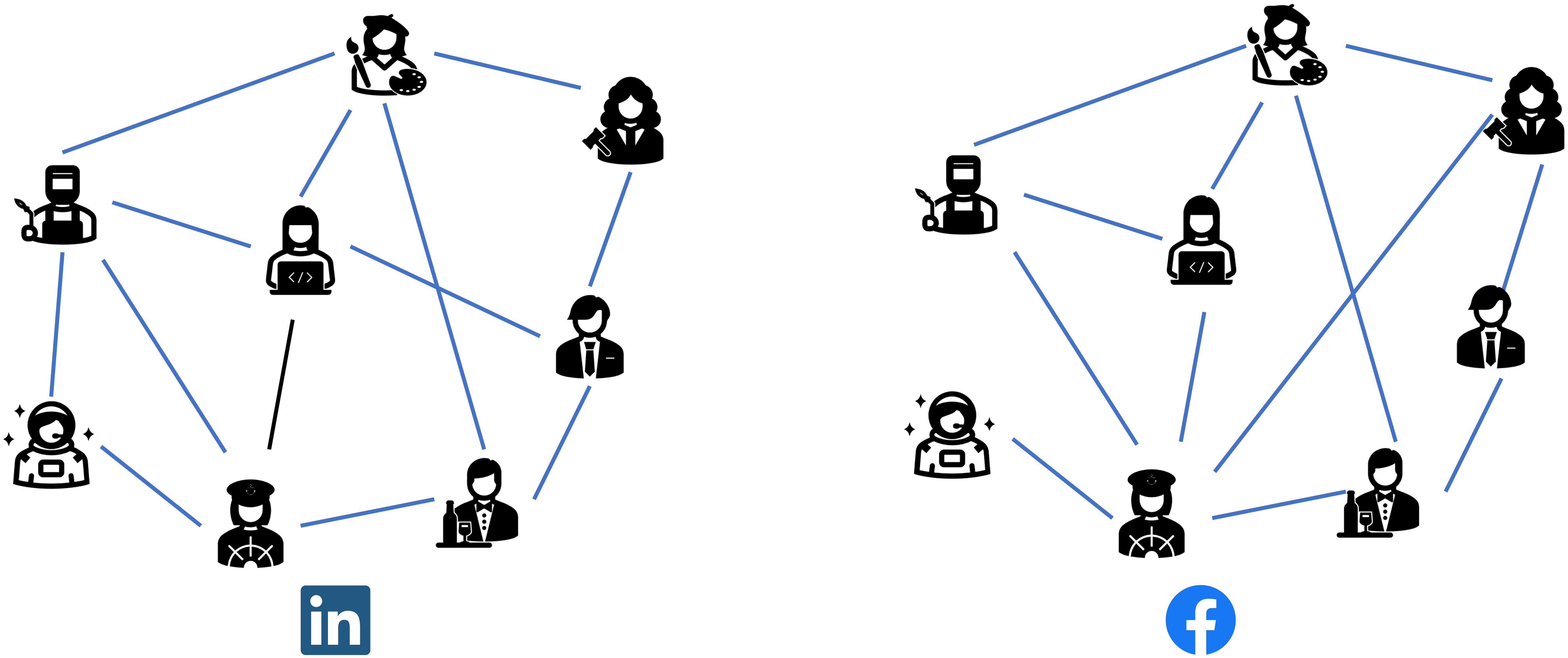
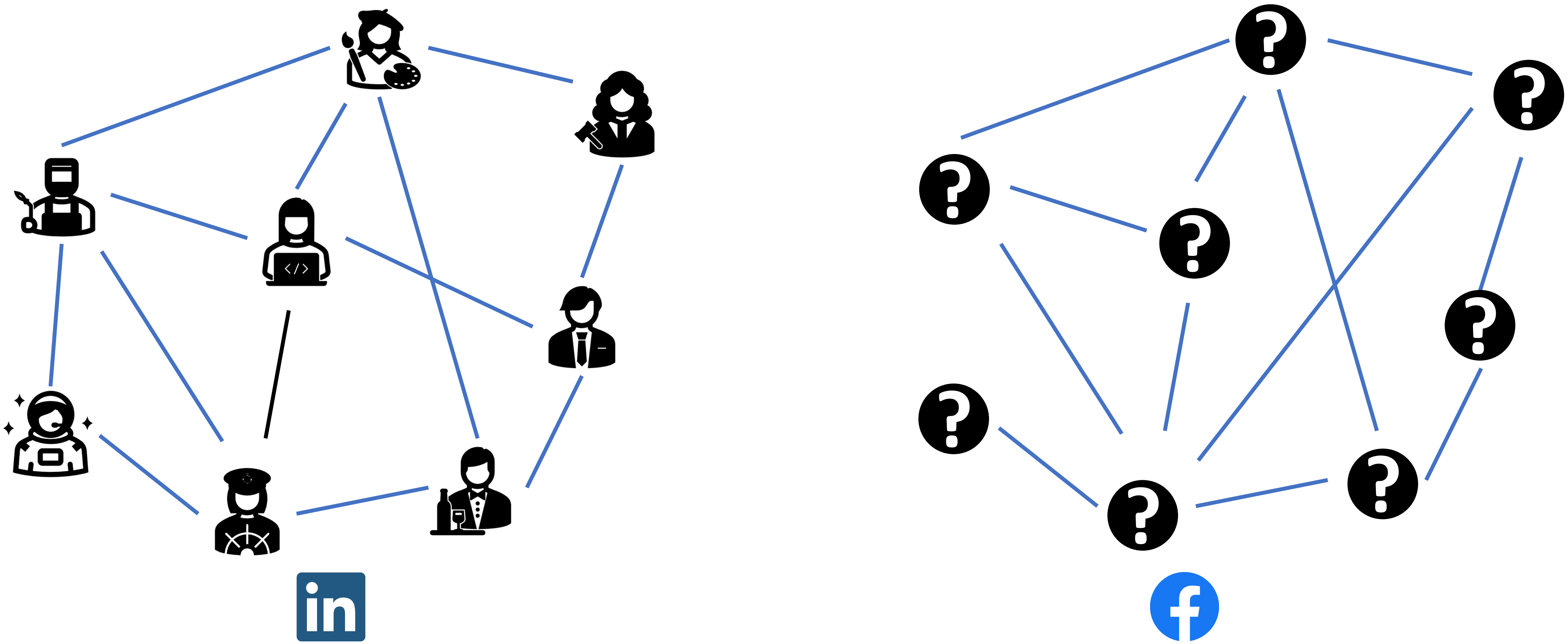
Existing Graph Alignment Algorithms

Ning Zhang

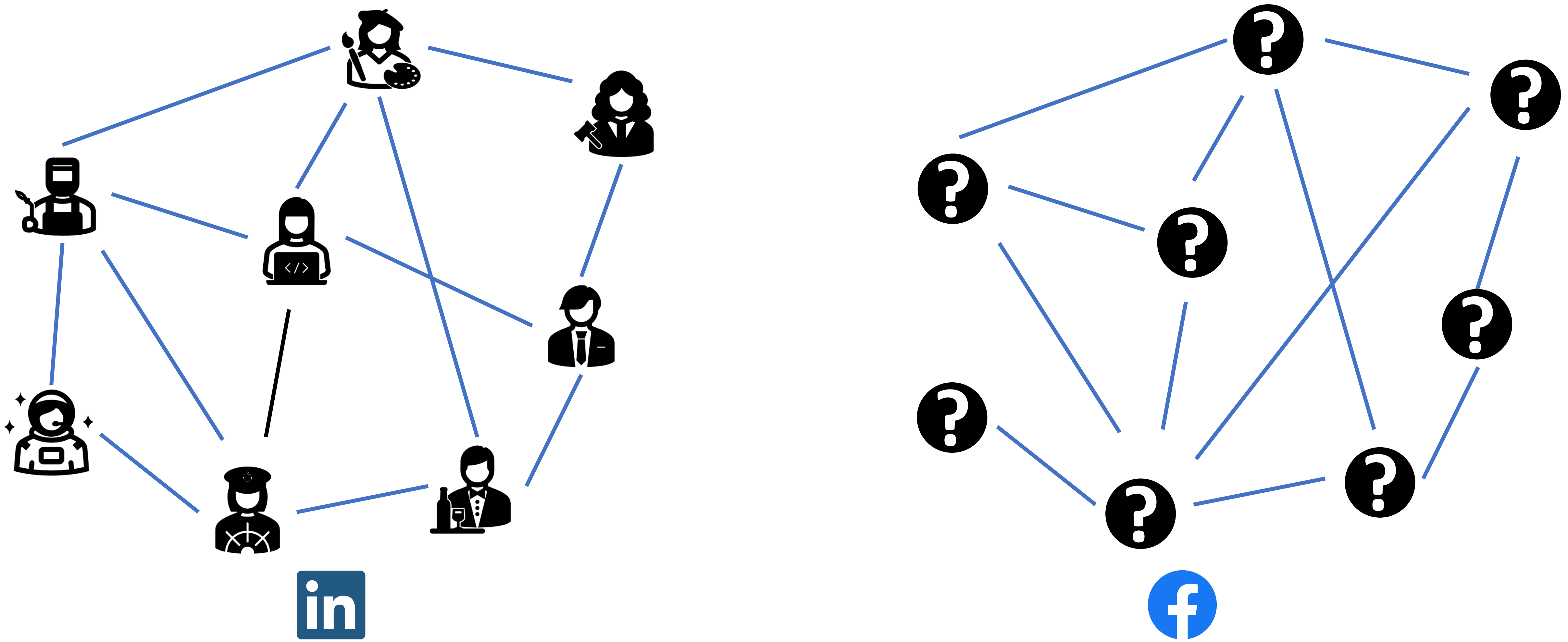
Social network deanonymization



Social network deanonymization



Social network deanonymization



Graph alignment: Find a **one-to-one correspondence** for users in the two “similar” networks

Graph Isomorphism

In graph theory, an **isomorphism** of graphs G and H is a bijection between the vertex sets

$$f: V(G) \to V(H)$$

s.t. any vertices u and v of G are adjacent in G iff $f(u)$ and $f(v)$ are adjacent in H .

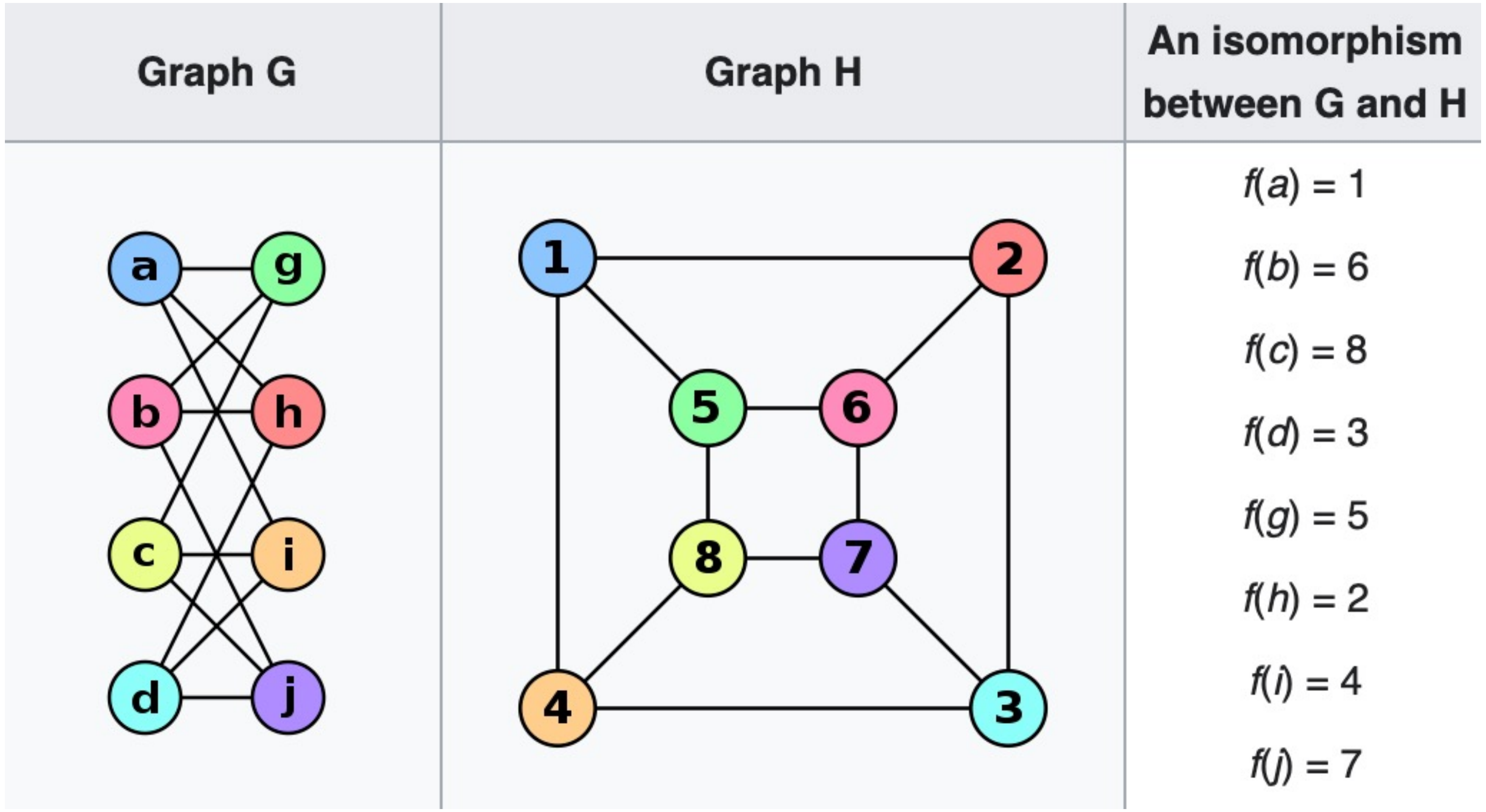


Figure source: wikipedia

Graph Isomorphism

Canonical labeling: assigning a unique label to each vertex such that the labels are invariant under isomorphism.

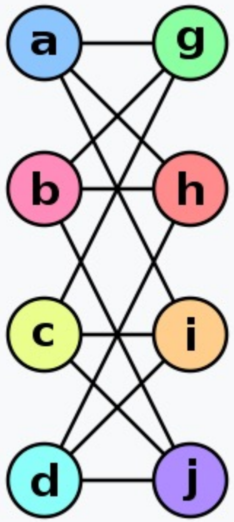
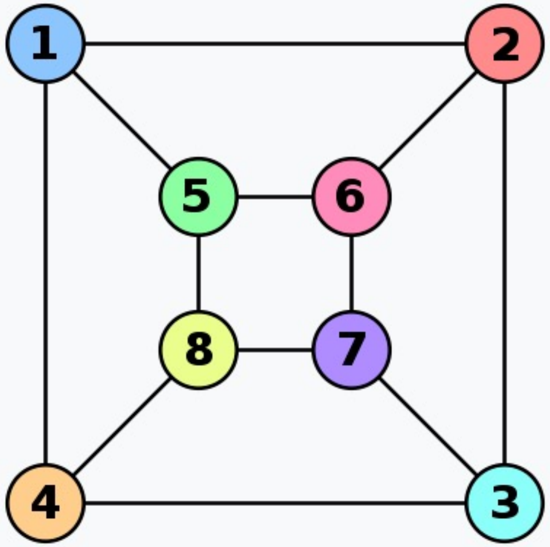
Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

Figure source: wikipedia

1. Graph Isomorphism

Canonical labeling: assigning a unique label to each vertex such that the labels are invariant under isomorphism.

Algorithms:

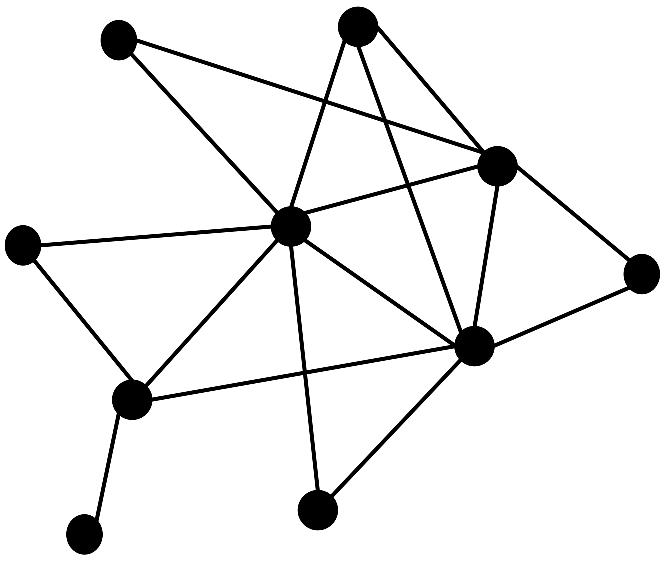
(1.1)LABEL Algorithm[1]

(1.2)Canonical labeling algorithm[2]

1.1 LABEL Algorithm[1]

Input: graph G and degree threshold L

e.g. L =3

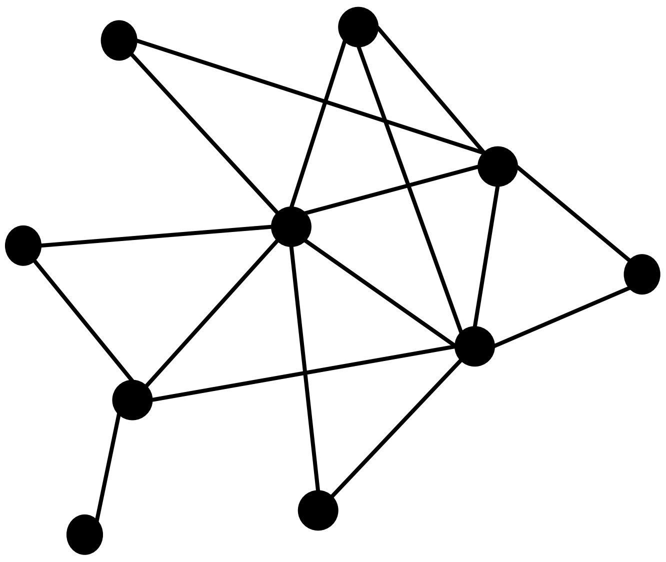


G

1.1 LABEL Algorithm[1]

Step1: Relabel the vertices of G so that they satisfy

$$d_G(v_1) \geq d_G(v_2) \geq \cdots \geq d_G(v_n)$$



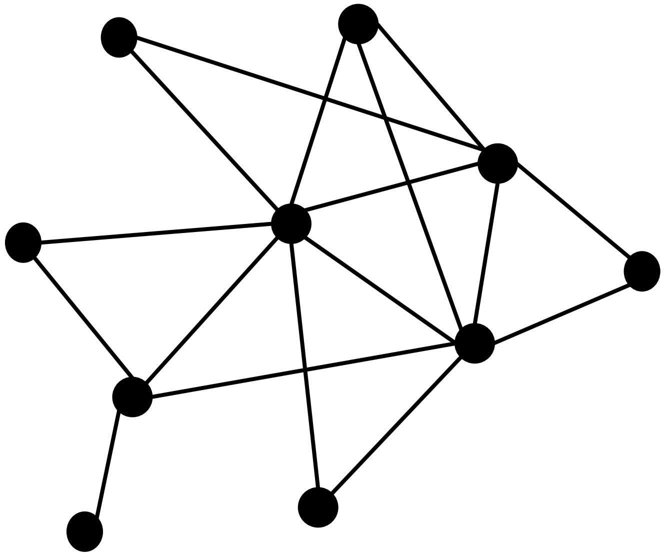
G

1.1 LABEL Algorithm[1]

Step1: Relabel the vertices of G so that they satisfy

$$d_G(v_1) \geq d_G(v_2) \geq \cdots \geq d_G(v_n)$$

$$7 \geq 6 \geq 5 \geq 4 \geq 3 \geq 2 \geq 2 \geq 2 \geq 2 \geq 1$$



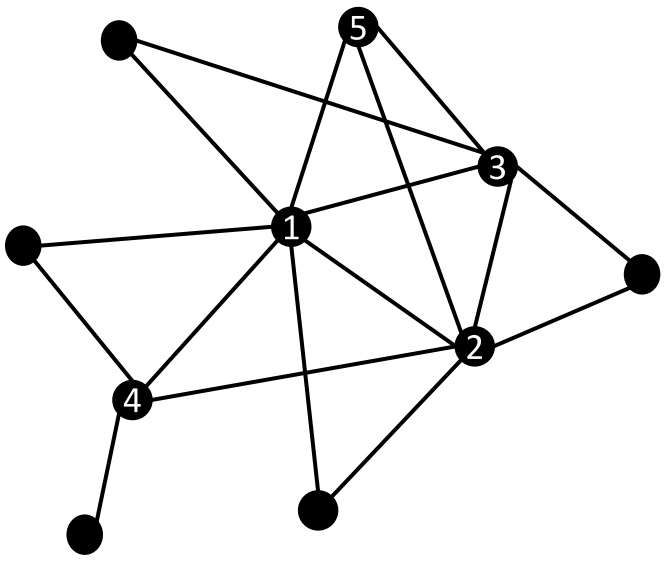
G

Note1: if there exists $i < L$, $d_G(v_i) \geq d_G(v_{i+1})$, then FAIL.

1.1 LABEL Algorithm[1]

Step1: Relabel the vertices of G so that they satisfy

$$d_G(v_1) \geq d_G(v_2) \geq \cdots \geq d_G(v_n)$$



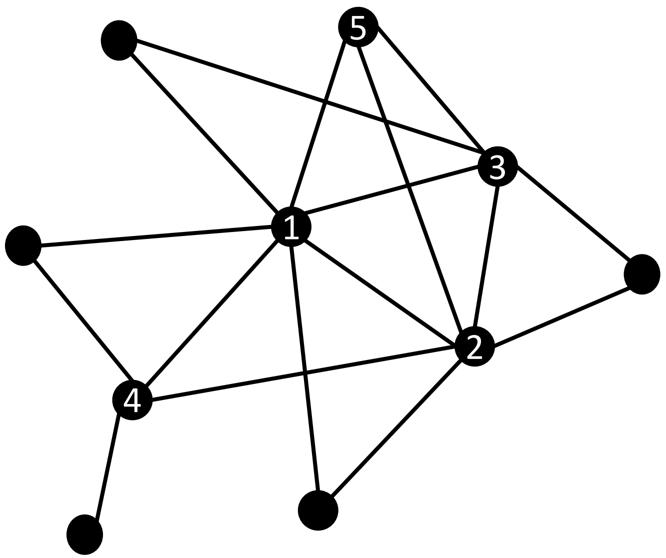
G

1.1 LABEL Algorithm[1]

Step2: For $i > L$, let $X_i = \{j \in \{1,2,\ldots,L\}\colon \{v_i, v_j\} \in E(G)\}$.

Relabel vertices $(v_{L+1}, v_{L+2} \ldots, v_n)$ so that these sets satisfy

$$X_{L+1} \succ X_{L+2} \succ \cdots \succ X_n$$



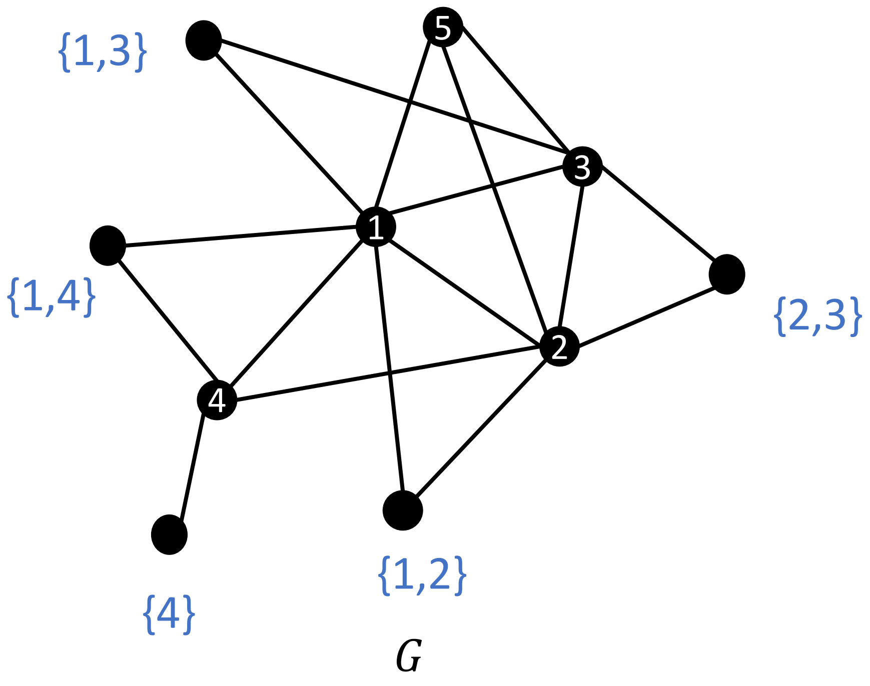
G

1.1 LABEL Algorithm[1]

Step2: For $i > L$, let $X_i = \{j \in \{1,2,\ldots,L\}: \{v_i, v_j\} \in E(G)\}$.

Relabel vertices $(v_{L+1}, v_{L+2} \ldots, v_n)$ so that these sets satisfy

$$X_{L+1} \succ X_{L+2} \succ \cdots \succ X_n$$

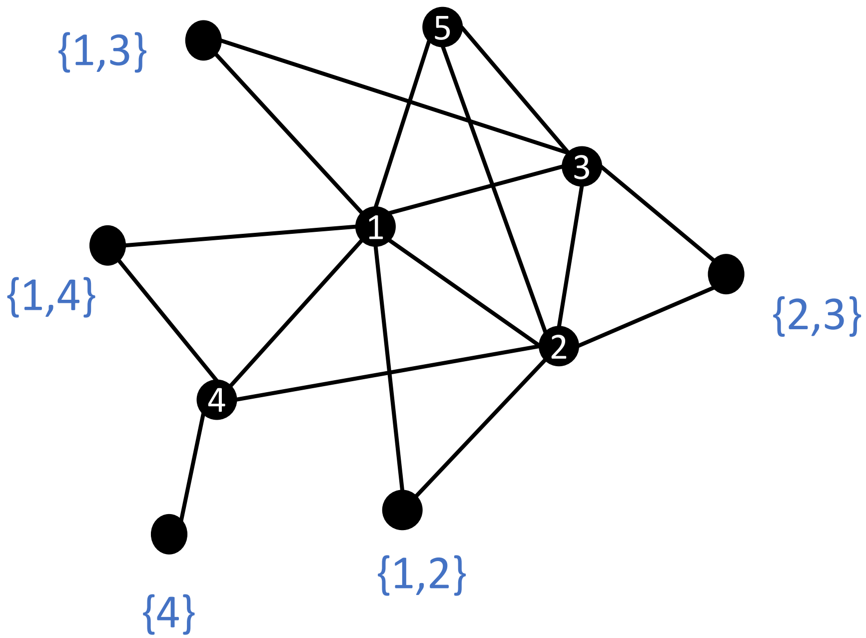


1.1 LABEL Algorithm[1]

Step2: For $i > L$, let $X_i = \{j \in \{1,2,\dots,L\}\colon \{v_i, v_j\} \in E(G)\}$.

Relabel vertices $(v_{L+1}, v_{L+2} \dots, v_n)$ so that these sets satisfy

$$X_{L+1} \succ X_{L+2} \succ \cdots \succ X_n$$



G

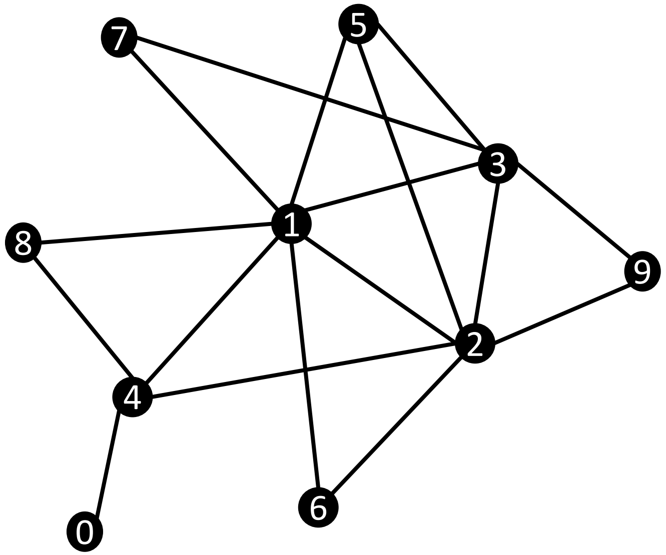
Note2: If there exists $i < n$ such that $X_i = X_{i+1}$ then FAIL

1.1 LABEL Algorithm[1]

Step2: For $i > L$, let $X_i = \{j \in \{1,2,\ldots,L\}\colon \{v_i, v_j\} \in E(G)\}$.

Relabel vertices $(v_{L+1}, v_{L+2} \ldots, v_n)$ so that these sets satisfy

$$X_{L+1} \succ X_{L+2} \succ \cdots \succ X_n$$



G

1.1 LABEL Algorithm[1]

LABEL Algorithm Summary:

Input: graph G and degree threshold L

Step1:

Relabel the vertices of G so that they satisfy

$$d_G(v_1) \geq d_G(v_2) \geq \cdots \geq d_G(v_n)$$

Step2:

For $i > L$, let $X_i = \{j \in \{1,2,\ldots,L\}: \{v_i, v_j\} \in E(G)\}$.

Relabel vertices $(v_{L+1}, v_{L+2} \ldots, v_n)$ so that these sets satisfy

$$X_{L+1} \succ X_{L+2} \succ \cdots \succ X_n$$

1.2 Canonical labeling algorithm[2]

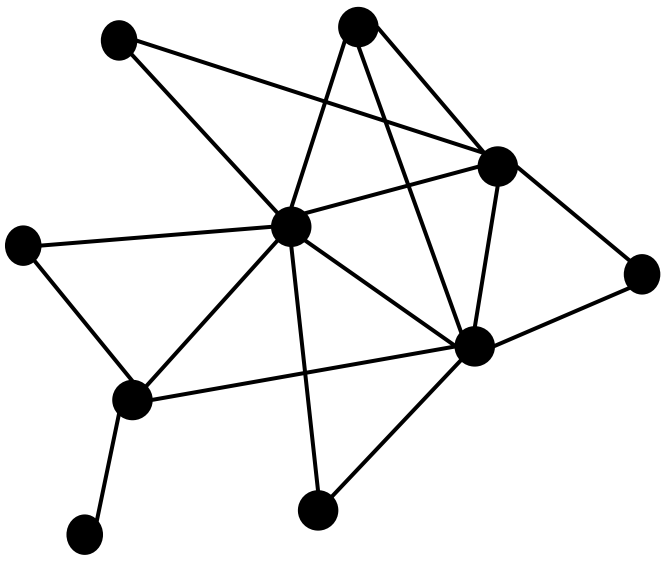
Key idea: distinguish all vertices of a graph using the degrees of their neighbors

degree neighborhood of a vertex: a sorted list of the degrees of the vertex's neighbors

1.2 Canonical labeling algorithm[2]

Step1. Compute vertex degrees

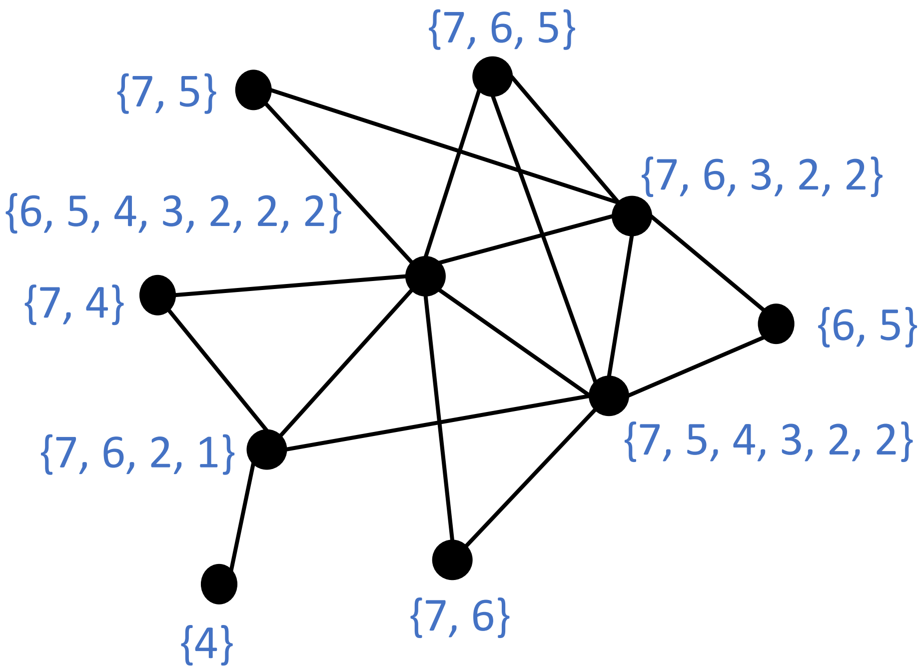
$7, 6, 5,4, 3, 2, 2, 2, 2, 1$



G

1.2 Canonical labeling algorithm[2]

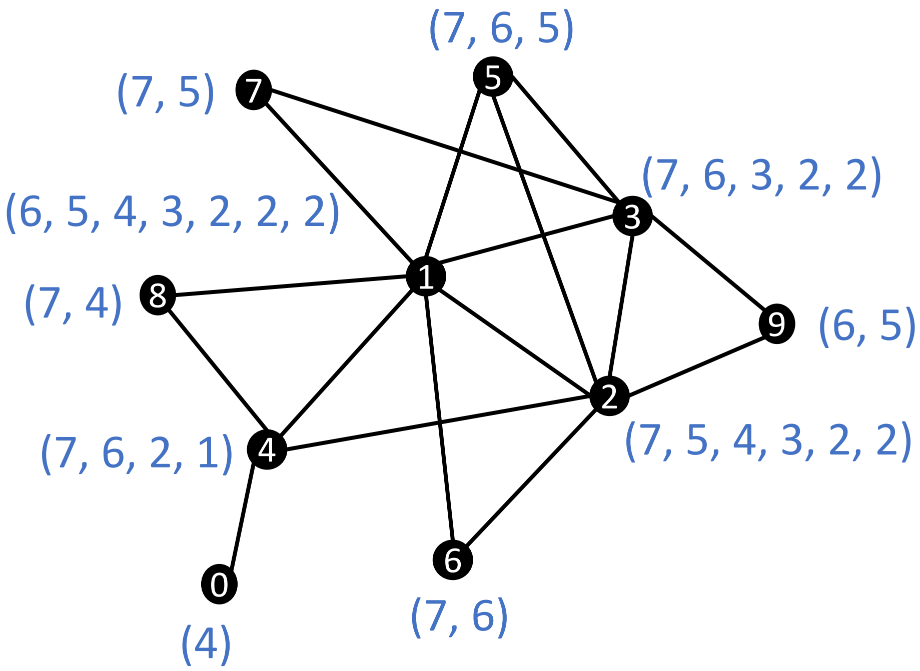
Step2. Compute degree neighborhoods for each vertex.



G

1.2 Canonical labeling algorithm[2]

Step3. Sort vertices by-degree neighborhoods in lexicographical order.



G

Note: If the degree neighborhoods are not distinct for each vertex, FAIL.

1.2 Canonical labeling algorithm[2]

Canonical labeling algorithm Summary:

Step1. Compute vertex degrees.

Step2. Compute degree neighborhoods for each vertex.

Step3. Sort vertices by-degree neighborhoods in lexicographical order.

2. Graph Alignment/Matching

Graph alignment: A noisy version of graph isomorphism, where we seek a **bijection** that minimizes the number of edge disagreements.

2. Graph Alignment/Matching

Graph alignment: A noisy version of graph isomorphism, where we seek a **bijection** that minimizes the number of edge disagreements.

Graph Alignment Algorithms:

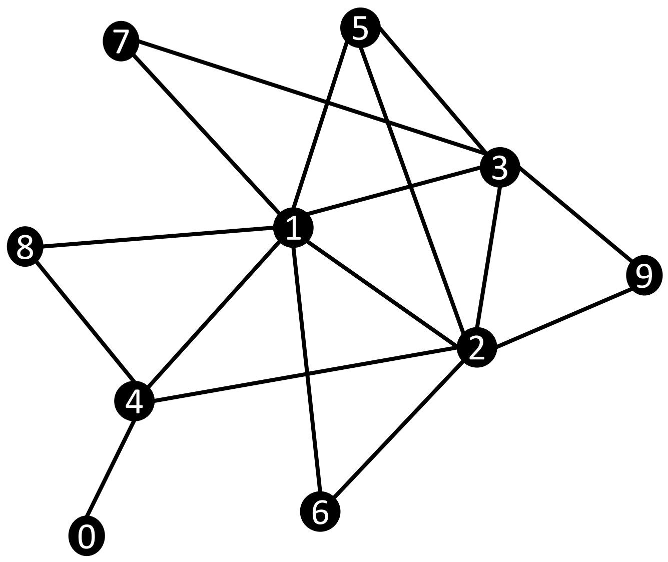
(2.1) Noisy LABEL algorithm [3]

(2.2) Black Swan Algorithm[4]

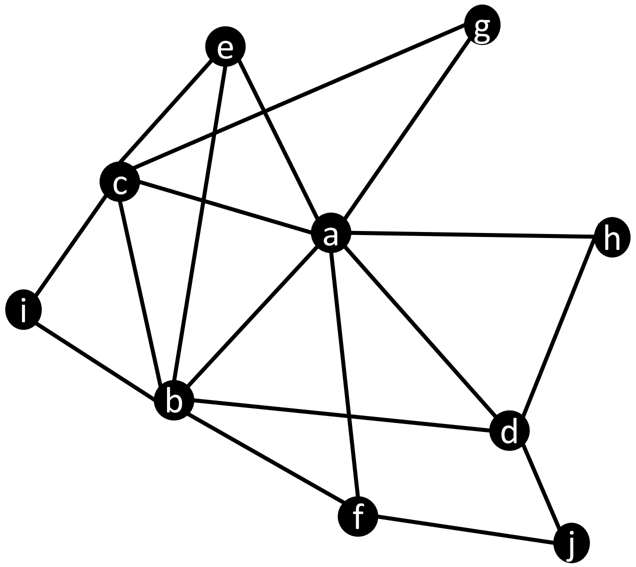
2.1 Noisy LABEL algorithm [3]

Input: Graph $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, and integer h

e.g. $h = 4$



G_1

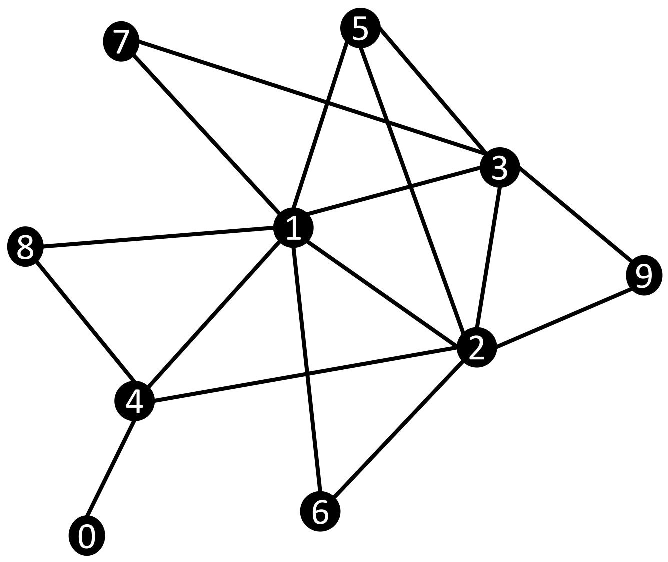


G_2

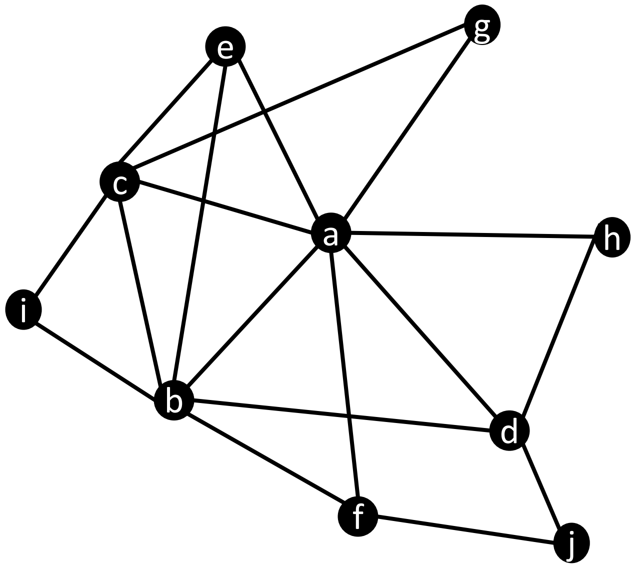
2.1 Noisy LABEL algorithm [3]

Step1: Anchor alignment (match high degree vertices)

$\boldsymbol{w_{G_1}} \in V_1^h, \boldsymbol{w_{G_2}} \in V_2^h$ are h highest degree vertices (decreasing order)



G_1



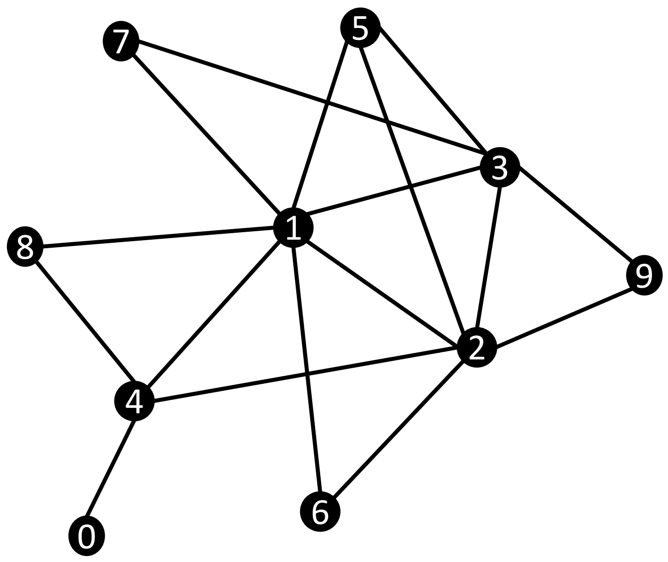
G_2

2.1 Noisy LABEL algorithm [3]

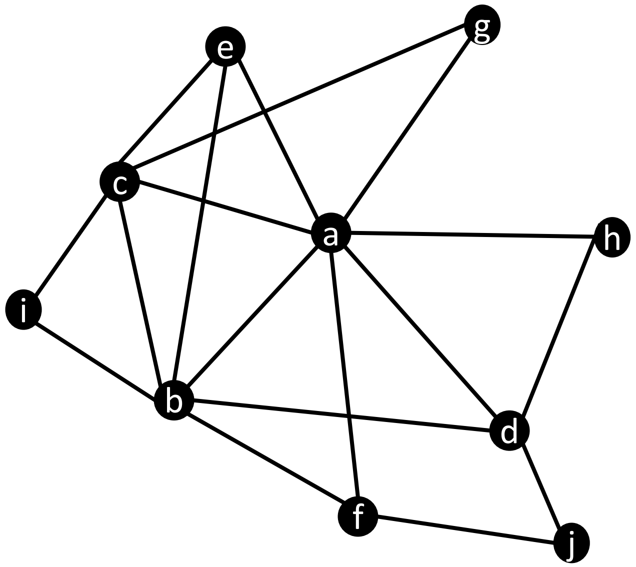
Step1: Anchor alignment (match high degree vertices)

$\boldsymbol{w}_{\boldsymbol{G}_1} \in V_1^h, \boldsymbol{w}_{\boldsymbol{G}_2} \in V_2^h$ are h highest degree vertices (decreasing order)

$$\boldsymbol{w}_{\boldsymbol{G}_1} = (1,2,3,4);\ \boldsymbol{w}_{\boldsymbol{G}_2} = (a, b, c, d)$$



G_1



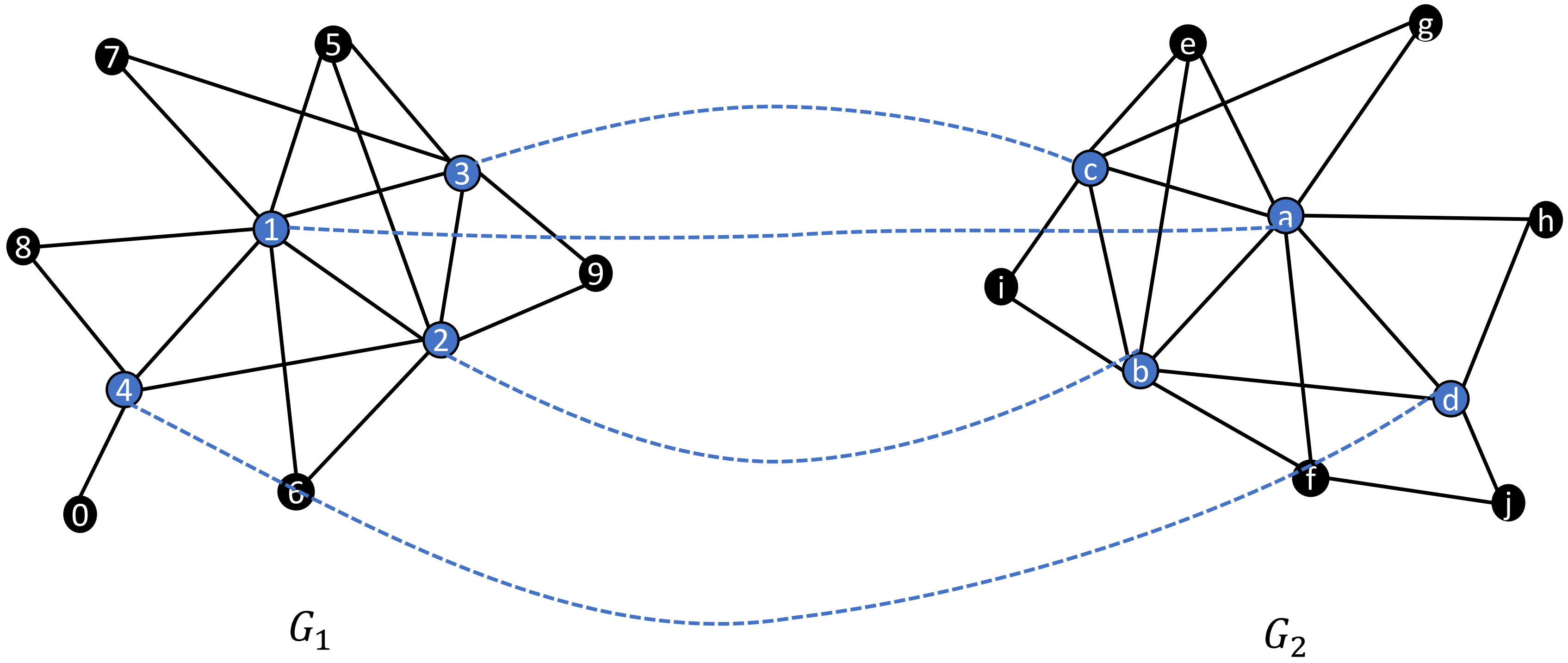
G_2

2.1 Noisy LABEL algorithm [3]

Step1: Anchor alignment (match high degree vertices)

$\boldsymbol{w}_{\boldsymbol{G}_1} \in V_1^h$, $\boldsymbol{w}_{\boldsymbol{G}_2} \in V_2^h$ are h highest degree vertices (decreasing order)

$$\boldsymbol{w}_{\boldsymbol{G}_1} = (1,2,3,4);\ \boldsymbol{w}_{\boldsymbol{G}_2} = (a, b, c, d)$$

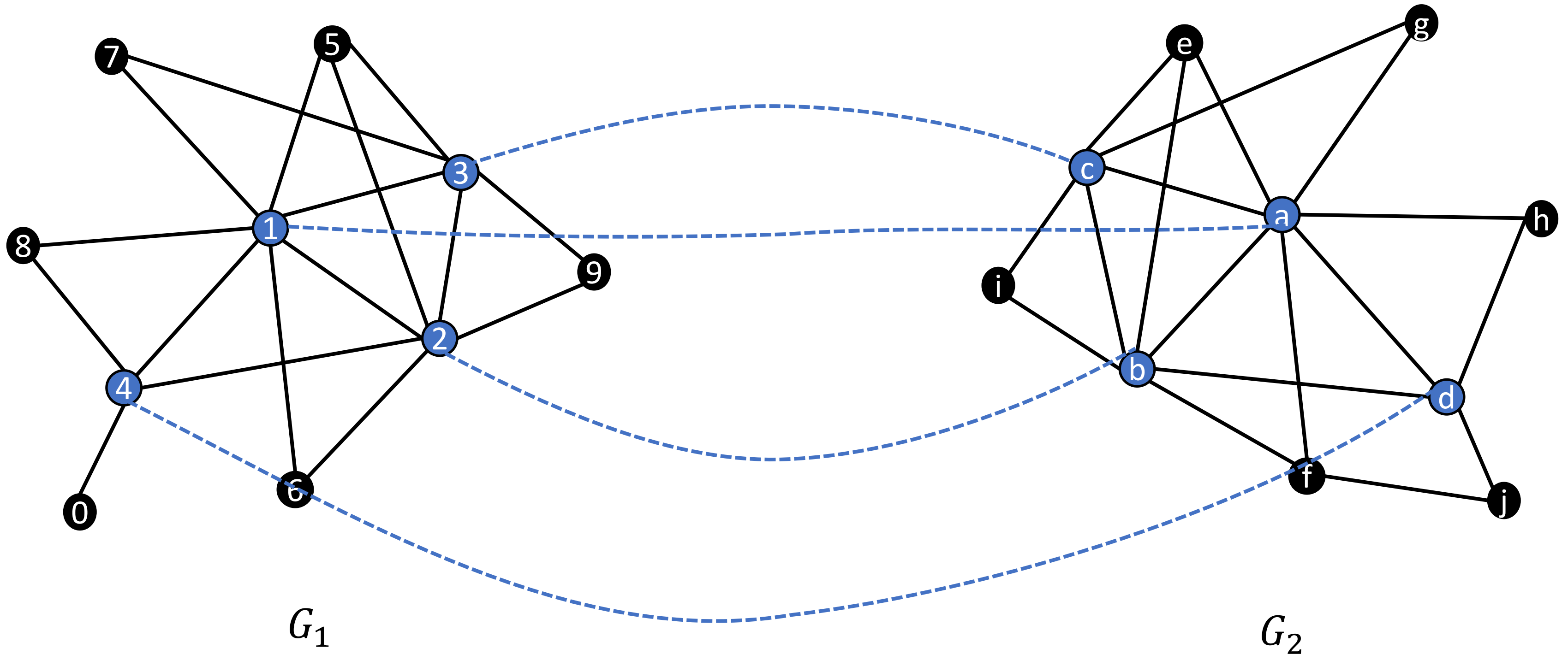


match: 1-a, 2-b, 3-c, 4-d

2.1 Noisy LABEL algorithm [3]

Step2: Bipartite alignment (match low degree vertices)

$\mathrm{sig}_{\mathrm{G}_1}(u), \mathrm{sig}_{\mathrm{G}_2}(u) \in \{0,1\}^h$ where $\mathrm{sig}_G(u)_i = \mathbb{I}\{(\mathrm{u}, \mathrm{w}_G(\mathrm{i})) \in E(G)\}$

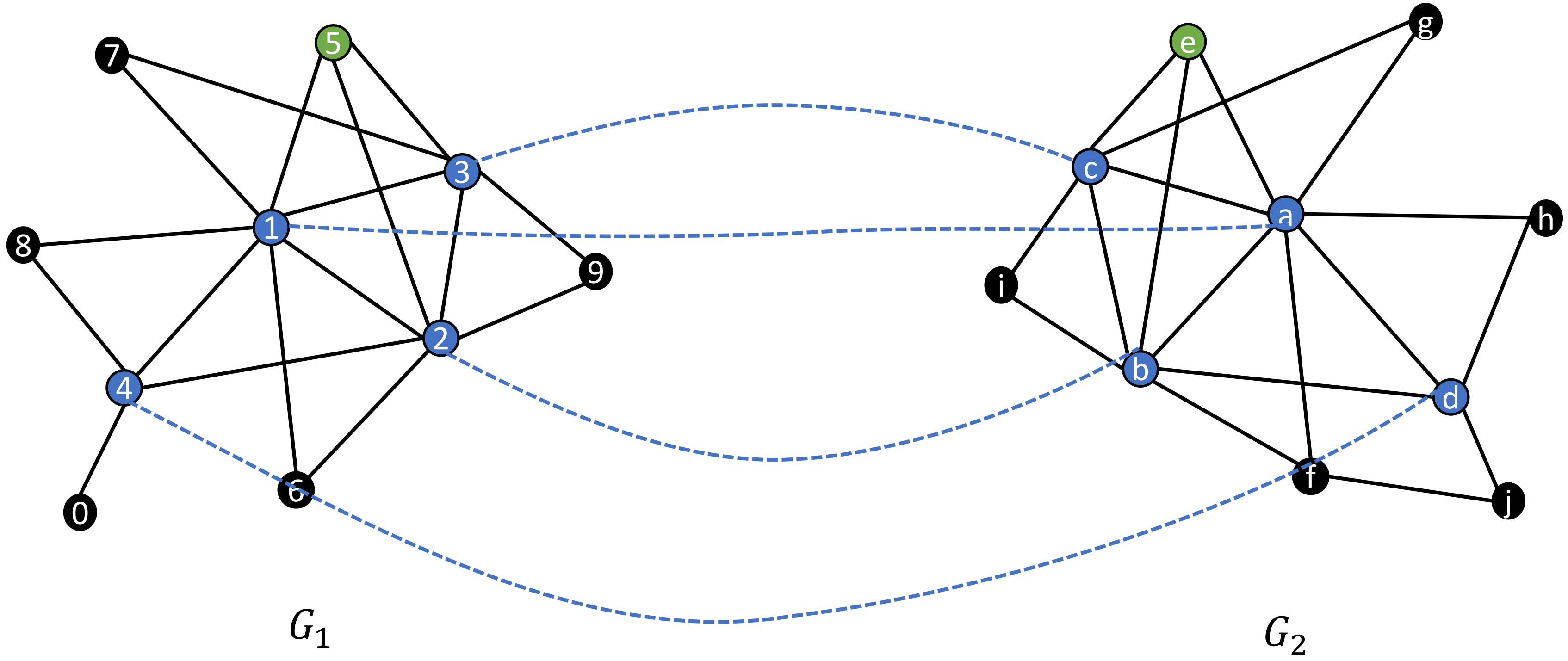


2.1 Noisy LABEL algorithm [3]

Step2: Bipartite alignment (match low degree vertices)

$\mathrm{sig}_{\mathrm{G}_1}(u), \mathrm{sig}_{\mathrm{G}_2}(u) \in \{0,1\}^h$ where $\mathrm{sig}_G(u)_i = \mathbb{I}\{(\mathrm{u}, \mathrm{w}_G(\mathrm{i})) \in E(G)\}$

e.g. $\mathrm{sig}_{\mathrm{G}_1}(5) = (1,1,1,0)$ and $\mathrm{sig}_{\mathrm{G}_1}(e) = (1,1,1,0)$

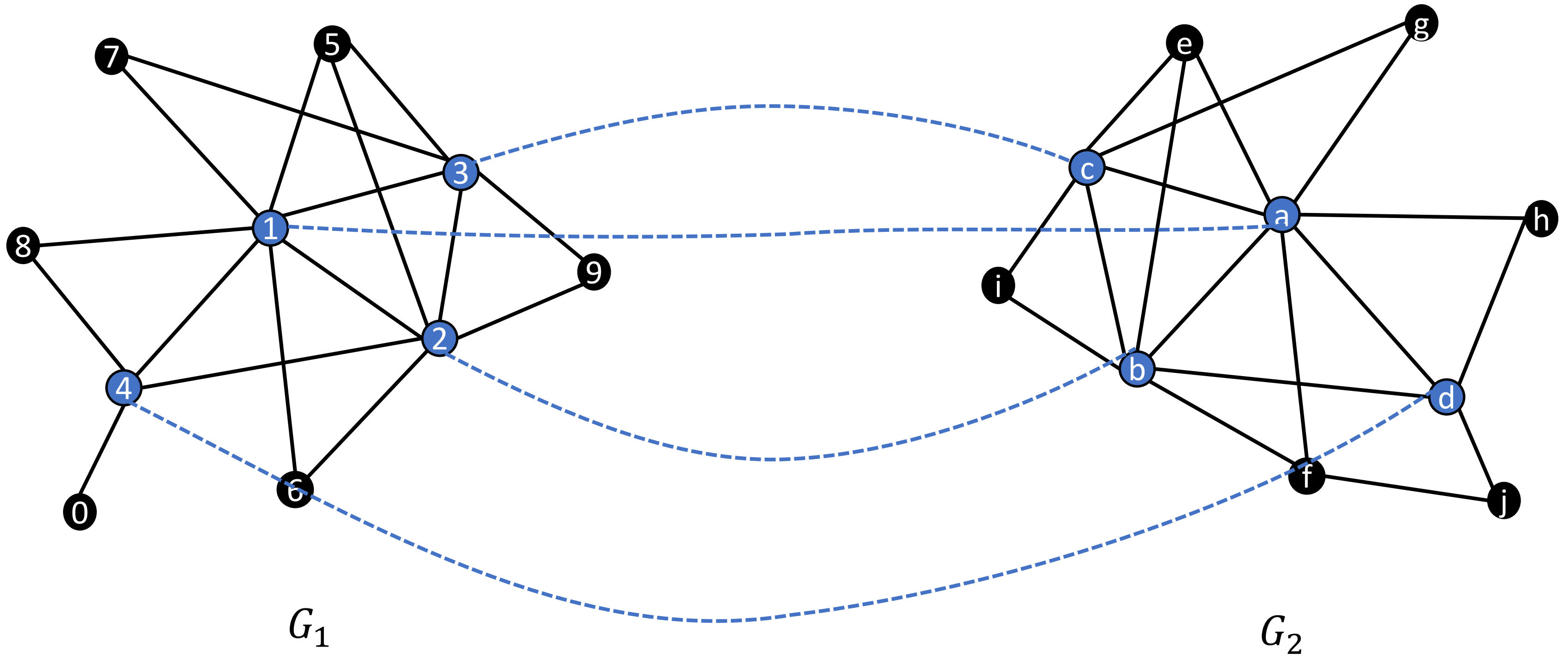


2.1 Noisy LABEL algorithm [3]

Step2: Bipartite alignment (match low degree vertices)

For $u \in V_1$,

check every $v \in G_2$ and match (u, v) if it minimize $|\mathrm{sig}_{G_1}(u) - \mathrm{sig}_{G_2}(v)|$



2.2 Black Swan Algorithm[4]

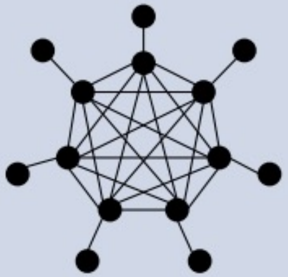
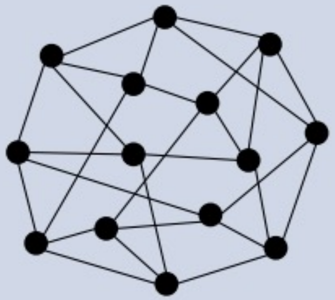
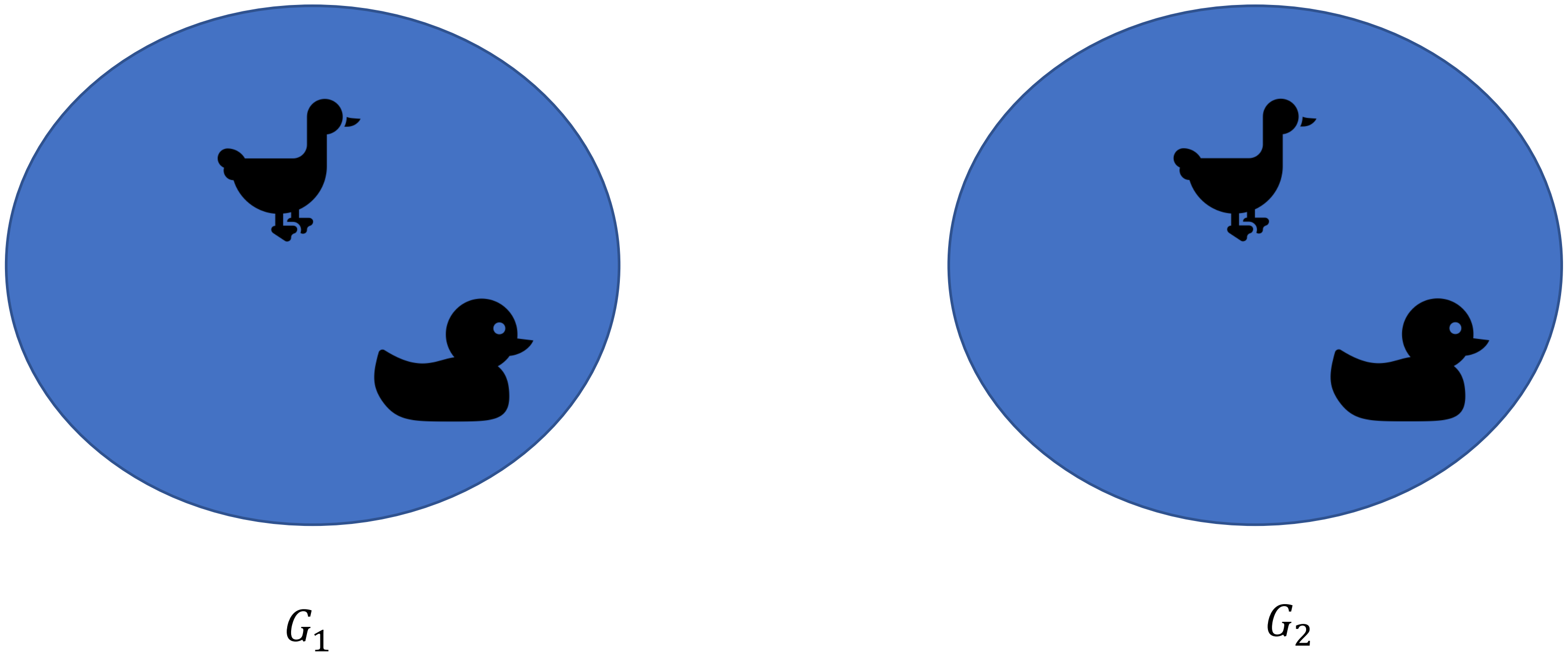
A Swan	A Black Swan
	
The variance of #appearance is **large**.	✓ #appearance **concentrates** near exp.
Too many automorphisms.	✓ **Unique** automorphism.
Large overlap with other swans.	✓ **Small** overlap with other black swans.

Figure credit to https://cnchou.github.io/docs/research/poster/poster-nips19.pdf

2.2 Black Swan Algorithm[4]

Step 1. Initialize a graph family containing large number of black swans

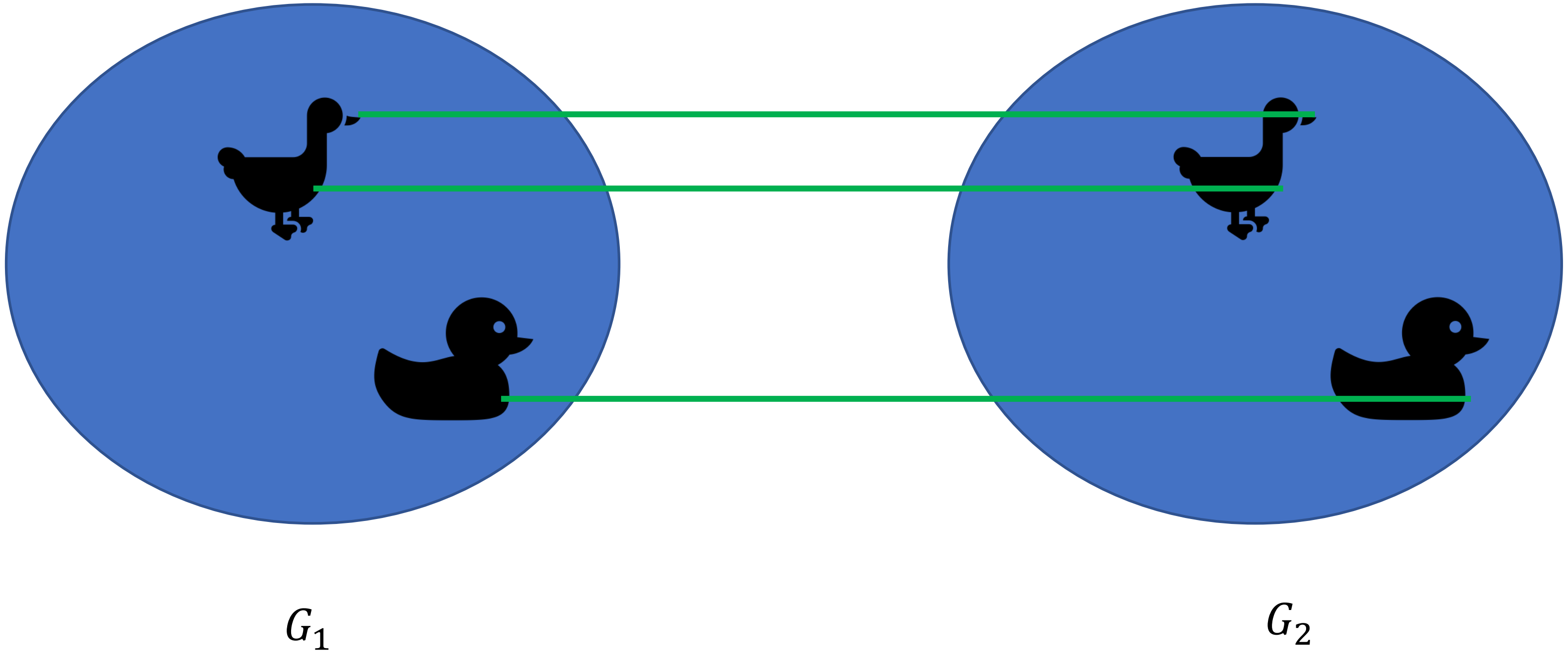


2.2 Black Swan Algorithm[4]

Step 1. Initialize a graph family containing large number of black swans

Step 2. Partial assignment

align vertices according to their location in black swans



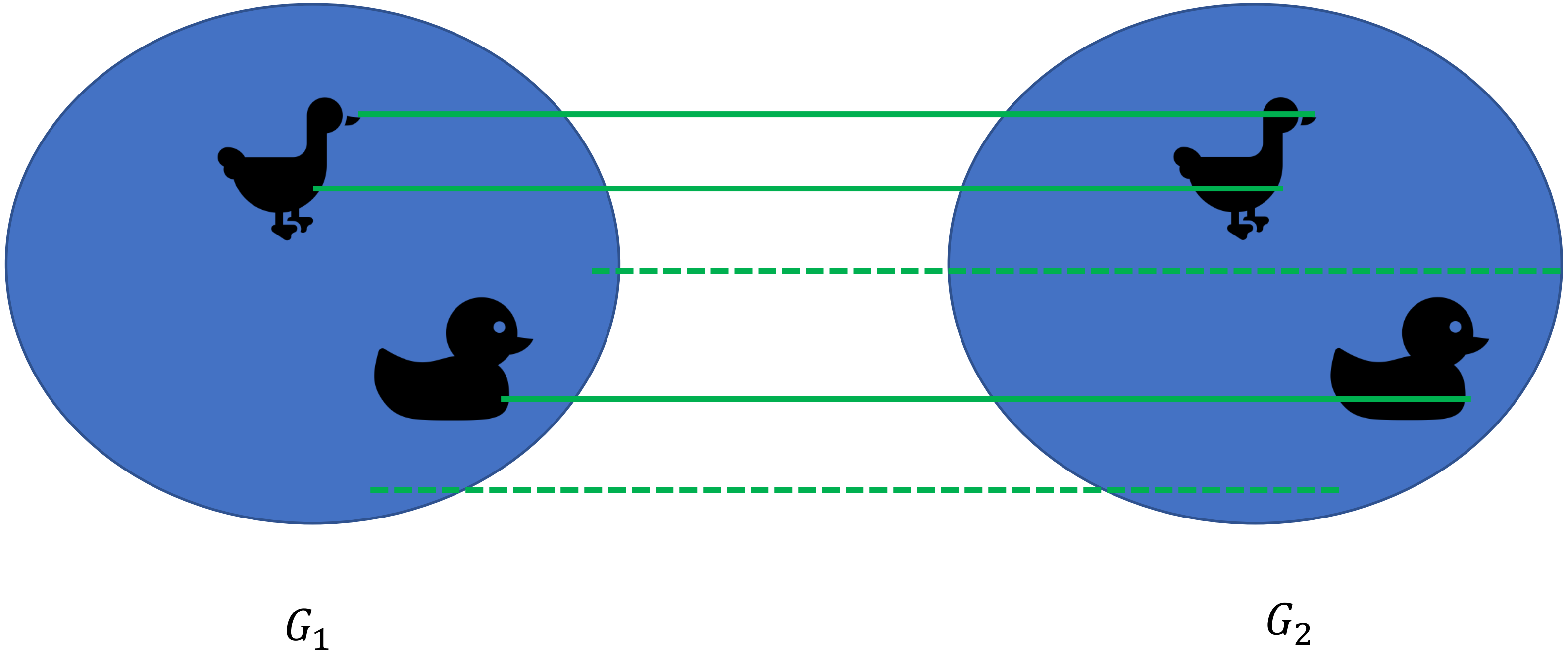
2.2 Black Swan Algorithm[4]

Step 1. Initialize a graph family containing large number of black swans

Step 2. Partial assignment

Step 3. Boosting

use the partial assignment as the seeds and generate a full permutation



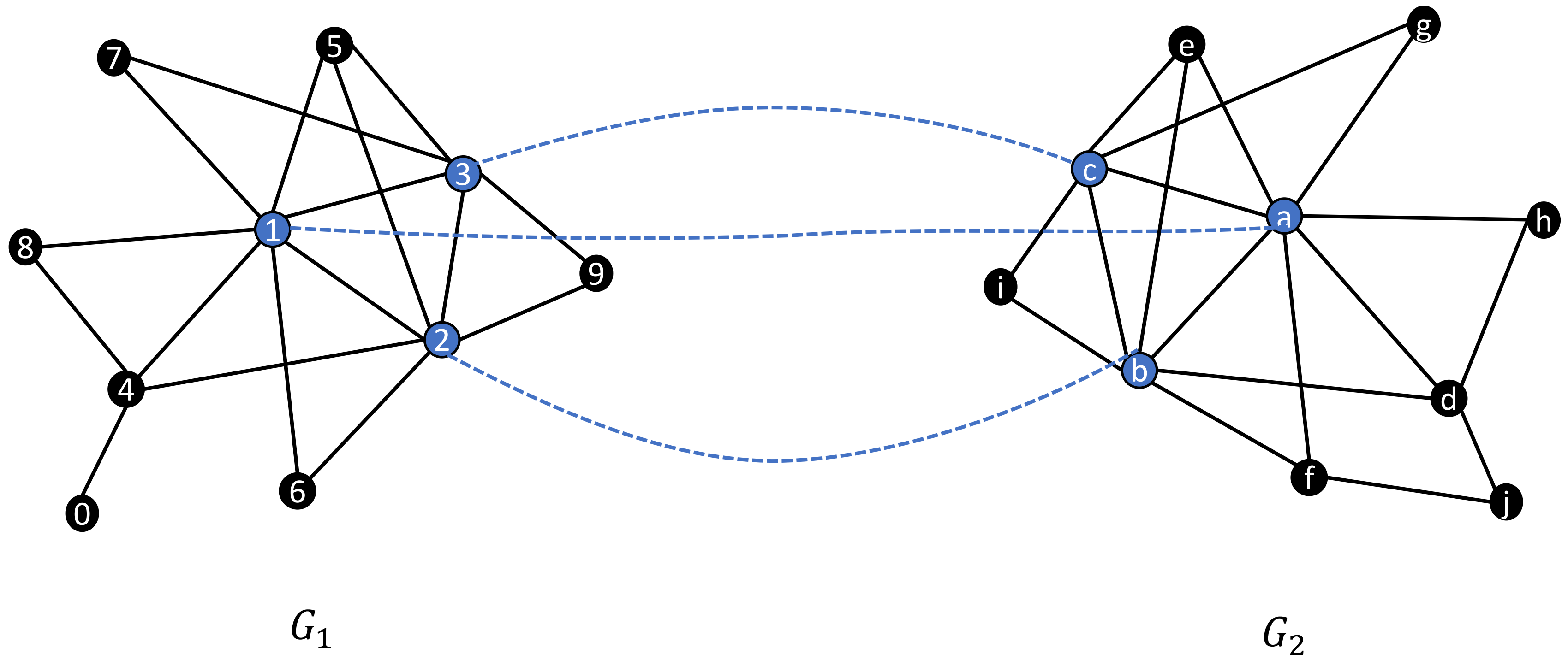
3. Seeded graph alignment

Seed set: collection of vertices $S_1 \subset V_1,\ S_2 \subset V_2$ where true alignment $S_1 \to S_2$ is known

3. Seeded graph alignment

Seed set: collection of vertices $S_1 \subset V_1,\ S_2 \subset V_2$ where true alignment $S_1 \to S_2$ is known

e.g. $S_1 = \{1,2,3\}$ and $S_2 = \{a, b, c\}$



3. Seeded graph alignment

Seed set: collection of vertices $S_1 \subset V_1,\ S_2 \subset V_2$ where true alignment $S_1 \to S_2$ is known

Goal: With extra information form seed set, find the one-to-one correspondence for the remaining vertices

3. Seeded graph alignment

Seed set: collection of vertices $S_1 \subset V_1,\ S_2 \subset V_2$ where true alignment $S_1 \to S_2$ is known

Goal: With extra information form seed set, find the one-to-one correspondence for the remaining vertices

Algorithms

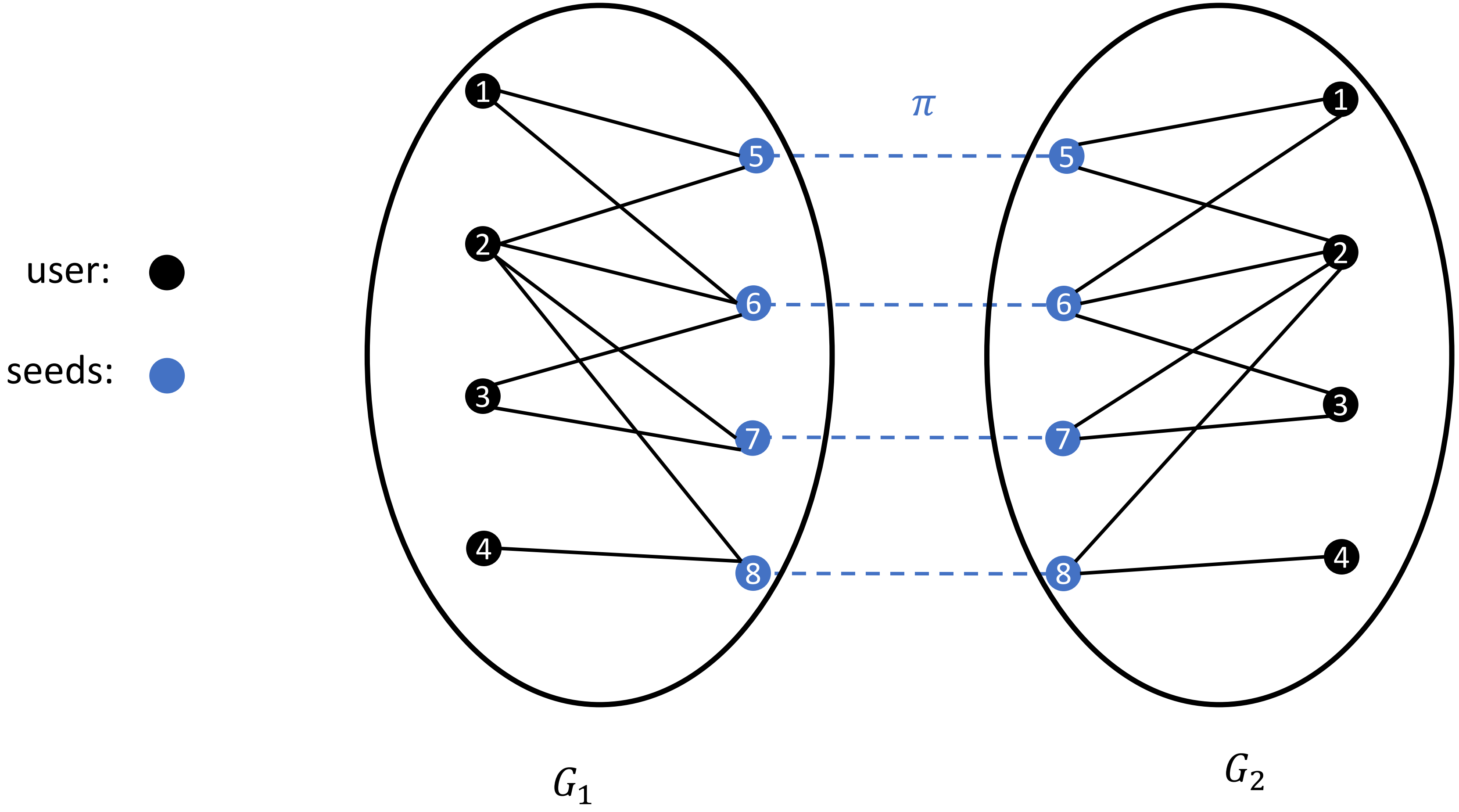
(3.1) User-matching Algorithm [5]

(3.2) Large neighborhood matching[6]

(3.3) Percolation Algorithm[7]

3.1 User-matching Algorithm[5]

Input: G_1 , G_2 and matching between the seed sets π: $S_1 \rightarrow S_2$, maximum degree D

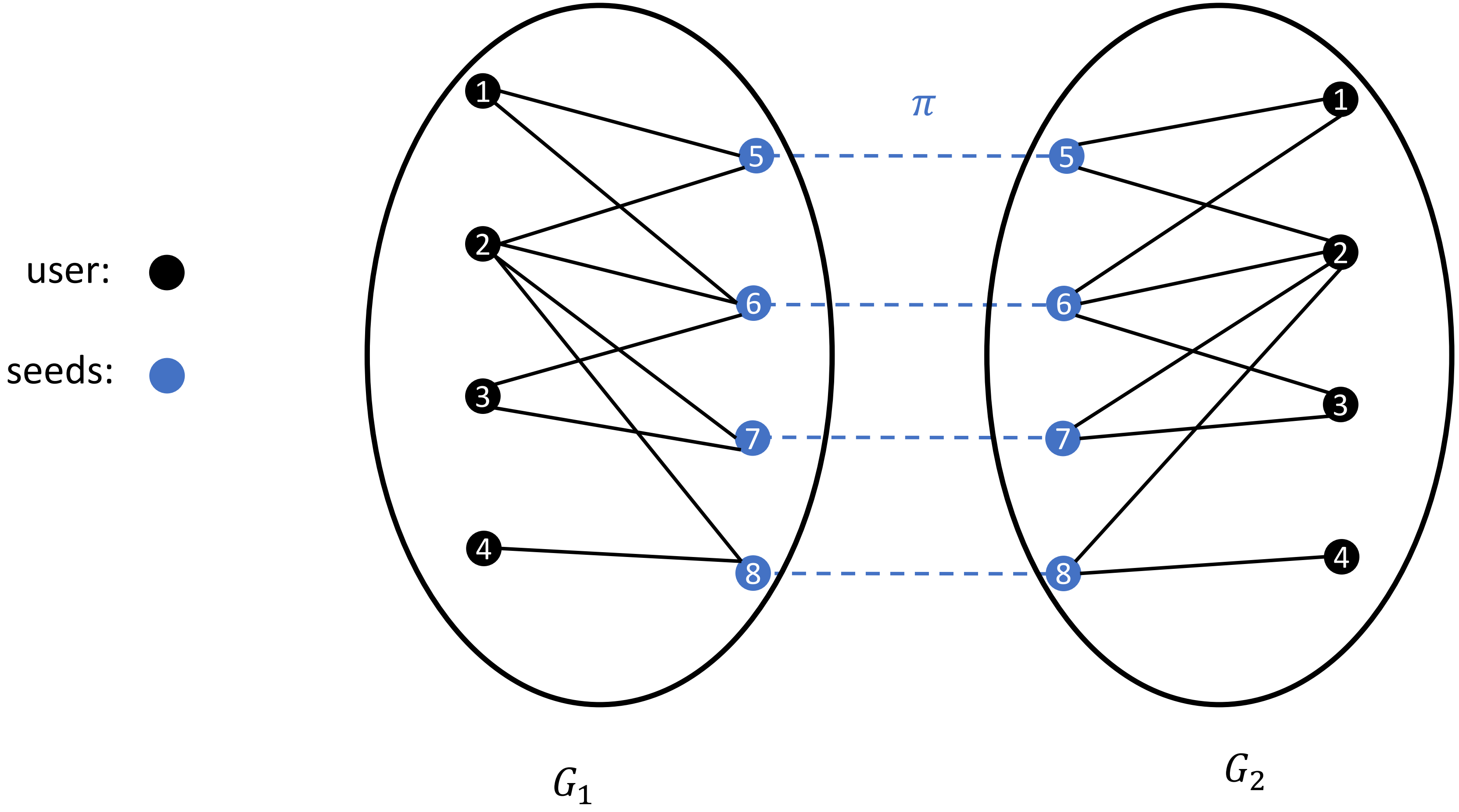


3.1 User-matching Algorithm [5]

For $j = \log D, \dots, 1$

For all user $u \in G_1, v \in G_2$ s.t. $d_{G_1}(u) \geq 2^j$ and $d_{G_2}(v) \geq 2^j$,

compute W_{uv} = # common **witnesses** between u and v and match largest score pair

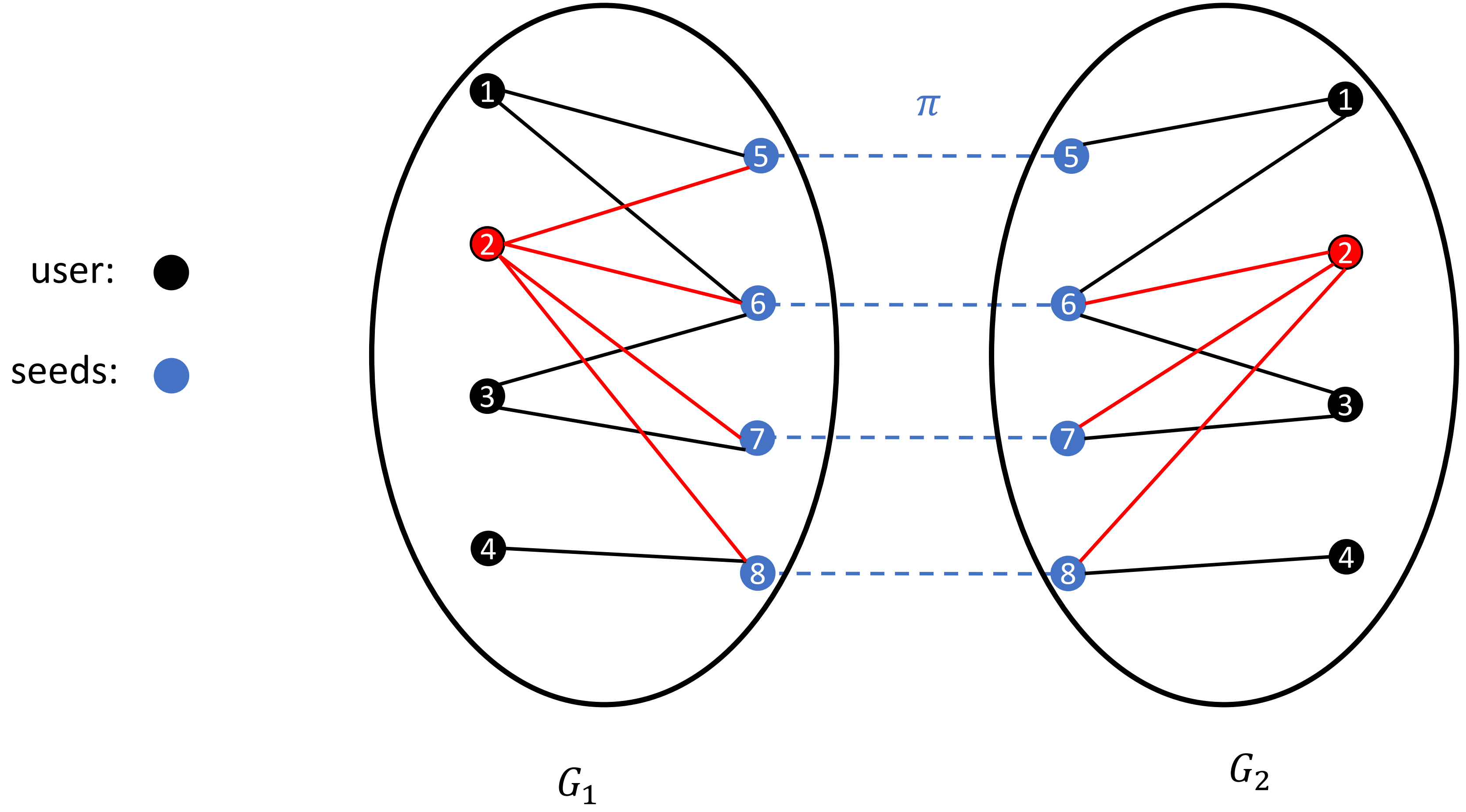


3.1 User-matching Algorithm [5]

For $j = \log D, \ldots, 1$

For all user $u \in G_1, v \in G_2$ s.t. $d_{G_1}(u) \geq 2^j$ and $d_{G_2}(v) \geq 2^j$,

compute $W_{uv} = \#$ common **witnesses** between u and v and match largest score pair



for $j = 2$

$d_{G_1}(2), d_{G_2}(2) \geq 2^2$

$N_{G_1}(2) \cap N_{G_2}(2) = \{6,7,8\}$

$W_{22} = 3$

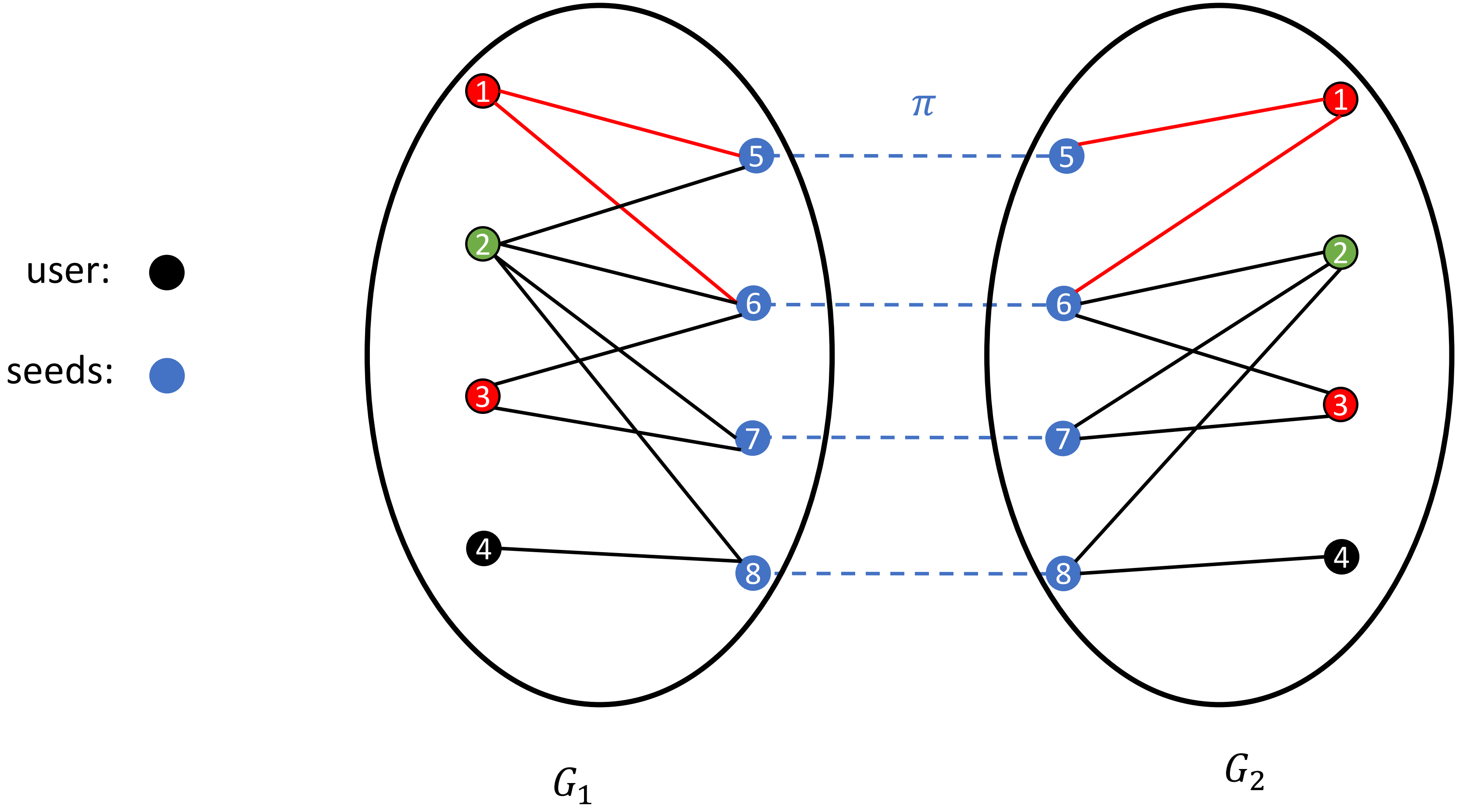
Then match 2-2

3.1 User-matching Algorithm [5]

For $j = \log D, \dots, 1$

For all user $u \in G_1$, $v \in G_2$ s.t. $d_{G_1}(u) \geq 2^j$ and $d_{G_2}(v) \geq 2^j$

Compute $W_{uv} =$ # common **witnesses** between u and v and match largest score pair



for $j = 1$

$d_{G_1}(1), d_{G_1}(3) \geq 2^1$

$d_{G_2}(1), d_{G_2}(3) \geq 2^1$

$N_{G_1}(1) \cap N_{G_2}(1) = \{5,6\}$

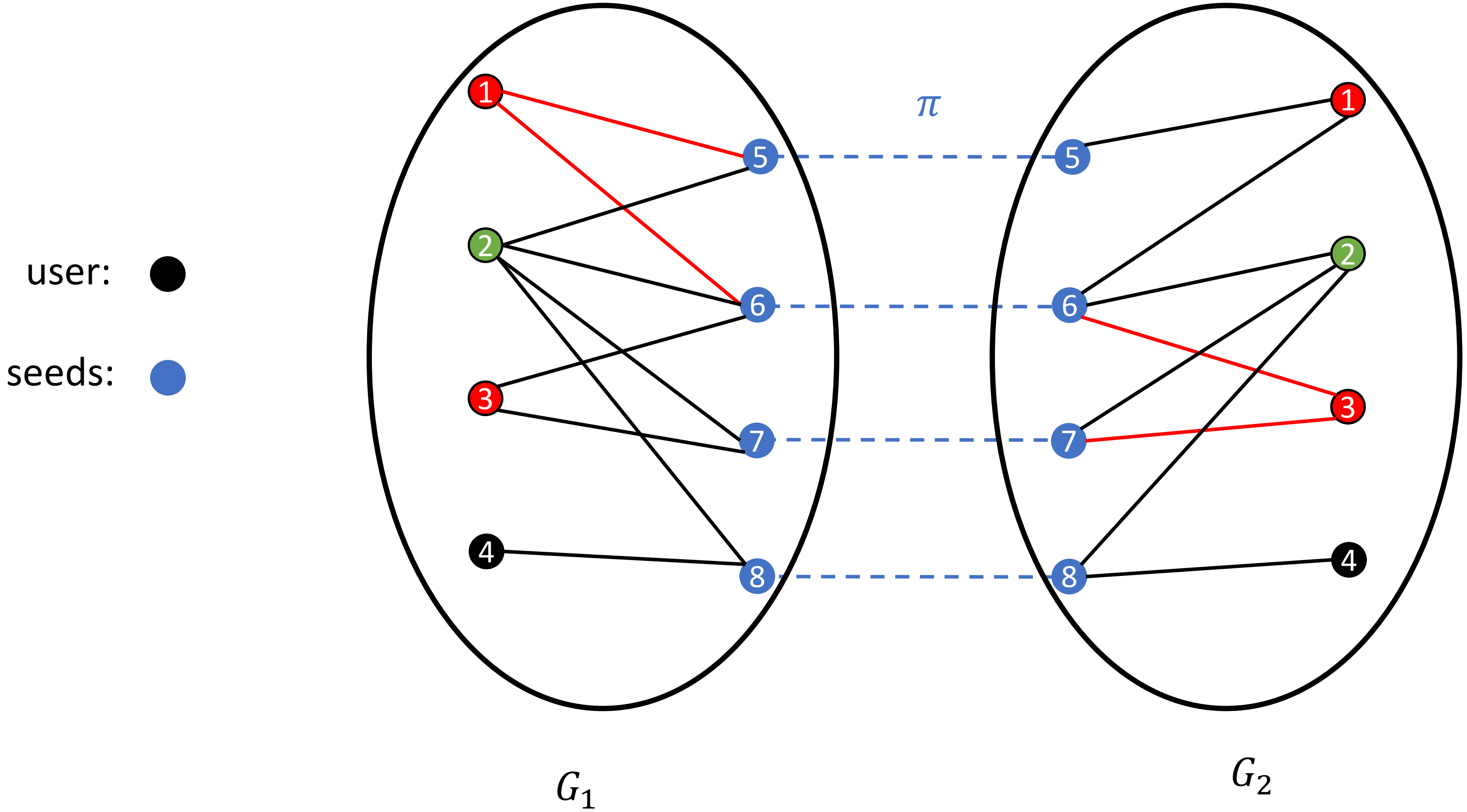
$W_{11} = 2$

3.1 User-matching Algorithm [5]

For $j = \log D, \dots, 1$

For all user $u \in G_1, v \in G_2$ s.t. $d_{G_1}(u) \geq 2^j$ and $d_{G_2}(v) \geq 2^j$

Compute $W_{uv} = \#$ common **witnesses** between u and v and match largest score pair



for $j = 1$

$d_{G_1}(1), d_{G_1}(3) \geq 2^1$

$d_{G_2}(1), d_{G_2}(3) \geq 2^1$

$N_{G_1}(1) \cap N_{G_2}(1) = \{5,6\}$

$W_{11} = 2$

$N_1(1) \cap N_2(3) = \{6\}$

$W_{13} = 1 < W_{11} = 2$

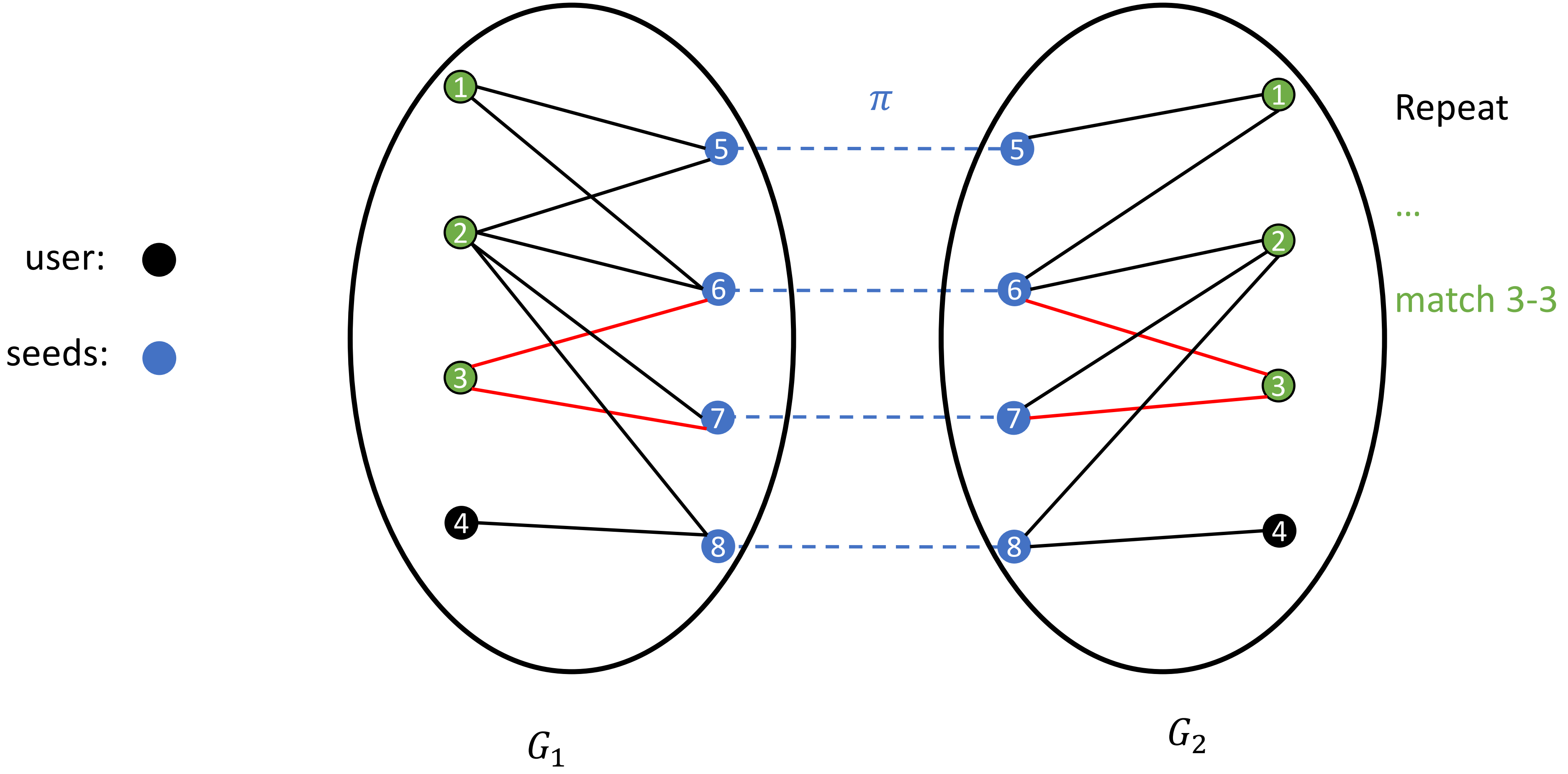
Then match 1-1

3.1 User-matching Algorithm[5]

For $j = \log D, \ldots, 1$

For all user $u \in G_1, v \in G_2$ s.t. $d_{G_1}(u) \geq 2^j$ and $d_{G_2}(v) \geq 2^j$

Compute $W_{uv} = \#$ common **witnesses** between u and v and match largest score pair

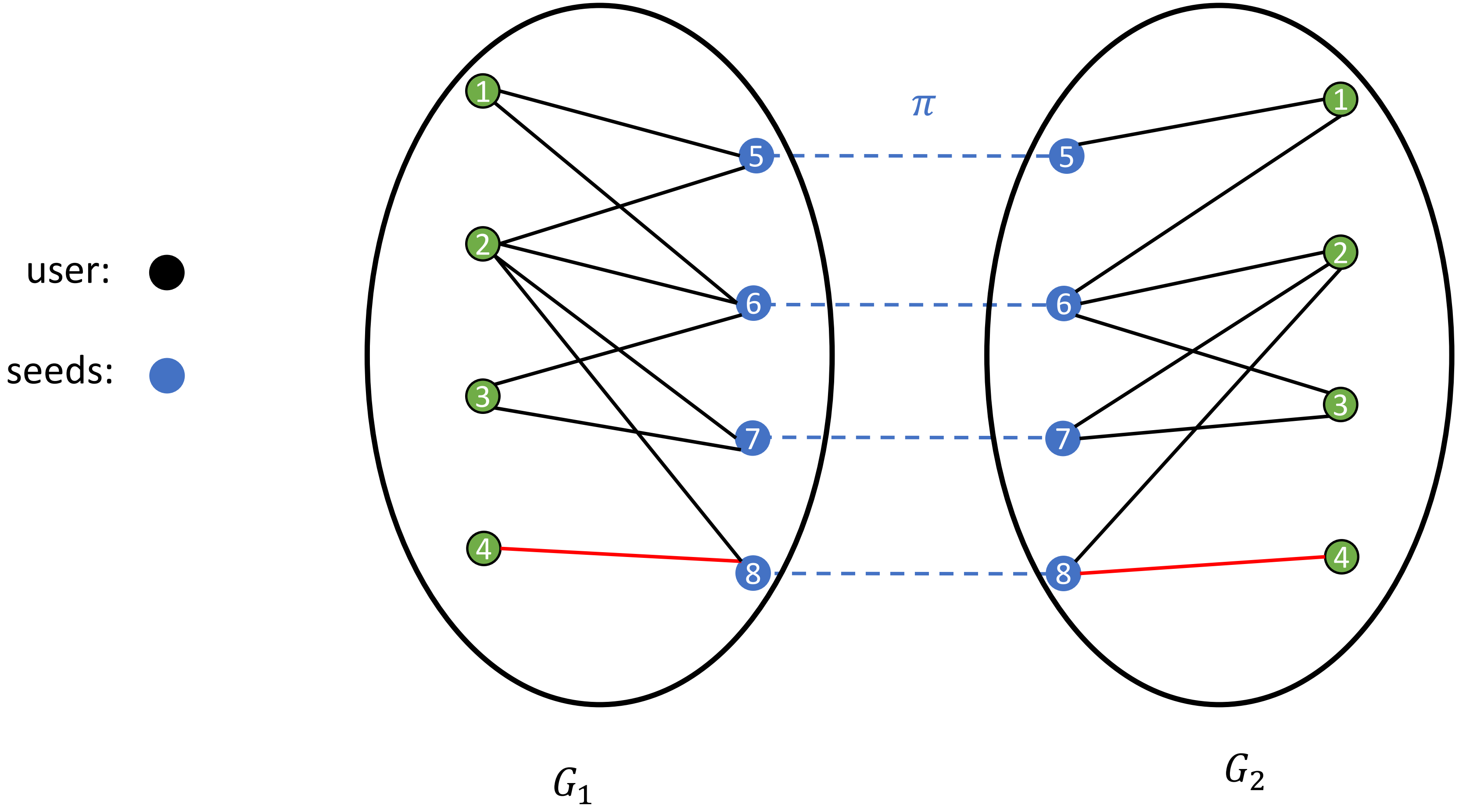


3.1 User-matching Algorithm[5]

For $j = \log D, \dots, 1$

For all user $u \in G_1, v \in G_2$ s.t. $d_{G_1}(u) \geq 2^j$ and $d_{G_2}(v) \geq 2^j$

Compute $W_{uv} = \#$ common **witnesses** between u and v and match largest score pair



Repeat

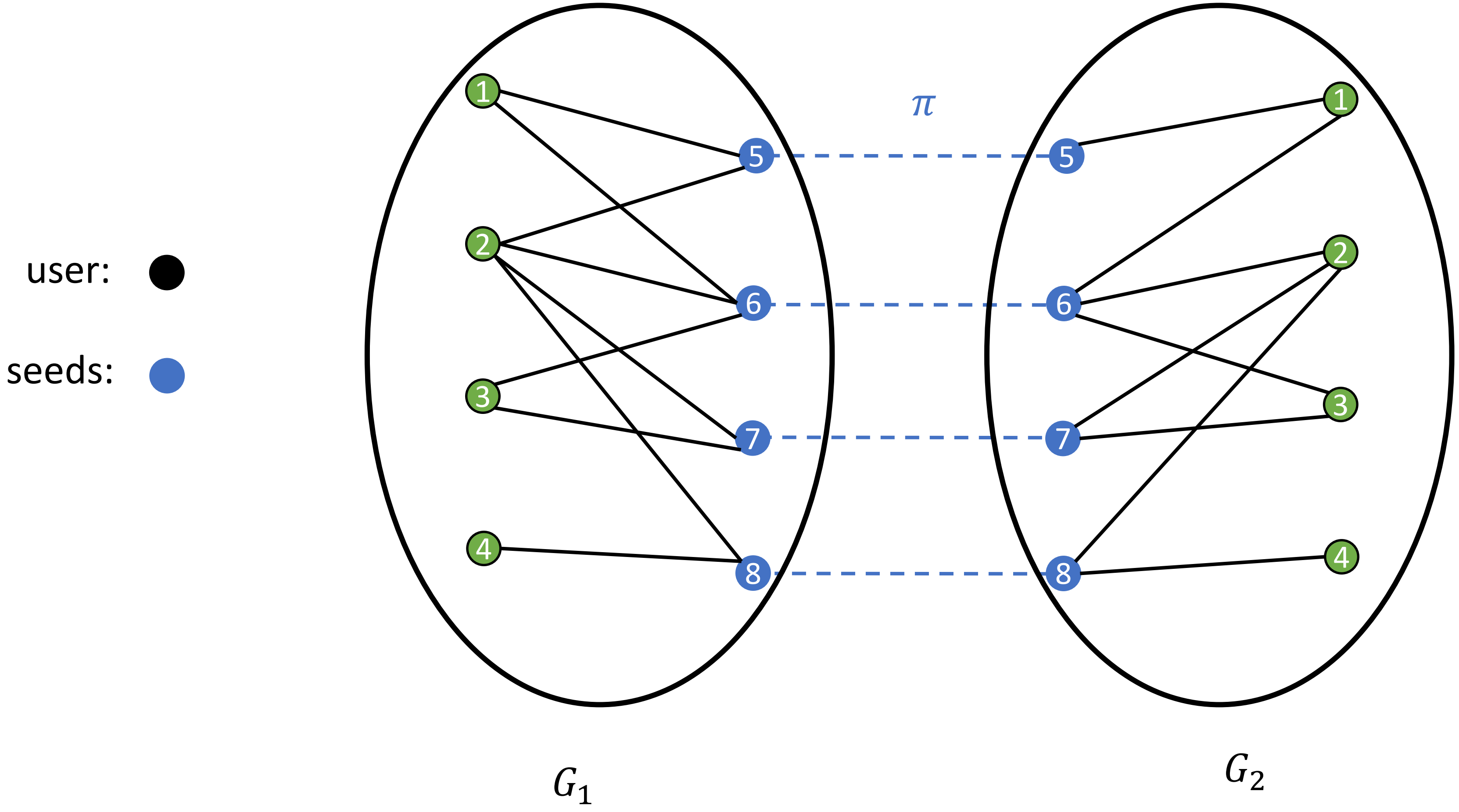
…

match 3-3

match 4-4

3.1 User-matching Algorithm[5]

Output: {1-1, 2-2, 3-3, 4-4}



3.2 Large neighborhood matching[6]

Input: G_1, G_2, π_0 and $m, l \in Z$

3.2 Large neighborhood matching[6]

Step 1. Align high degree vertices

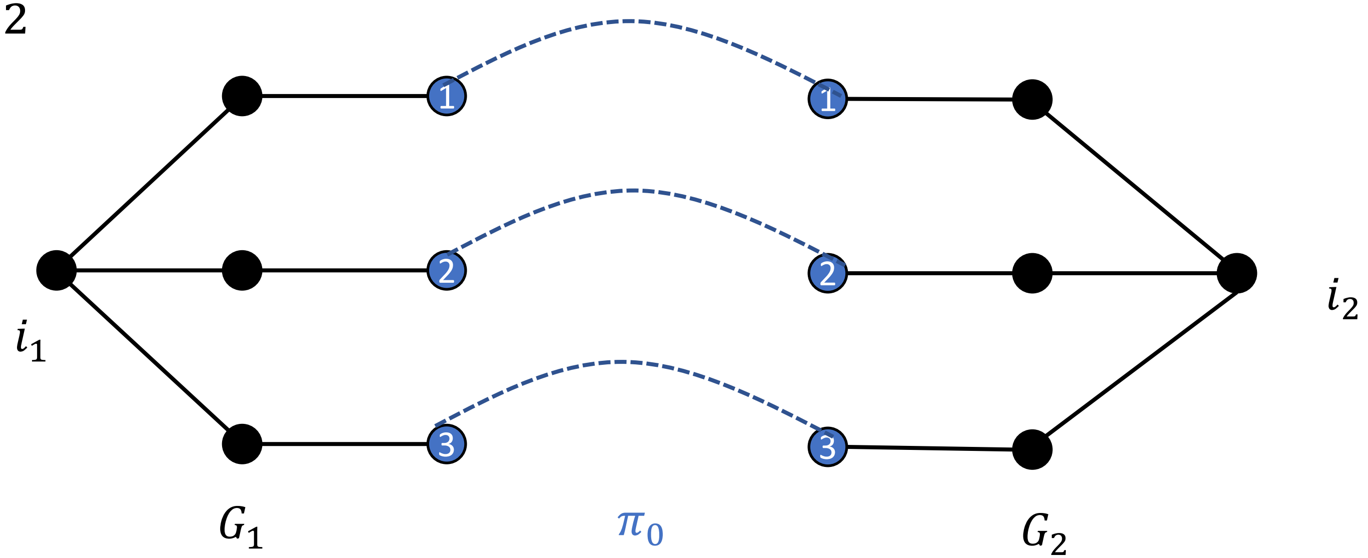
For $i_1 \in G_1$ and $i_2 \in G_2$, if

- there are m independent l –paths in G_2 from i_2 to a set of m seeded vertices $L \subset \Gamma_\ell^{G_2}(i_2)$

- there are m independent l –path in G_1 from i_1 to the same set of m seeded vertices $\pi_0(L) \subset \Gamma_\ell^{G_2}(i_1)$,

then set $\pi(i_2) = i_2$

e.g. $l = 2, m = 3$



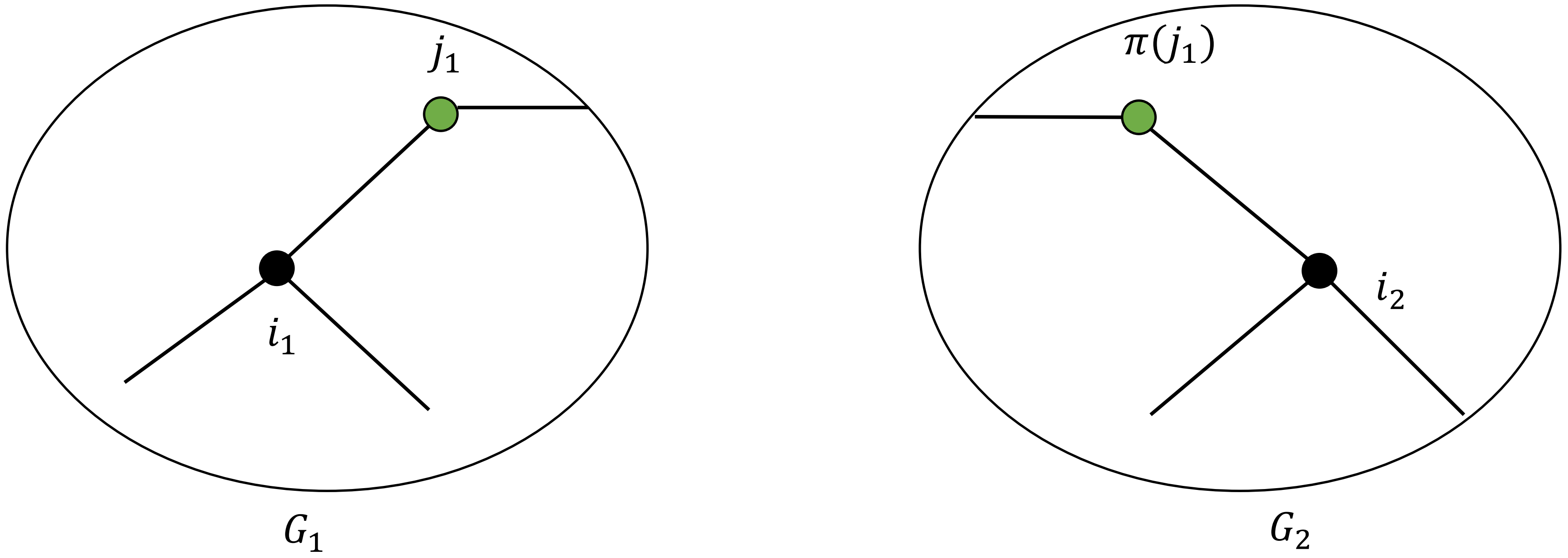
3.2 Large neighborhood matching[6]

Input: G_1, G_2, π_0 and $m, l \in Z$

Step 1. Align high degree vertices

Step 2. Align low degree vertices

For all the pairs of unmatched vertices (i_1, i_2), if i_1 is adjacent to a matched vertex j_1 in G_1 and i_2 is adjacent to vertex $\pi\,(j_1)$ in G_2, set $\pi(i_2) = i_1$.



3.2 Large neighborhood matching[6]

Input: G_1, G_2, π_0 and $m, l \in Z$

Step 1. Align high degree vertices

Step 2. Align low degree vertices

For all the pairs of unmatched vertices (i_1, i_2), if i_1 is adjacent to a matched vertex j_1 in G_1 and i_2 is adjacent to vertex $\pi\,(j_1)$ in G_2, set $\pi(i_2) \ = \ i_1$.

Output: π

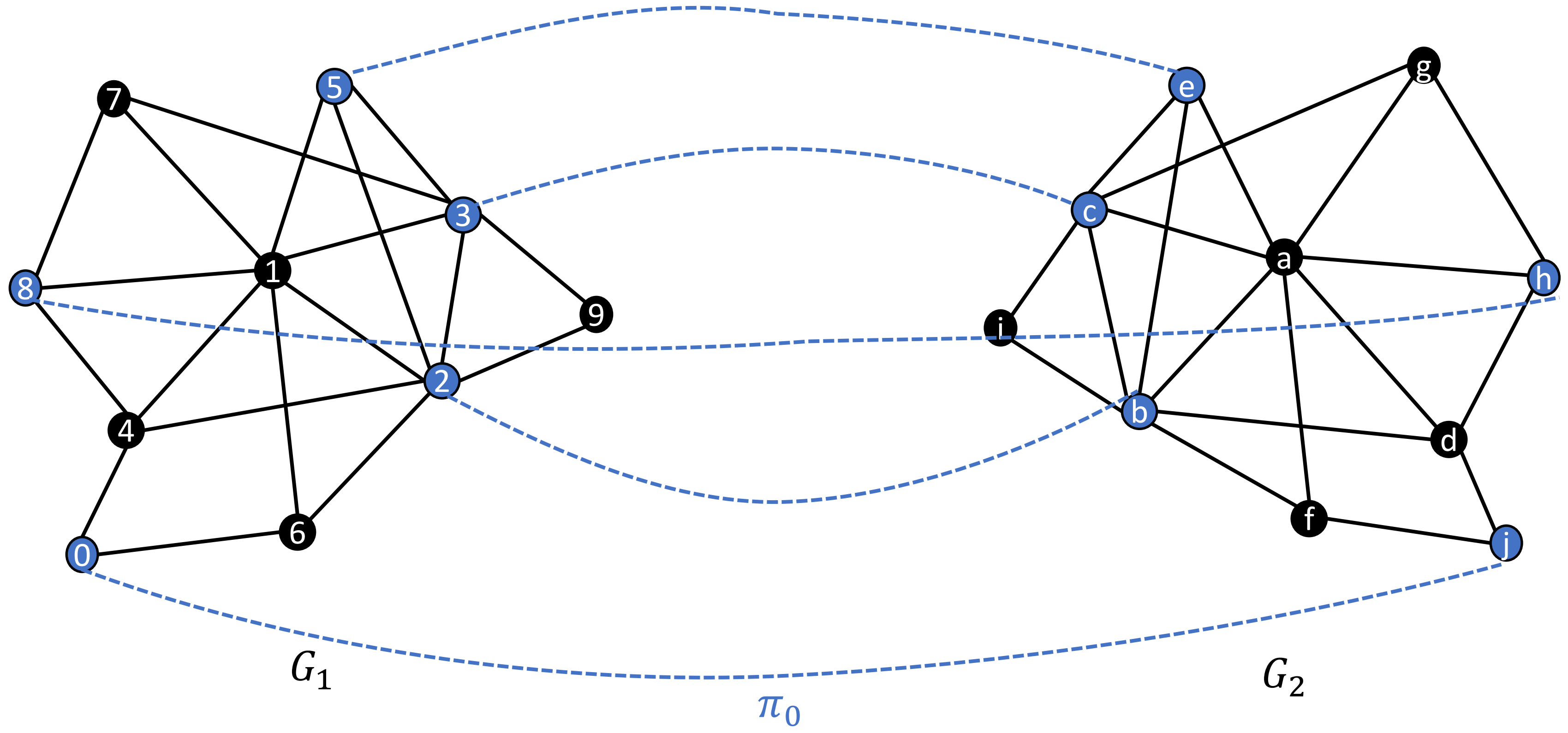
3.3 Percolation Algorithm[7]

Key idea: in each iteration, we map any two nodes with at least r neighboring pairs already mapped.

3.3 Percolation Algorithm[7]

Input: $G_1 = (V_1, E_1), G_2 = (V_2, E_2), \pi_0, r$

e.g. $r = 2$



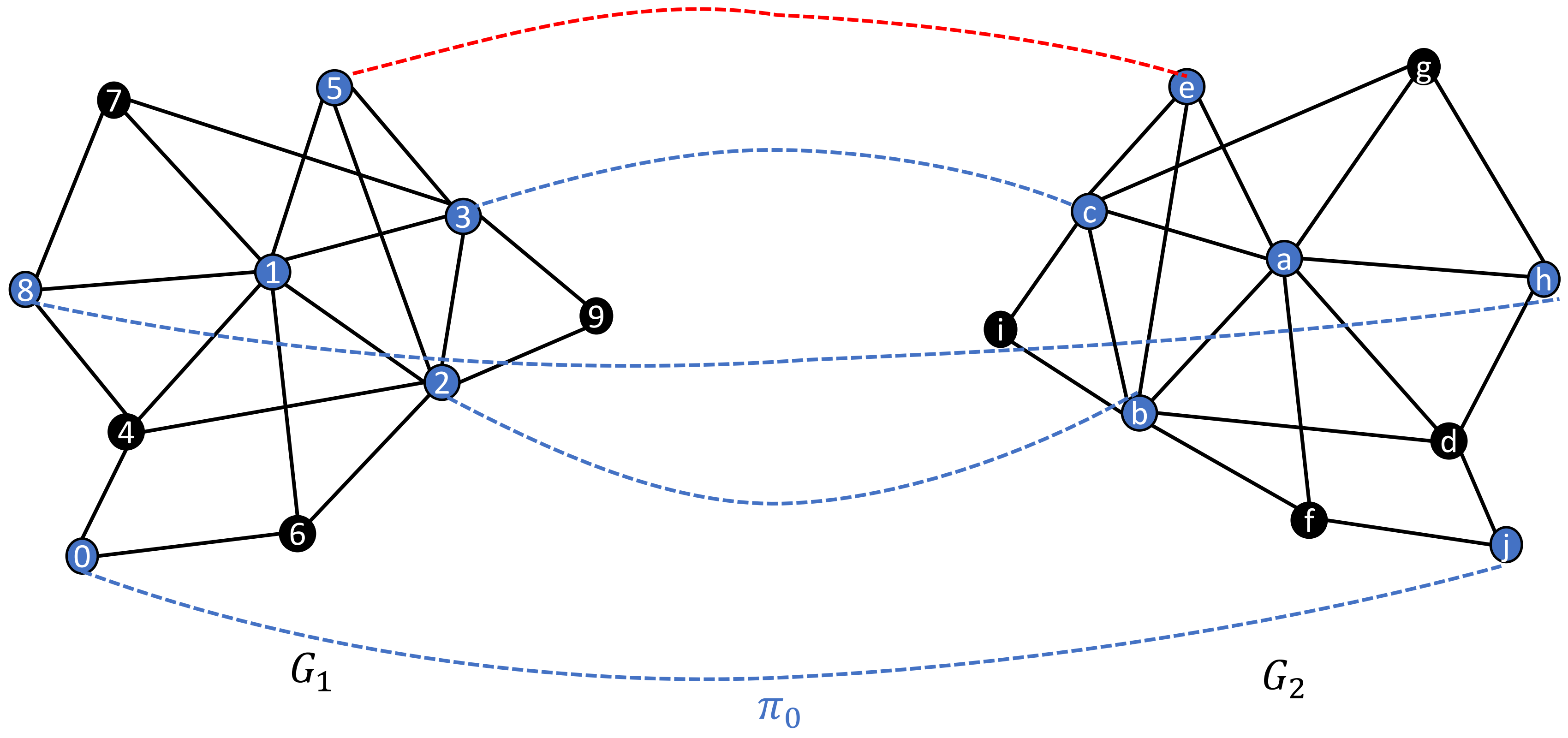
3.3 Percolation Algorithm[7]

Associate with every pair of nodes $(i \in V_1, j \in V_2)$ a count of marks $M_{i,j}$ in the following way:

at each time step t, uses exactly one unused but already mapped pair (i_t, j_t)

$t = 1$

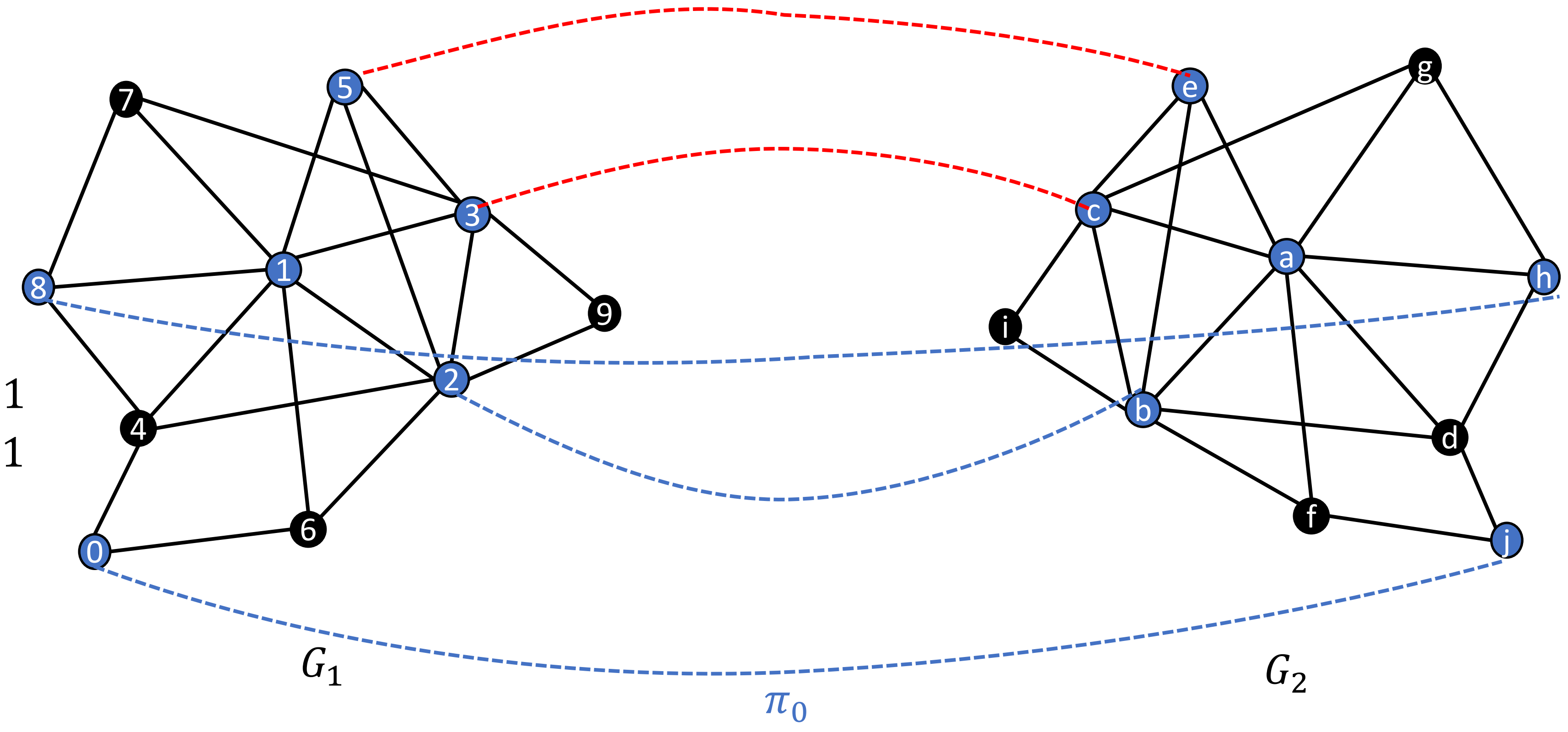
$M_{1,a} = 1$



3.3 Percolation Algorithm[7]

Associate with every pair of nodes $(i \in V_1, j \in V_2)$ a count of marks $M_{i,j}$ in the following way:

at each time step t, uses exactly one unused but already mapped pair (i_t, j_t)



$t = 2$

$M_{1,a} = 2$

$M_{7,g}, M_{7,i} = 1$

$M_{9,g}, M_{9,i} = 1$

3.3 Percolation Algorithm[7]

Associate with every pair of nodes $(i \in V_1, j \in V_2)$ a count of marks $M_{i,j}$ in the following way:

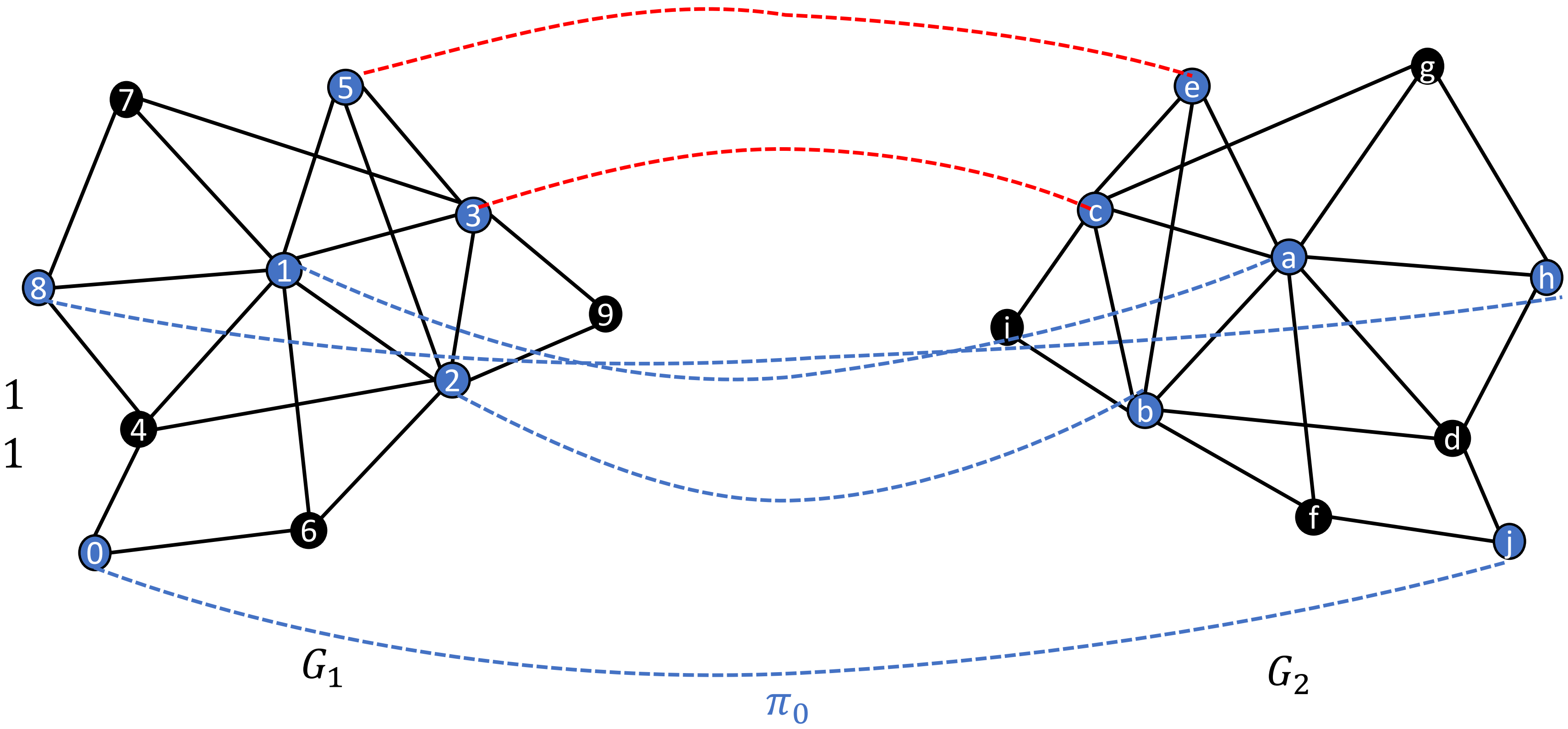
at each time step t, uses exactly one unused but already mapped pair (i_t, j_t)

$t = 2$

$M_{1,a} = 2$

$M_{7,g}, M_{7,i} = 1$

$M_{9,g}, M_{9,i} = 1$



3.3 Percolation Algorithm[7]

Associate with every pair of nodes $(i \in V_1, j \in V_2)$ a count of marks $M_{i,j}$ in the following way:

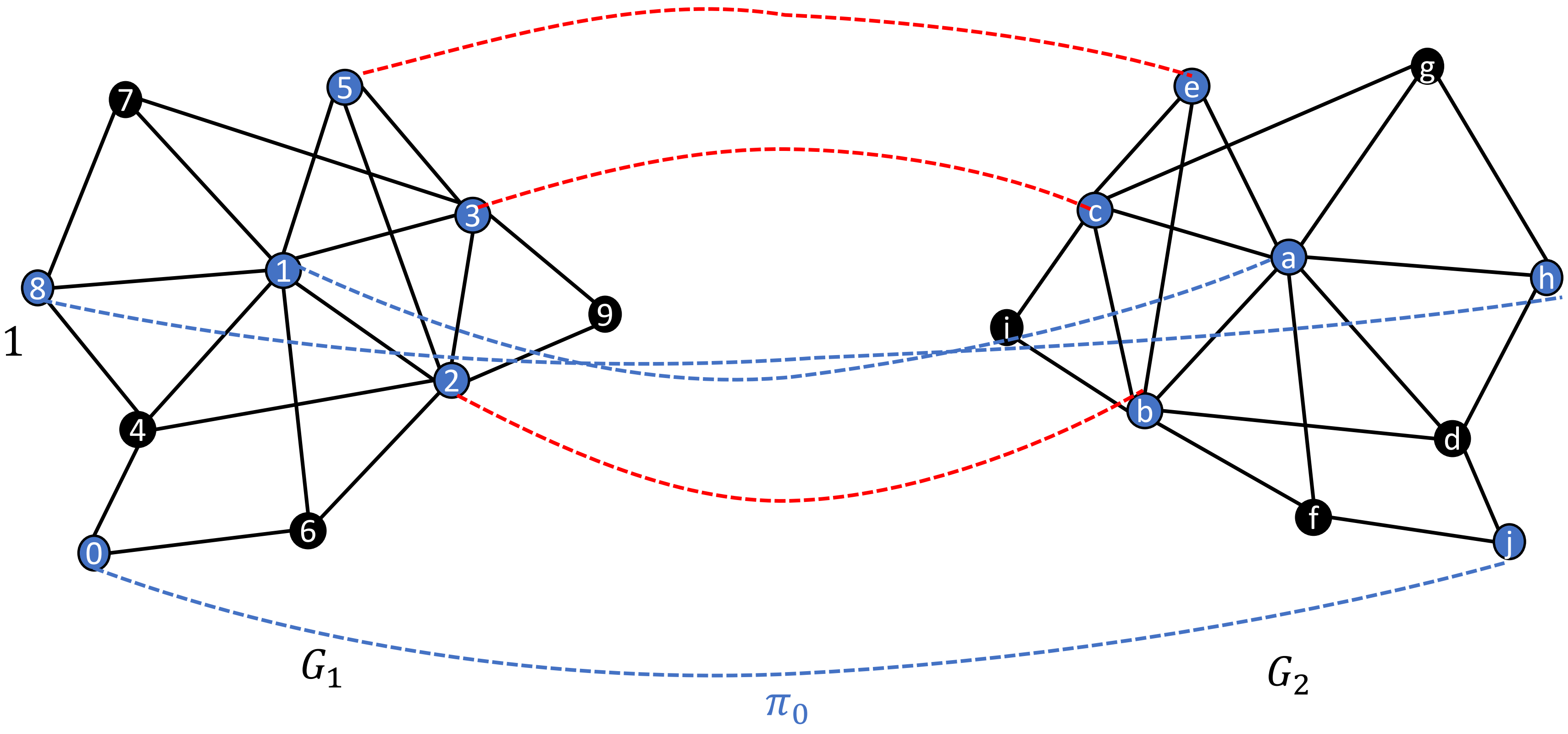
at each time step t, uses exactly one unused but already mapped pair (i_t, j_t)

$t = 3$

$M_{7,g}, M_{7,i} = 1$

$M_{9,g} = 1$

$M_{9,i} = 2$

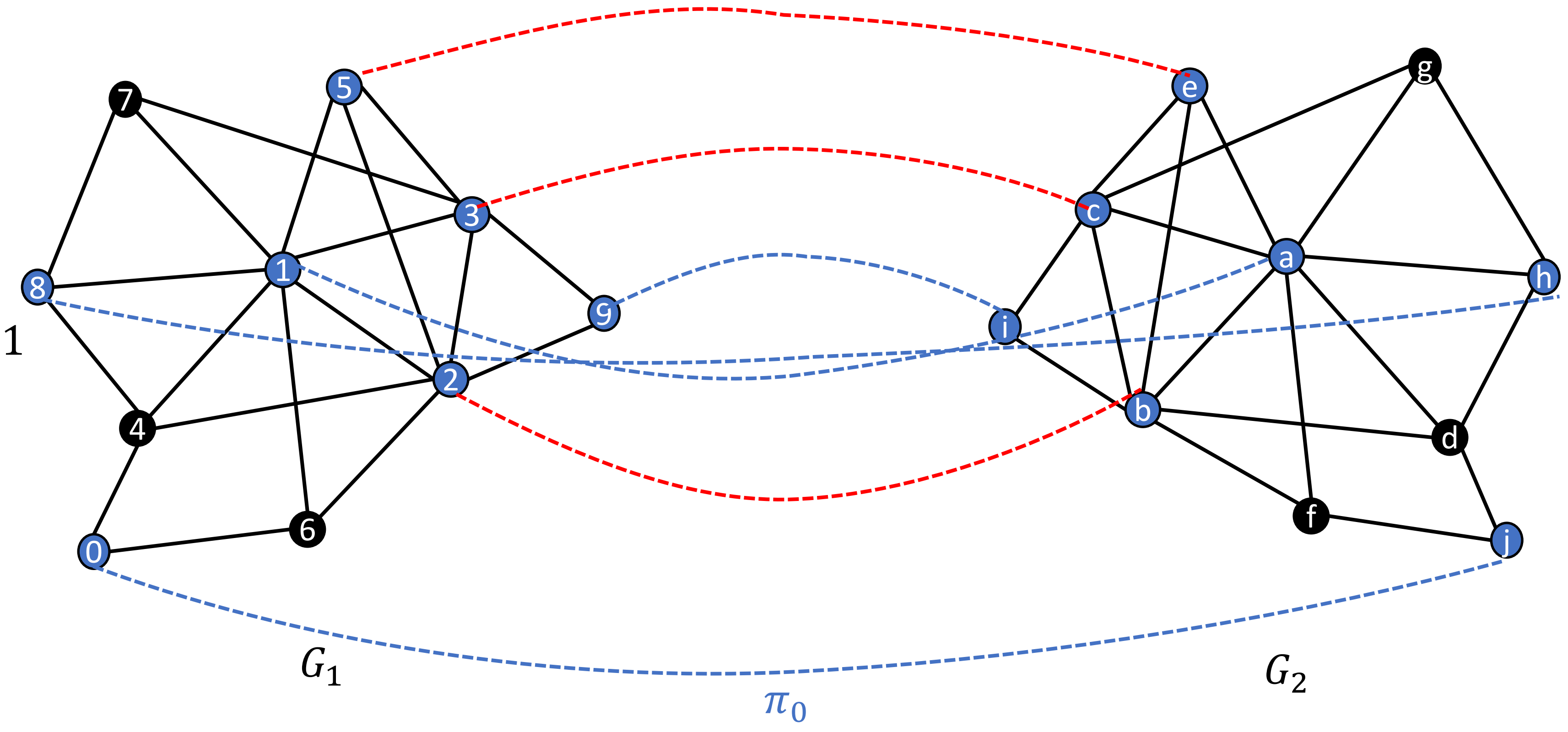


3.3 Percolation Algorithm[7]

Associate with every pair of nodes $(i \in V_1, j \in V_2)$ a count of marks $M_{i,j}$ in the following way:

at each time step t, uses exactly one unused but already mapped pair (i_t, j_t)

$t = 3$
$M_{7,g}, M_{7,i} = 1$
$M_{9,g} = 1$
$M_{9,i} = 2$



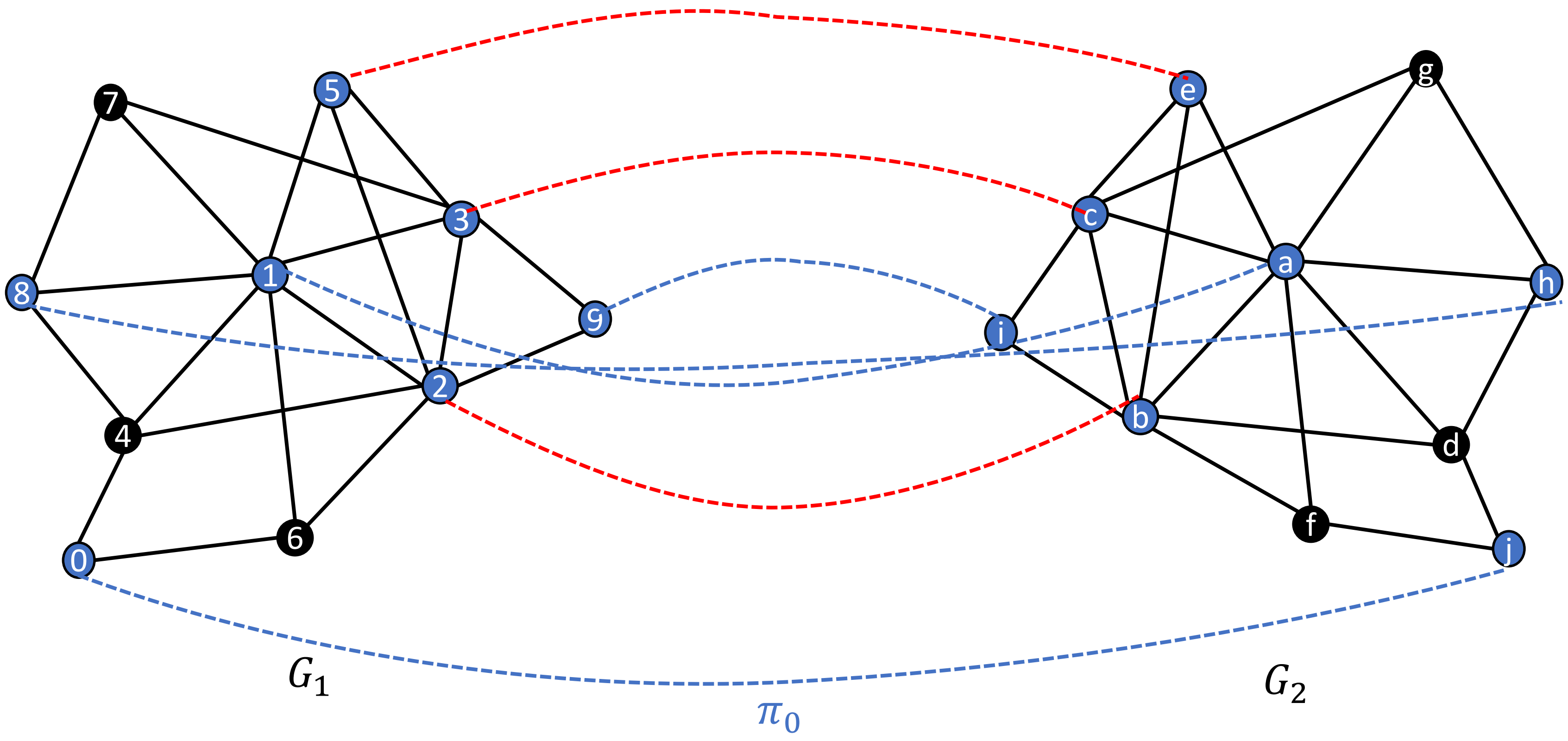
3.3 Percolation Algorithm[7]

Associate with every pair of nodes $(i \in V_1, j \in V_2)$ a count of marks $M_{i,j}$ in the following way:

at each time step t, uses exactly one unused but already mapped pair (i_t, j_t)

…

repeat until there are no unused pairs.



References

[1] Babai, László, Paul Erdos, and Stanley M. Selkow. "Random graph isomorphism." *SIaM Journal on computing* 9.3 (1980): 628-635.

[2] Czajka, Tomek, and Gopal Pandurangan. "Improved random graph isomorphism." *Journal of Discrete Algorithms* 6.1 (2008): 85-92.

[3] Dai, Osman Emre, et al. "Analysis of a canonical labeling algorithm for the alignment of correlated erdos-rényi graphs." *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3.2 (2019): 1-25.

[4] Barak, Boaz, et al. "(Nearly) efficient algorithms for the graph matching problem on correlated random graphs." *Advances in Neural Information Processing Systems* 32 (2019): 9190-9198.

[5]Korula, Nitish, and Silvio Lattanzi. "An efficient reconciliation algorithm for social networks." *arXiv preprint arXiv:1307.1690* (2013).

[6] Mossel, Elchanan, and Jiaming Xu. "Seeded graph matching via large neighborhood statistics." *Random Structures & Algorithms* 57.3 (2020): 570-611.

[7] Yartseva, Lyudmila, and Matthias Grossglauser. "On the performance of percolation graph matching." *Proceedings of the first ACM conference on Online social networks*. 2013.