_1 and G_2, that satisfy the required condition. Graphs are not isomorphic (degrees' set are not equivalent; also confirmed by is_isomorphic() function of NetworkX). Both graphs contain 4 shortest paths at distance 1 and 2 shortest paths at distance 2, and no pairs at larger distances. Therefore,

$$\varphi(G_1) = \varphi(G_2) = (4, 2, 0, \ldots),$$

so the shortest path kernel indeed maps them to the same representation.

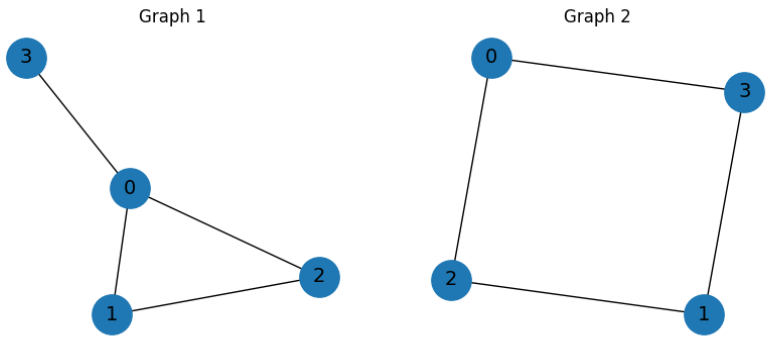

Figure 1: Two non-isomorphic graphs mapped to the same representation by the shortest path kernel.

5 Question 5

In Figure 2 we show two graphs, G and G' to which we will perform one full iteration of the Weisfeiler–Lehman (WL) subtree kernel algorithm.

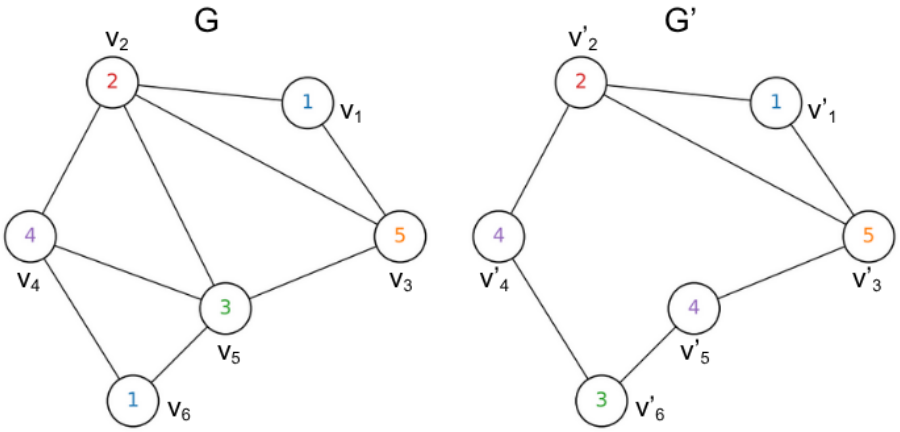

Figure 2: Graph G with nodes $v_1, \ldots, v_6$ and graph G' with nodes $v'_1, \ldots, v'_6$, each assigned a discrete label.

We perform the one full iteration of WL subtree kernel algorithm as described in Algorithm 2 of [1].

Multiset-label determination

For each node v, we need to assign multiset of neighbor labels to it, i.e., we need to define $M_1(v) := \{\, l_0(u) \mid u \in N(v) \,\}$, where $l_0(u)$ is the initial label for node u, and $N(v)$ is the set of neighbors for v. Therefore, for graph G:

- Node v_1: Its neighbors are v_2 (label 2) and v_3 (label 5), thus, $M_1(v_1) := \{2, 5\}$.
- Node v_2: Its neighbors are v_1 (label 1), v_3 (label 5), v_4 (label 4) and v_5 (label 3), thus, $M_1(v_2) := \{1, 5, 4, 3\}$.
- Node v_3: Its neighbors are v_1 (label 1), v_2 (label 2) and v_5 (label 3), thus, $M_1(v_3) := \{1, 2, 3\}$.
- Node v_4: Its neighbors are v_2 (label 2), v_5 (label 3) and v_6 (label 1), thus, $M_1(v_4) := \{2, 3, 1\}$.
- Node v_5: Its neighbors are v_2 (label 2), v_3 (label 5), v_4 (label 4) and v_6 (label 1), thus, $M_1(v_5) := \{2, 5, 4, 1\}$.
- Node v_6: Its neighbors are v_4 (label 4) and v_5 (label 3), thus, $M_1(v_6) := \{4, 3\}$.

Similarly, for graph G' and nodes v' we have:

- Node v'_1: Its neighbors are v'_2 (label 2) and v'_3 (label 5), thus, $M_1(v'_1) := \{2, 5\}$.
- Node v'_2: Its neighbors are v'_1 (label 1), v'_3 (label 5), v_4 (label 4), thus, $M_1(v'_2) := \{1, 5, 4\}$.

- Node v_3': Its neighbors are v_1' (label 1), v_2' (label 2) and v_5' (label 4), thus, $M_1(v_3') := \{1, 2, 4\}$.
- Node v_4': Its neighbors are v_2' (label 2) and v_6' (label 3), thus, $M_1(v_4') := \{2, 3\}$.
- Node v_5': Its neighbors are v_3 (label 5), and v_6 (label 3), thus, $M_1(v_5') := \{5, 3\}$.
- Node v_6': Its neighbors are v_4 (label 4) and v_5 (label 4), thus, $M_1(v_6') := \{4, 4\}$.

Sorting each multiset

We first sort each multiset $M_1(v)$ in ascending order and concatenate them into a string s_1.

Graph G : $s_1(v_1) = \{25\}$, $s_1(v_2) = \{1345\}$, $s_1(v_3) = \{123\}$, $s_1(v_4) = \{123\}$, $s_1(v_5) = \{1245\}$, $s_1(v_6) = \{34\}$.

Graph G' : $s_1(v_1') = \{25\}$, $s_1(v_2') = \{145\}$, $s_1(v_3') = \{124\}$, $s_1(v_4') = \{23\}$, $s_1(v_5') = \{35\}$, $s_1(v_6') = \{44\}$.

Now, for every node in two graphs we need to add its initial label to its multiset.

Graph G : $s_1(v_1) = \{125\}$, $s_1(v_2) = \{21345\}$, $s_1(v_3) = \{5123\}$, $s_1(v_4) = \{4123\}$, $s_1(v_5) = \{31245\}$, $s_1(v_6) = \{134\}$.

Graph G' : $s_1(v_1') = \{125\}$, $s_1(v_2') = \{2145\}$, $s_1(v_3') = \{5124\}$, $s_1(v_4') = \{423\}$, $s_1(v_5') = \{435\}$, $s_1(v_6') = \{344\}$.

Label compression

We now need to map each string $s_1(v)$ and $s_1(v')$ to a new compressed label. In [1], the algorithm applies a hash function to guarantee unique relabeling across the graph. However, we will approximate the hashing by mapping the unique string to a discrete set of labels $\{a, b, c, ...\}$:

$$
\begin{aligned}
&s_1(v_1) = s_1(v_1') = \{125\} \longrightarrow a,\\
&s_1(v_2) = \{21345\} \longrightarrow b,\\
&s_1(v_3) = \{5123\} \longrightarrow c,\\
&s_1(v_4) = \{4123\} \longrightarrow d,\\
&s_1(v_5) = \{31245\} \longrightarrow e,\\
&s_1(v_6) = \{134\} \longrightarrow f,\\
&s_1(v_2') = \{2145\} \longrightarrow g,\\
&s_1(v_3') = \{5124\} \longrightarrow h,\\
&s_1(v_4') = \{423\} \longrightarrow i,\\
&s_1(v_5') = \{435\} \longrightarrow j,\\
&s_1(v_6') = \{344\} \longrightarrow k.
\end{aligned}
$$

Relabeling

The final step of the algorithm is to assign each vertex v and v' a new label $l_1(v)$ and $l_1(v')$, respectively. New label will be the corresponding compressed label from the previous step:

Graph G : $l_1(v_1) = a$, $l_1(v_2) = b$, $l_1(v_3) = c$, $l_1(v_4) = d$, $l_1(v_5) = e$, $l_1(v_6) = f$.

Graph G' : $l_1(v_1') = a$, $l_1(v_2') = g$, $l_1(v_3') = h$, $l_1(v_4') = i$, $l_1(v_5') = j$, $l_1(v_6') = k$.

Subtree kernel

We can now compute the subtree kernel using the formulas in [1]:

$$k_{\mathrm{WL}}^{(h)}(G, G') = k(G_0, G'_0) + k(G_1, G'_1) + \cdots + k(G_h, G'_h),$$

where $\{G_0, \ldots, G_h\}$ and $\{G'_0, \ldots, G'_h\}$ are the WL graph sequences of G and G', respectively and h denotes the number of iterations. Base kernel k is computed as follows:

$$k(G_i, G'_i) \;=\; \phi(G_i)\, \phi(G'_i)$$

where $\phi(G_i)$ and $\phi(G'_i)$ denote the number of occurrences of a label in graphs G_i and G'_i.
In our case, before the iteration, the set of labels is $\{1, 2, 3, 4, 5\}$. From 2, we have:

$$\phi(G_0) = (2,\ 1,\ 1,\ 1,\ 1),$$

and

$$\phi(G'_0) = (1,\ 1,\ 1,\ 2,\ 1).$$

Therefore, we can compute the $k(G_0, G'_0)$ as:

$$k(G_0, G'_0) = 2 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 2 + 1 \cdot 1 = 7.$$

After the iteration, the set of labels is $\{a, b, c, d, e, f, g, h, i, j, k\}$. From 5 we have:

$$\phi(G_1) = (1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0),$$

and

$$\phi(G'_1) = (1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 1,\ 1,\ 1,\ 1).$$

Therefore, we can compute the $k(G_1, G'_1)$ as:

$$k(G_1, G'_1) = 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 = 1.$$

Finally, the WL subtree kernel value after one iteration is:

$$k_{\mathrm{WL}}^{(1)}(G, G') = k(G_0, G'_0) + k(G_1, G'_1) = 7 + 1 = 8.$$

Conclusion

We see that the final kernel value is 7+1=8, where the contribution from the kernel before refinement dominates. Indeed, if we look at the two initial graphs, they share the same label set, and the frequencies of these labels are quite similar, thus, the kernel value measuring similarity is moderate (i.e., 7). After one WL iteration, the labels are recomputed based on their previous label and the labels of their neighbors. In this setting, the kernel value drops to 1, meaning that almost none of the refined labels match between the two graphs and that their local neighborhood similarity is very low. Therefore, the WL kernel tells us that the two graphs actually have low structural similarity beyond the initial labels themselves.

References

[1] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler–lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.