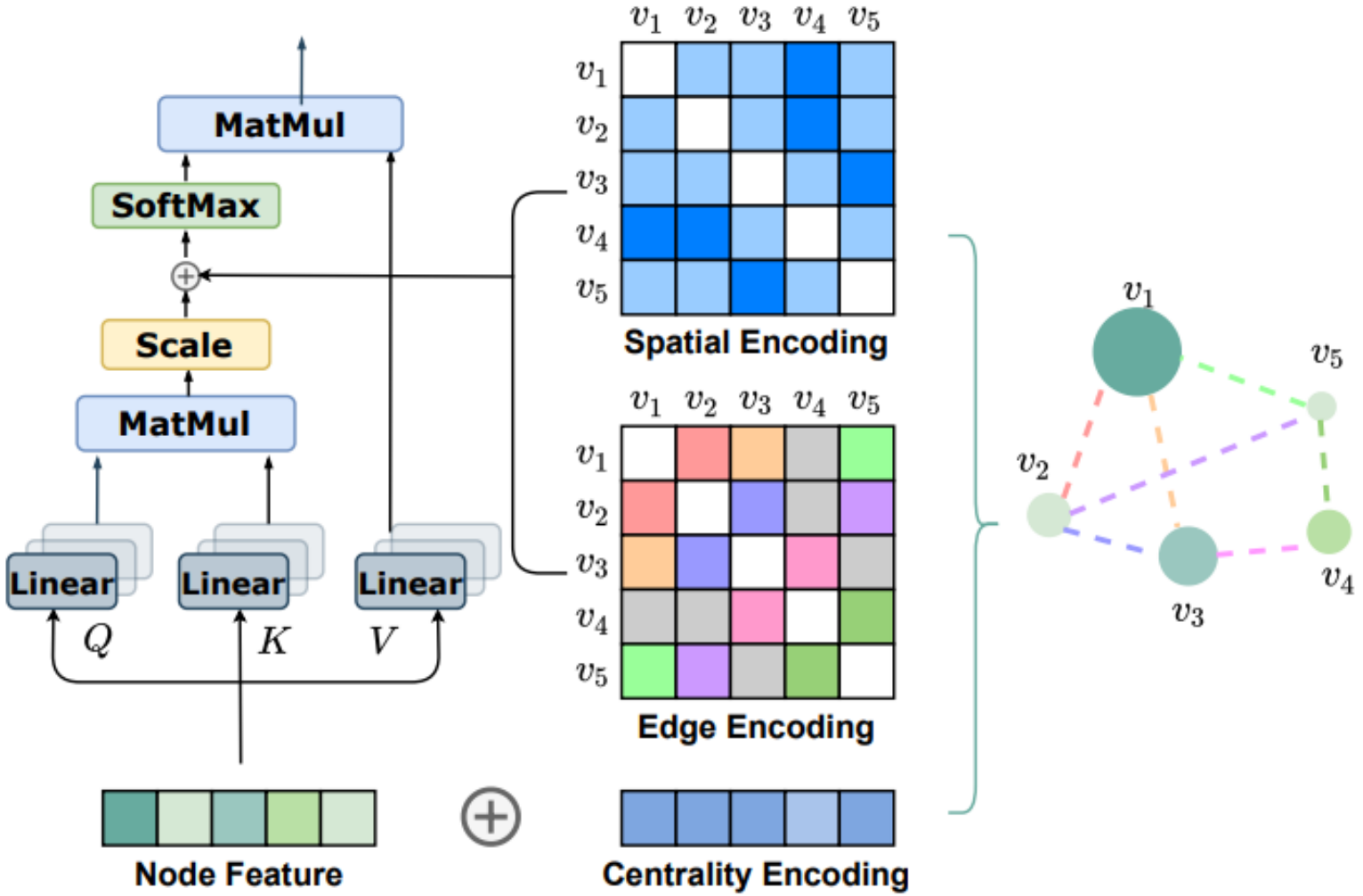
IIC-3641 Aprendizaje Automático basado en Grafos

https://github.com/marcelomendoza/IIC3641

- RECAPITULACIÓN -

Graphormer

En la AF8 trabajamos con el Graphormer:



Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., Schuurmans, D., & Liu, T. (2021). Do Transformers Really Perform Bad for Graph Representation? In NeurIPS 2021. https://arxiv.org/abs/2106.05234

Graphormer

Al combinarlo con *pooling*, pudimos usar el graphormer para clustering. Usando la matriz de asignación, pudimos recalcular la matriz de adyacencia:

$$A' = S^T A S$$

Donde $S \in \mathbb{R}^{n \times k}, \quad k < n$.

Una limitación de este enfoque es que aplica sólo a grafos conocidos. En el graphormer, cada nodo será representado como un *token* del vocabulario de nodos. Esto es útil para entrenamiento pero en rigor no sabemos cual es su capacidad expresiva.

Abordaremos el problema de la capacidad expresiva en la clase de hoy.

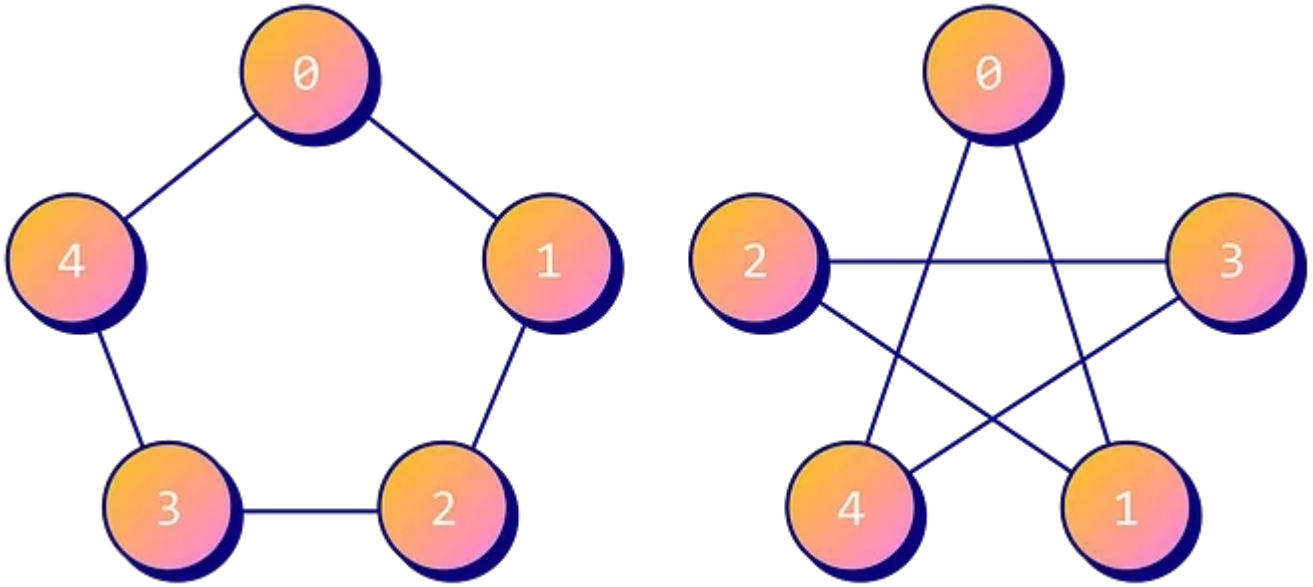
- GRAPH ISOMORPHISM NETWORKS -

¿Cuán expresivos son los modelos que hemos visto?

Algo relevante en GML consiste en dilucidar si los modelos que hemos visto tienen la capacidad de distinguir grafos que estructuralmente son equivalentes. La idea es que la expresividad de una red neuronal de grafo está relacionada con su capacidad de distinguir isomorfismos de grafos.

¿Qué es un isomorfismo de grafo?

Existe un isomorfismo entre dos grafos si tienen la misma estructura, esto es, tienen idénticas conexiones pero una permutación en el etiquetado de los nodos.



Existe un test denominado test de Weisfeiler-Lehman el cual nos puede decir si dos grafos **no son isomórficos**. El test no da garantías de isomorfismo pero si de no isomorfismo.

Test de Weisfeiler-Lehman

Veamos algunas definiciones.

Isomorfismo de grafos: Dos grafos son considerados isomórficos si existe un mapping entre los nodos de ambos grafos que preserva la adyacencia de los nodos.

Test de isomorfismo: problema difícil, no se conoce un algoritmo de tiempo polinomial que determine si dos grafos son isomórficos. Tampoco se sabe si es NP-completo. Se dice que es un ejemplo de problema NP-intermedio.

The Weisfeiler-Lehman test: se basa en la reducción a la forma canónica de ambos grafos. *Si ambos grafos difieren en sus formas canónicas, entonces no son isomórficos*.

El reverso no es cierto. Dos grafos no isomórficos podrían compartir la misma forma canónica. Es decir, si el test reduce ambos grafos a la misma forma canónica, no podemos decir nada.

Test de Weisfeiler-Lehman

Test (invariante para obtener forma canónica)

- En la iteración *i* del test asignaremos a cada nodo una tupla $L_{i,n}$ que contiene las etiquetas que el nodo ha tenido en las iteraciones anteriores (*compressed label*) y un multiset[1] con los *compressed labels* de sus vecinos.

- En cada iteración *i*, a cada nodo *n* se le asignará un nuevo compressed label $C_{i,n}$.

Algoritmo para obtener forma canónica

1. Comenzamos inicializando con $C_{0,n} = 1$ a todos los nodos.

2. En la iteración *i* del algoritmo, para cada nodo *n*, asignamos a $L_{i,n}$ la tupla que contiene las etiquetas anteriores del nodo, es decir, $C_{i-1,n}$, y el multiset $C_{i-1,m}$, donde *m* se mueve entre los vecinos de *n*.

3. Completamos la iteración *i* haciendo que $C_{i,n}$ sea un nuevo compressed label, usando el hash $L_{i,n}$. Todo par de nodos con el mismo $L_{i,n}$ debe provenir de un mismo $C_{i,n}$.

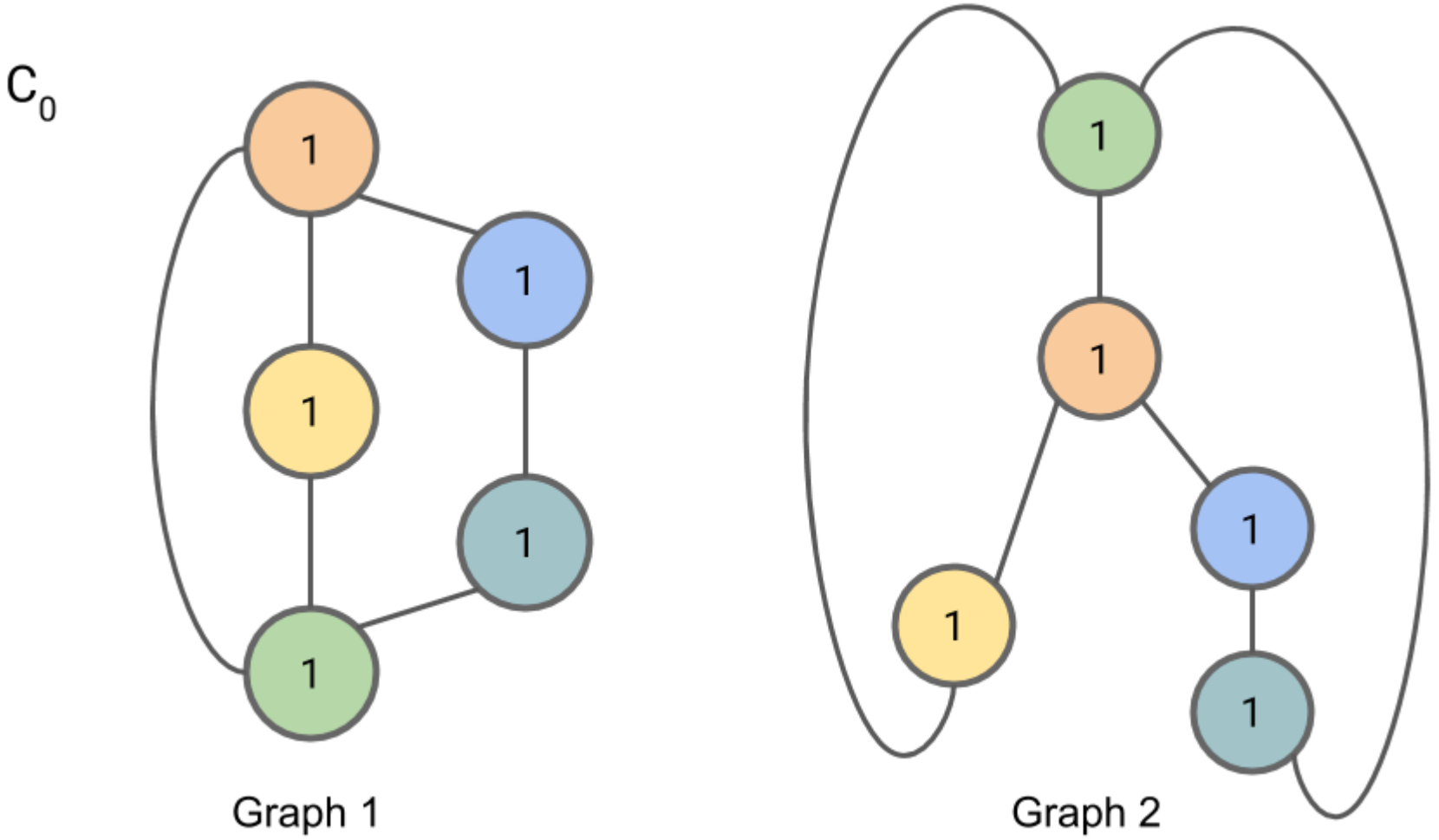
4. Particionaremos los nodos del grafo según su *compressed label*. Terminamos si $i = N$ o bien si no hay cambios en la partición de nodos entre dos iteraciones consecutivas.

[1] Un multiset es un conjunto en el cual se pueden repetir los elementos

Test de Weisfeiler-Lehman

Algoritmo para obtener forma canónica

1. Comenzamos inicializando con $C_{0,n} = 1$ a todos los nodos.

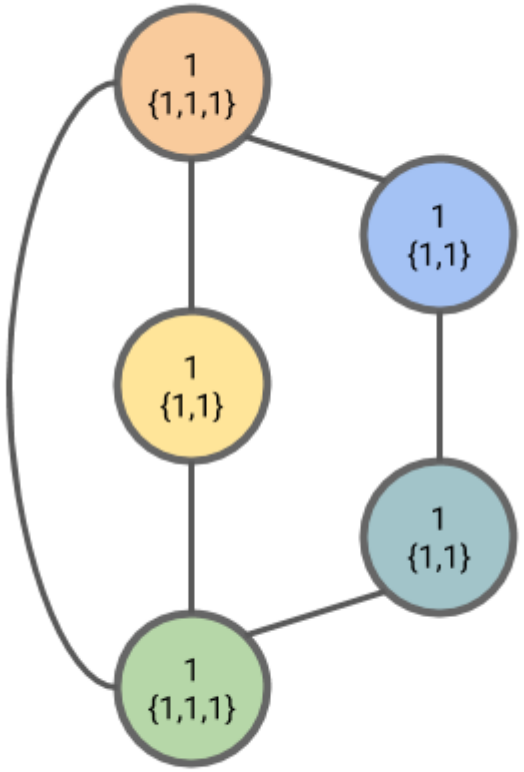


Test de Weisfeiler-Lehman

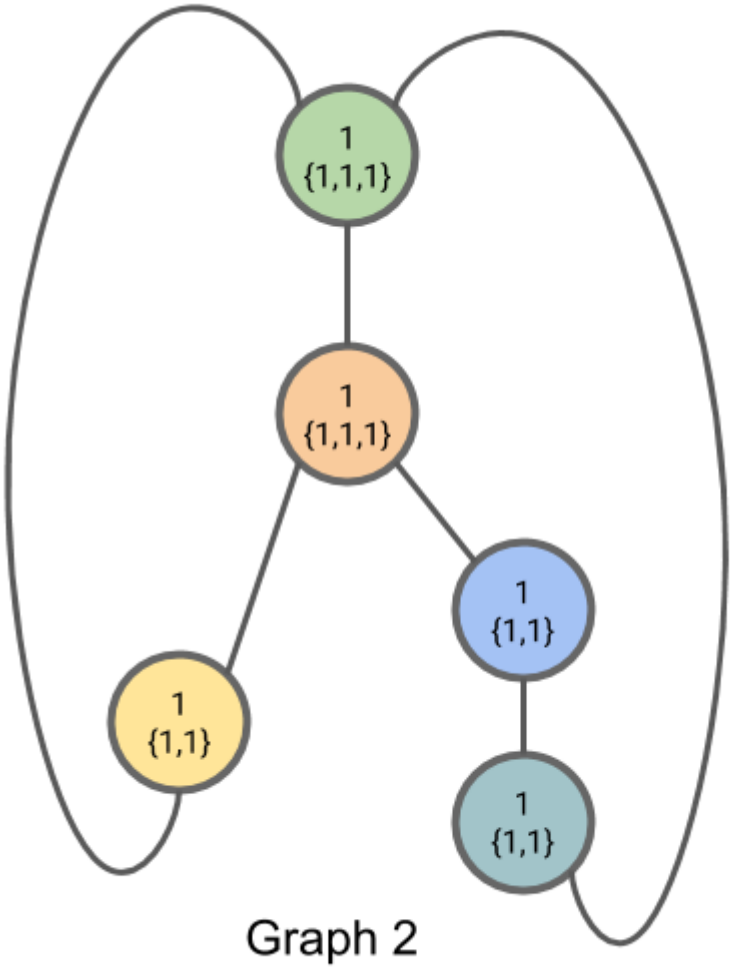
Algoritmo para obtener forma canónica

2. En la iteración *1* del algoritmo, para cada nodo *n*, asignamos a $L_{i,n}$ la tupla que contiene las etiquetas anteriores del nodo, es decir, $C_{i-1,n}$, y el multiset $C_{i-1,m}$, donde *m* se mueve entre los vecinos de *n*.

L_1



Graph 1



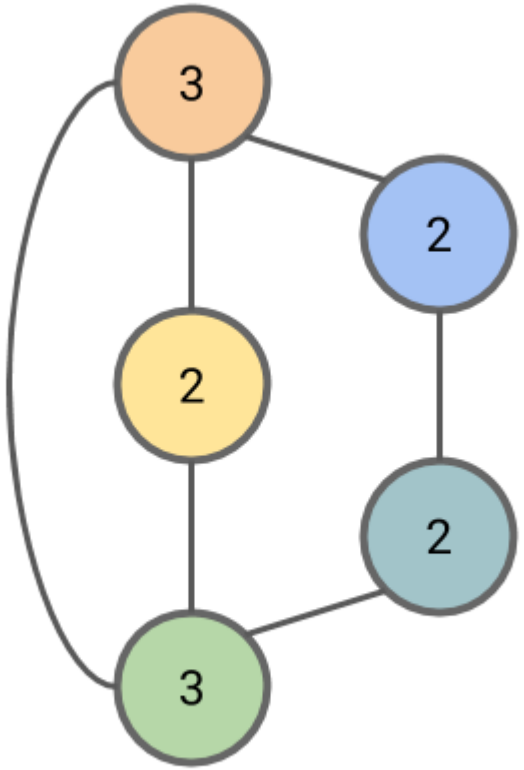
Graph 2

Test de Weisfeiler-Lehman

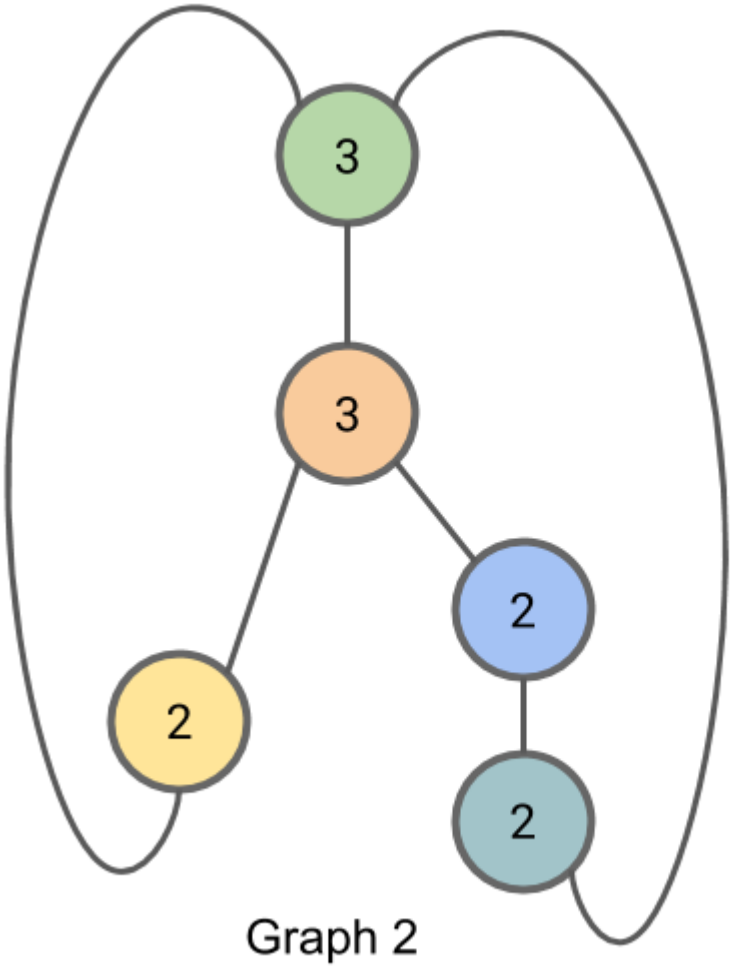
Algoritmo para obtener forma canónica

3. Completamos la iteración *1* haciendo que $C_{i,n}$ sea un nuevo compressed label, usando el hash $L_{i,n}$. Todo par de nodos con el mismo $L_{i,n}$ debe provenir de un mismo $C_{i,n}$.

C_1



Graph 1

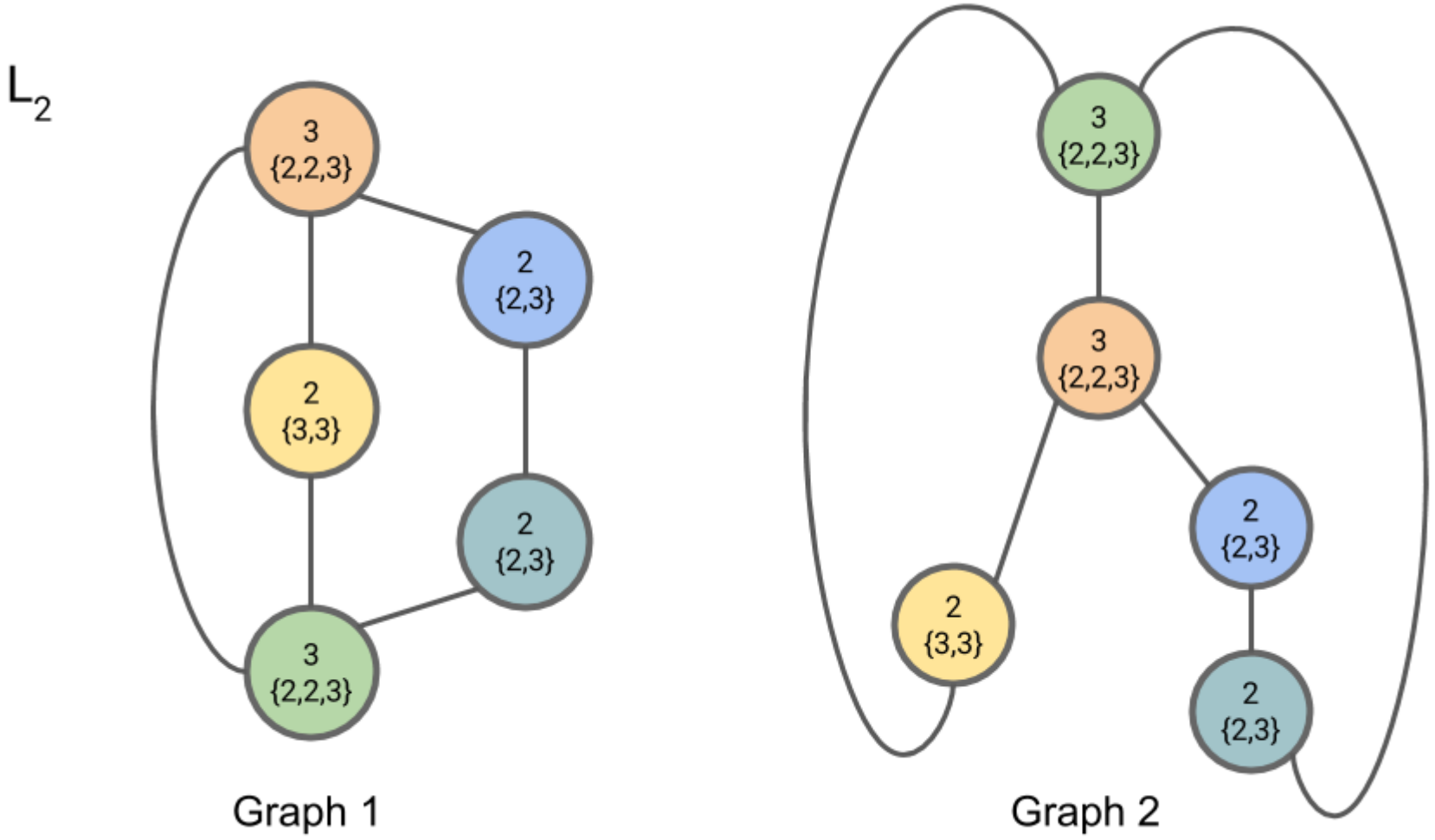


Graph 2

Test de Weisfeiler-Lehman

Algoritmo para obtener forma canónica

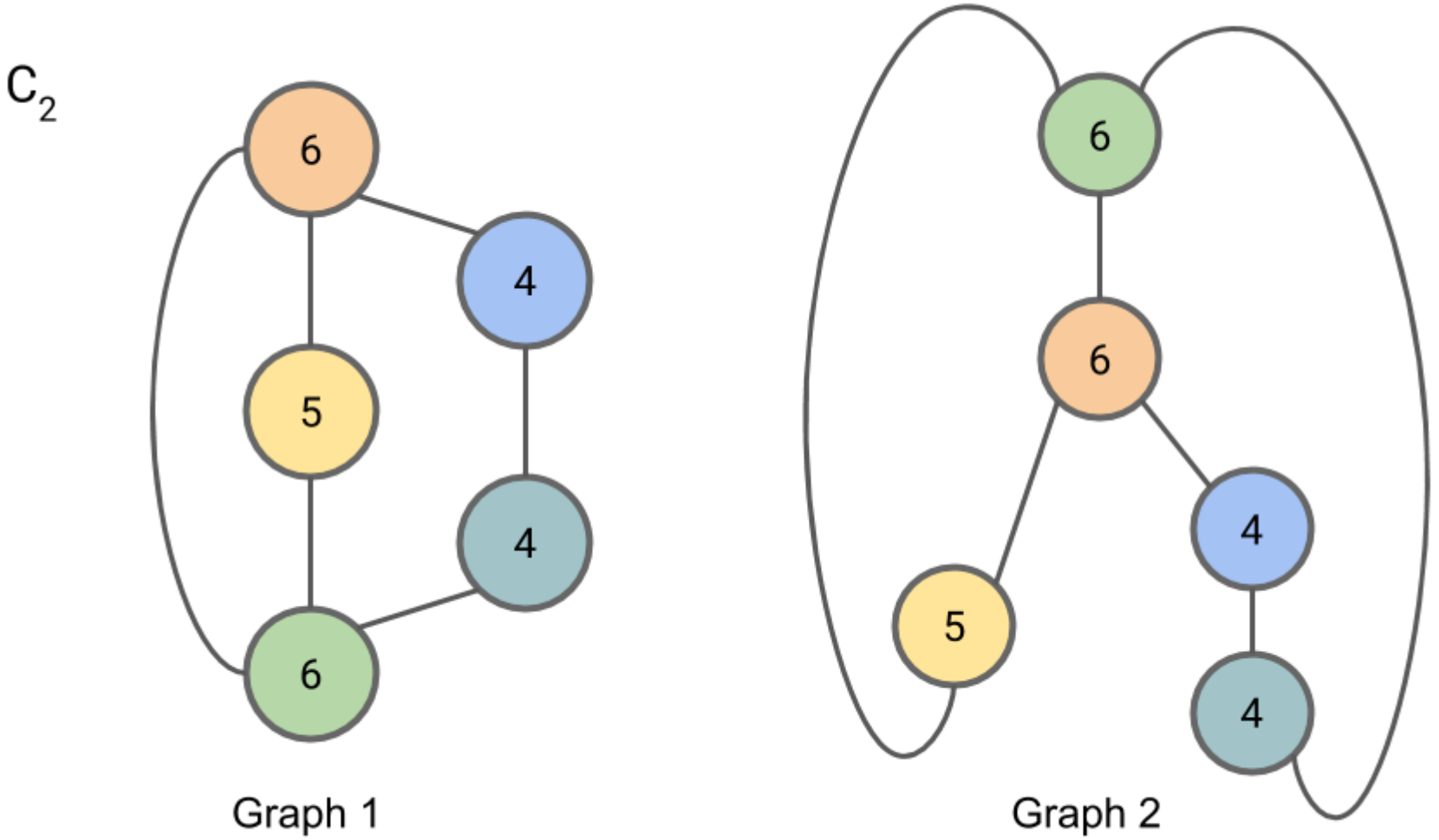
2. En la iteración *2* del algoritmo, para cada nodo *n*, asignamos a $L_{i,n}$ la tupla que contiene las etiquetas anteriores del nodo, es decir, $C_{i-1,n}$, y el multiset $C_{i-1,m}$, donde *m* se mueve entre los vecinos de *n*.



Test de Weisfeiler-Lehman

Algoritmo para obtener forma canónica

3. Completamos la iteración *2* haciendo que $C_{i,n}$ sea un nuevo compressed label, usando el hash $L_{i,n}$. Todo par de nodos con el mismo $L_{i,n}$ debe provenir de un mismo $C_{i,n}$.

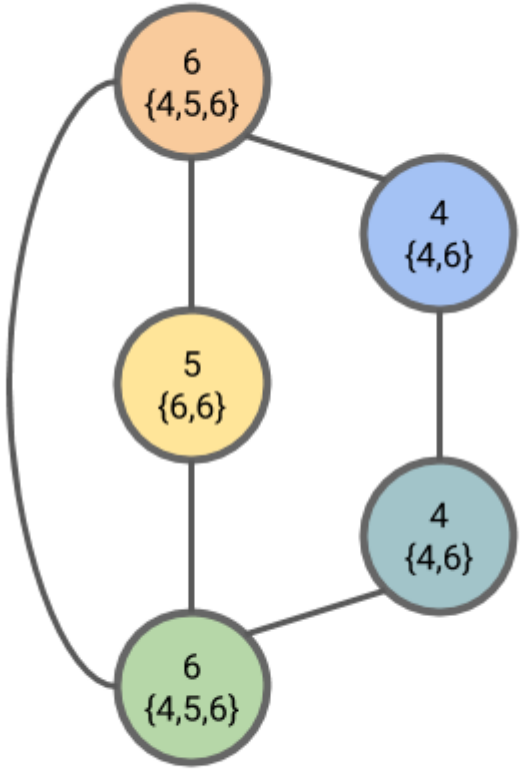


Test de Weisfeiler-Lehman

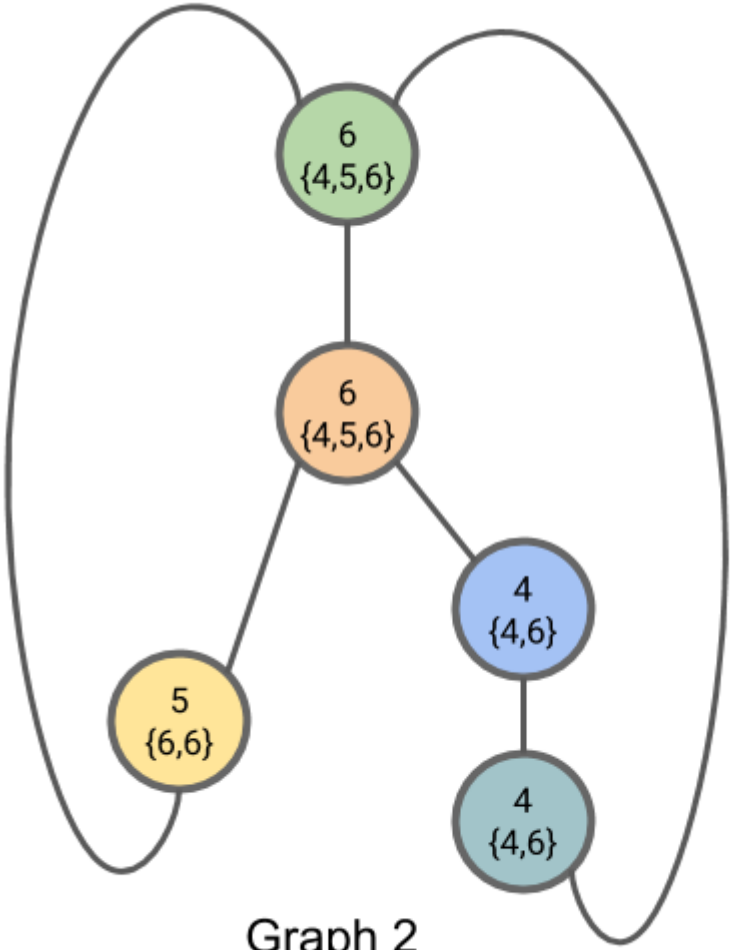
Algoritmo para obtener forma canónica

2. En la iteración *3* del algoritmo, para cada nodo *n*, asignamos a $L_{i,n}$ la tupla que contiene las etiquetas anteriores del nodo, es decir, $C_{i-1,n}$, y el multiset $C_{i-1,m}$, donde *m* se mueve entre los vecinos de *n*.

L_3



Graph 1



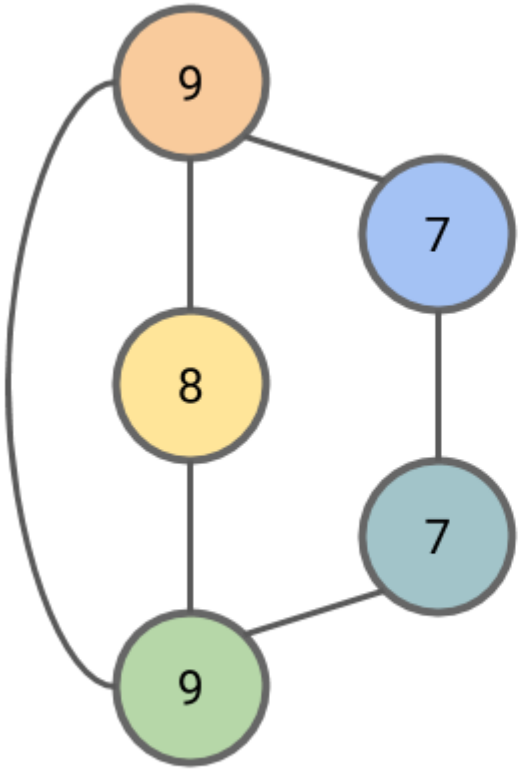
Graph 2

Test de Weisfeiler-Lehman

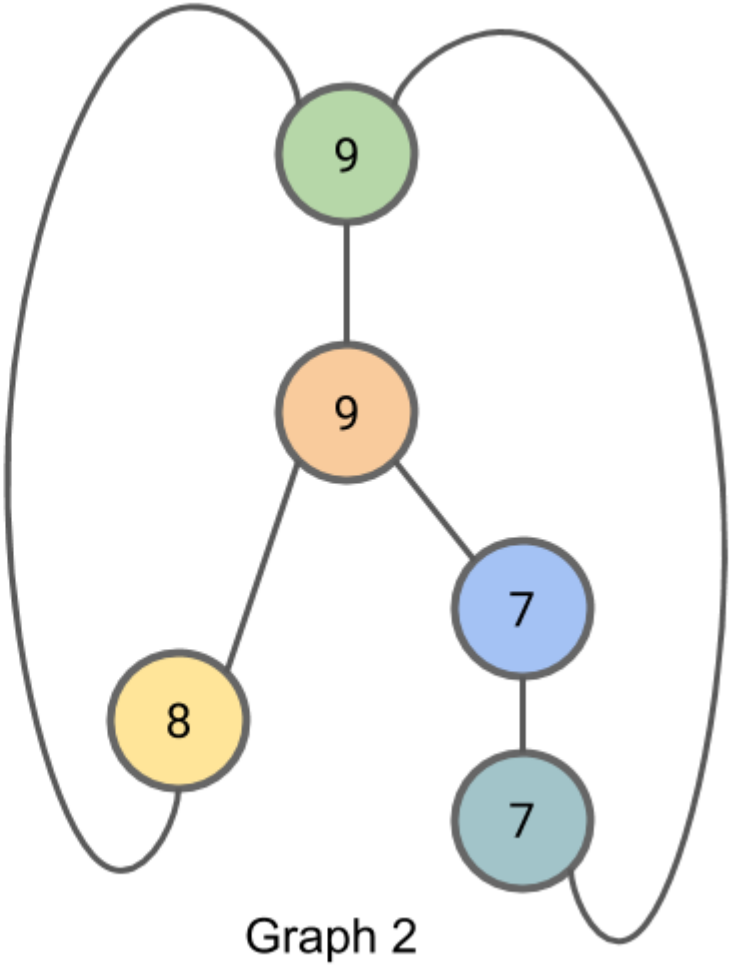
Algoritmo para obtener forma canónica

3. Completamos la iteración *3* haciendo que $C_{i,n}$ sea un nuevo compressed label, usando el hash $L_{i,n}$. Todo par de nodos con el mismo $L_{i,n}$ debe provenir de un mismo $C_{i,n}$.

C_3



Graph 1



Graph 2

Observemos que no han habido cambios entre C_2 y C_3, entonces el algoritmo termina.
Observamos que los mappings son equivalentes: 2 7s, 1 8, 2 9s.

Graph Isomorphic Networks (GIN)

Las redes GIN usan un agregador a nivel de nodos que **imita** el WL test. El efecto de usar agregación basado en el WL test es que la red produzca embeddings de nodos distintos si los grafos no son isomorfos.

Seguiremos el razonamiento del paper. Las condiciones para que una GNN mapee a embeddings distintos dos grafos G1 y G2 que el test WL decide que no son isomórficos son:

a) La red calcula iterativamente los embeddings de los nodos según:

$$h_v^{(k)} = \phi\left(h_v^{(k-1)}, f\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)\right)$$

donde tanto *f* como ϕ deben ser funciones inyectivas (proyección única para cada elemento).

b) El graph read-out de la red, que opera sobre un multiset de nodos $\left\{h_v^{(k)}\right\}$, debe ser inyectivo.

Xu, K., Hu, W., Leskovec, J., Jegelka, S. How powerful are graph neural networks?, ICLR 2019.
https://arxiv.org/pdf/1810.00826v3

Graph Isomorphic Networks (GIN)

Asumamos que el espacio de características de la entrada $\mathcal{X}$ es contable. Esto es importante ya que debemos estudiar la inyectividad y en estos espacios esta propiedad está bien caracterizada.

Lema (ver demo en paper): Existe una función $f : \mathcal{X} \to \mathbb{R}^n$ tal que para infinitas elecciones de epsilon, se cumple que:

$$h(c, X) = (1 + \epsilon) \cdot f(c) + \sum_{x \in X} f(x)$$

es único para cada par (c, X), donde $c \in \mathcal{X}$ y $X \subset \mathcal{X}$ es un multiset de tamaño acotado. Más aún, cualquier función g sobre tales pares puede descomponerse como:

$$g\,(c, X) = \varphi\left((1 + \epsilon) \cdot f(c) + \sum_{x \in X} f(x)\right)$$

para alguna función φ.

👉 Xu, K., Hu, W., Leskovec, J., Jegelka, S. How powerful are graph neural networks?, ICLR 2019. https://arxiv.org/pdf/1810.00826v3

Graph Isomorphic Networks (GIN)

En base a estos resultados, GIN propone aprender *f* y φ en base al teorema de aproximación universal de MLPs[1]. En la práctica, se modela la composición de ambas funciones $f^{(k+1)} \circ \varphi^{(k)}$ con una MLP, dado que **la MLP puede aprender la composición de funciones**.

En la primera iteración, no necesitamos una MLP antes de la suma de características de entrada, ya que si las features son one hot encodings, por definición estas sumas son inyectivas. Además, epsilón se puede aprender. Luego, la función de update de nodos de GIN se define según:

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right)\cdot h_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right)$$

[1] Hornik, K., Stinchcombe, M., White, H. Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359-366, 1989.

Graph Isomorphic Networks (GIN) – Graph read out

GIN puede generar mediante un graph read out un embedding para el grafo completo. El graph read out puede operar a nivel de un subconjunto de los nodos del grafo original.

Los autores de GIN observan que los embeddings de todas las capas de la red aportan información (algo parecido a lo que ocurre con BERT). Entonces, proponen que el graph read out se obtenga concatenando las representaciones obtenidas en todas las capas:

$$h_G = \mathrm{CONCAT}\Big(\mathrm{READOUT}\Big(\Big\{h_v^{(k)} | v \in G\Big\}\Big) \mid k = 0, 1, \ldots, K\Big)$$

En esta ecuación, la función de READOUT puede reeeplazarse por la suma, caso en el cual GIN probablemente cumpla con el WL test.

Algunos aspectos prácticos: GIN tiene suficiente capacidad expresiva con MLP de al menos dos capas. Con MLP de una capa no funciona.

GIN muestra que gracias a su inyectividad sobre grafos, es adecuado para tareas de clasificación de grafos. Es decir, es un modelo que permite hacer las tres tareas de GML (clasificación de nodos, predicción de enlaces y clustering).