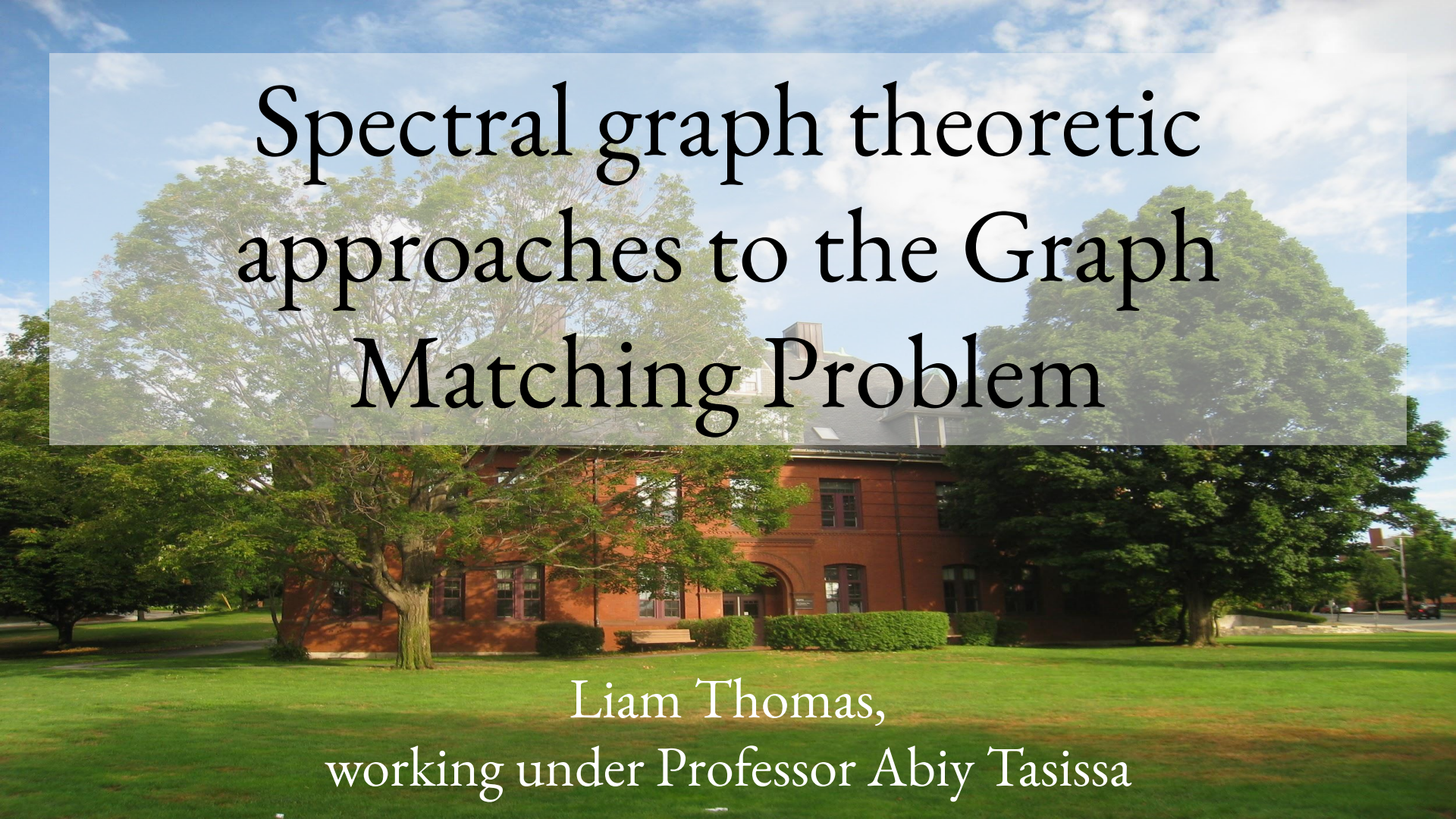
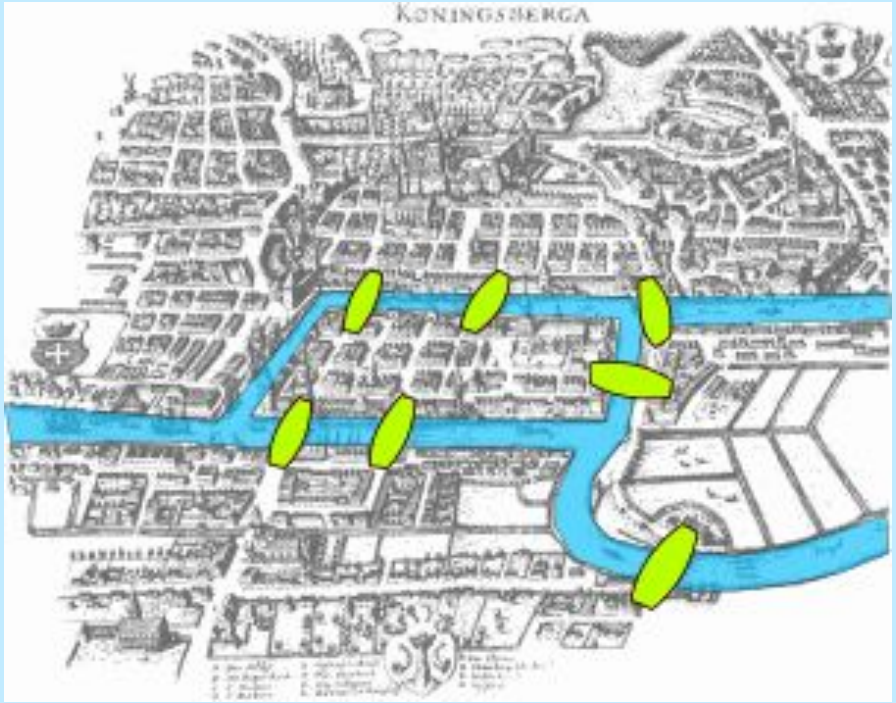
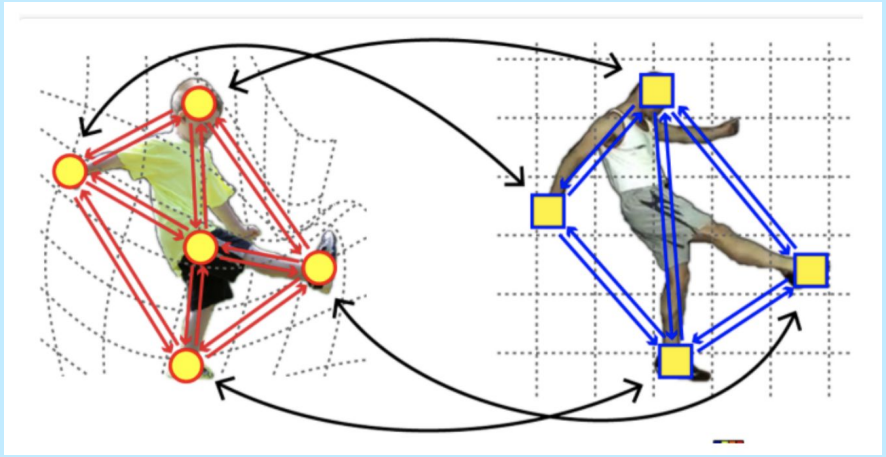


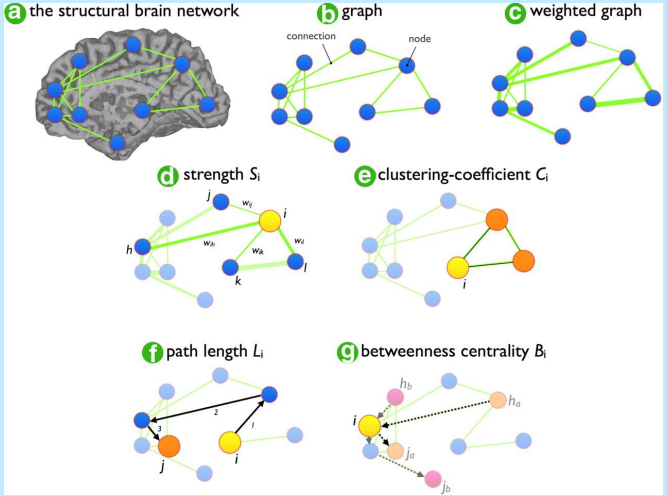
Spectral graph theoretic approaches to the Graph Matching Problem

Liam Thomas,
working under Professor Abiy Tasissa

Part I: Graphs and their associated matrices



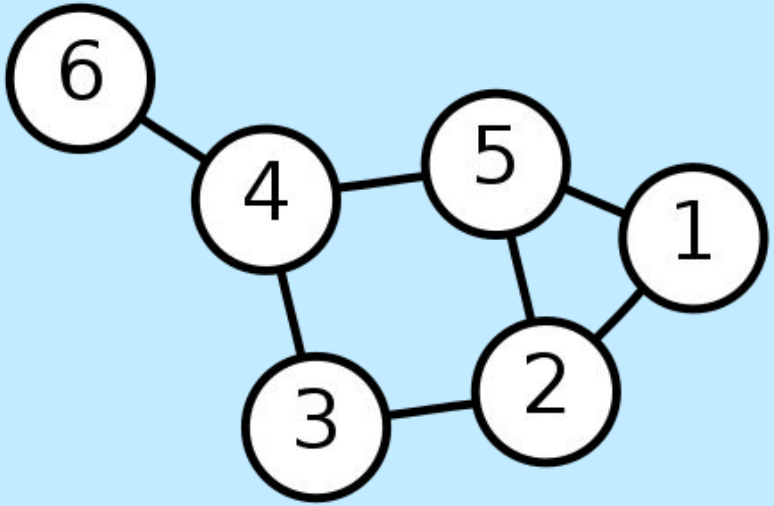




The adjacency matrix

A(i,j) = 1 if node *i* is connected to node *j*, and **0** otherwise.

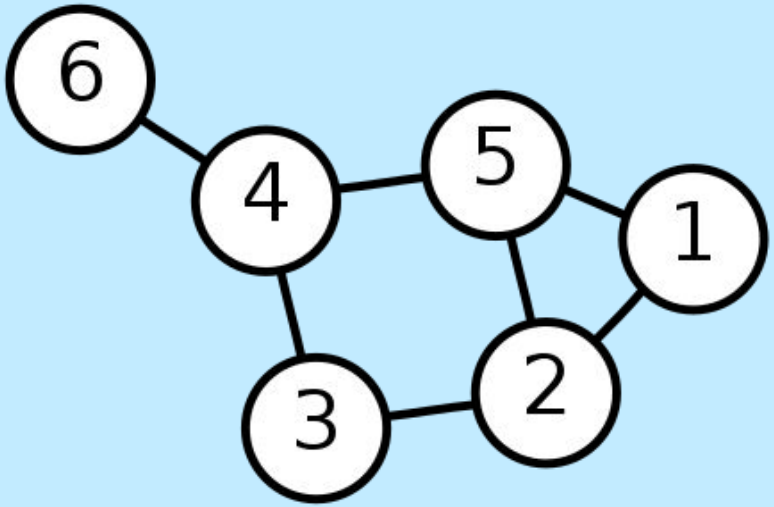
In row 1, node 1 is connected to 5 and 2, so the fifth and sixth entries of the first row are equal to 1.



0	1	0	0	1	0
1	0	1	0	1	0
0	1	0	1	0	0
0	0	1	0	1	1
1	1	0	1	0	0
0	0	0	1	0	0

The degree matrix

The **degree matrix** of the graph works as follows: node 1 is connected to 2 other nodes, so the **first** entry of the degree matrix will be 2, and so forth.



2	0	0	0	0	0
0	3	0	0	0	0
0	0	2	0	0	0
0	0	0	3	0	0
0	0	0	0	3	0
0	0	0	0	0	1

The Laplacian matrix

The last matrix we'll introduce is the **Laplacian.** The Laplacian is simply the **degree matrix minus the adjacency matrix.** So if we take the degree and adjacency matrices from the previous example, we can determine the Laplacian matrix:

Degree matrix

2	0	0	0	0	0
0	3	0	0	0	0
0	0	2	0	0	0
0	0	0	3	0	0
0	0	0	0	3	0
0	0	0	0	0	1

−

Adjacency matrix

0	1	0	0	1	0
1	0	1	0	1	0
0	1	0	1	0	0
0	0	1	0	1	1
1	1	0	1	0	0
0	0	0	1	0	0

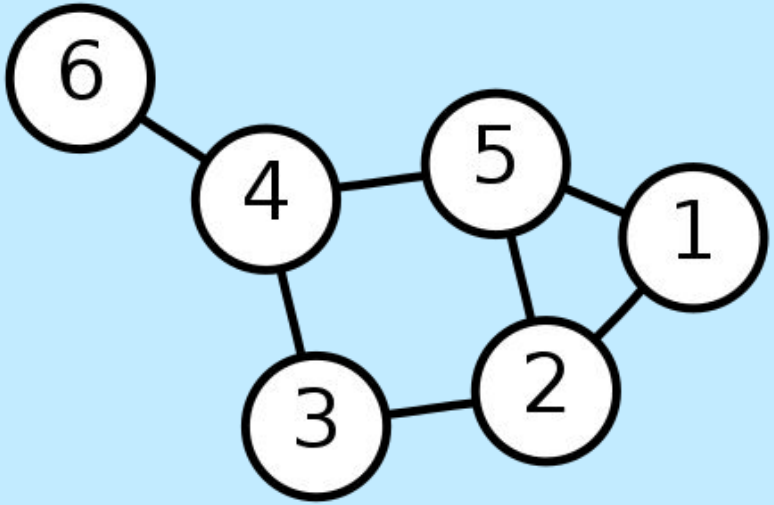
=

Laplacian matrix

2	-1	0	0	-1	0
-1	3	-1	0	-1	0
0	-1	2	-1	0	0
0	0	-1	3	-1	-1
-1	-1	0	-1	3	0
0	0	0	-1	0	1

Part II: Measuring distance on graphs

Before we return to Laplacian matrices, we need to develop some intuition for defining **distances** on graphs. If we use ***shortest-path distance,*** **the distance between 1 and 6 would be 3 edges.**

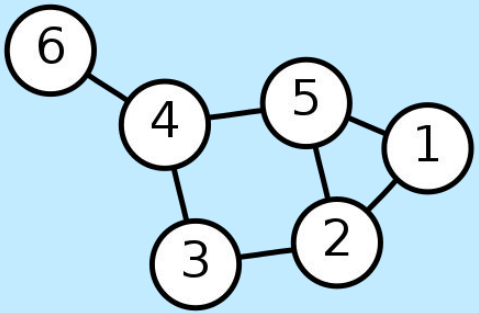


However, random walks on the graph allow us to construct a more general notion of distance on graphs.

The transition matrix

Transition matrices allow us to “hop” between any two nodes of the graph. No matter where we are hopping on the graph, the probabilities of the transition matrix never change. Here is the transition matrix for the graph we’ve been looking at:

The first row says that we will hop from node 1 to either node 2 or node 5 with equal probability. The second row says that, from node 2, we will hop to node 1, 3, or 5 with equal probability.



0	1/2	0	0	1/2	0
1/3	0	1/3	0	1/3	0
0	1/2	0	1/2	0	0
0	0	1/3	0	1/3	1/3
1/3	1/3	0	1/3	0	0
0	0	0	1	0	0

Diffusion distance

Let $p(t, y|x)$ denote the probability of reaching node *y* from node *x* in a random walk of length *t*.

Then the ***diffusion distance*** between nodes x_0 and x_1 on a graph is given by the formula:

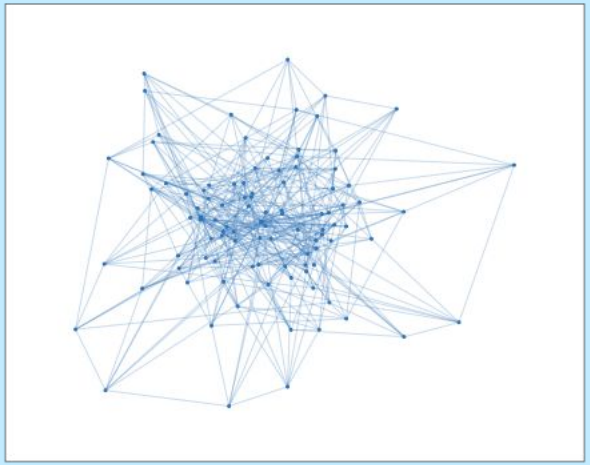
$$D_t^2(x_0, x_1) := ||p(t, y|x_o) - p(t, y|x_1)||^2$$

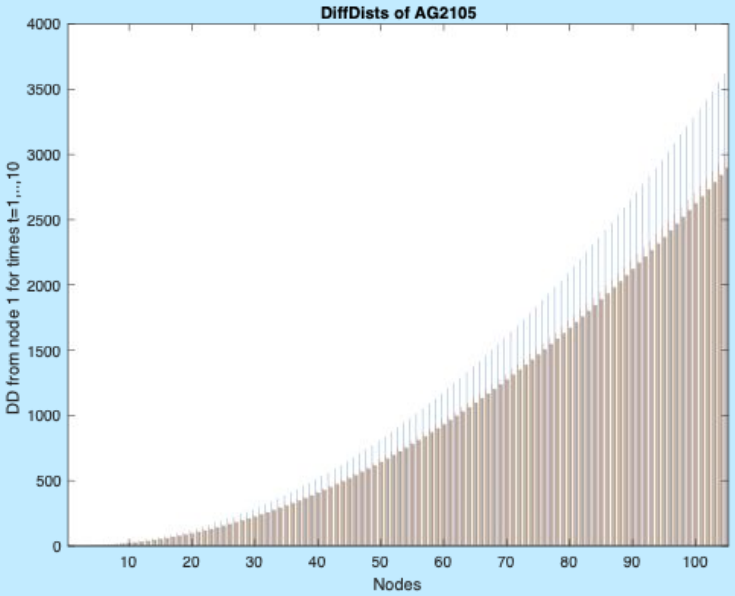
In words, we measure the difference between the probability of reaching a ***third*** node *y* in a random walk of length *t* starting at node x_0 and the probability of reaching *y* in a random walk of length *t* starting at node x_1.

Time vs. diffusion distance

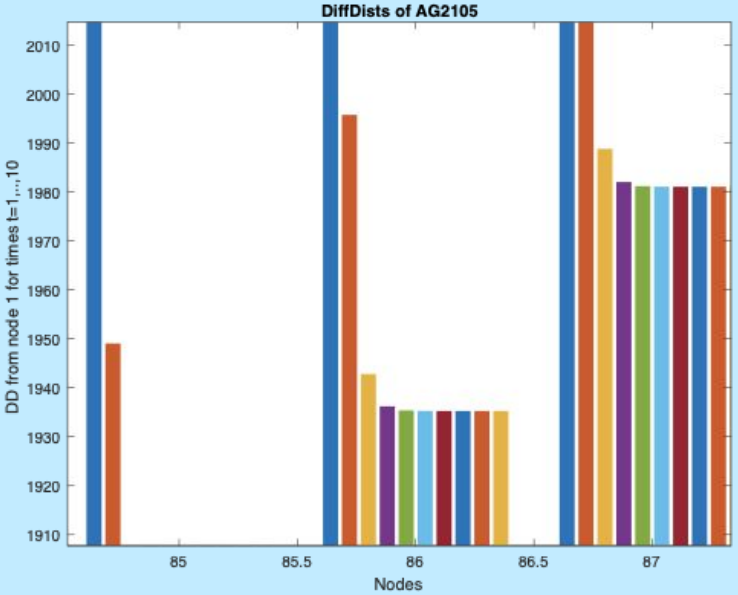
Let’s elaborate a little bit on how changing time affects the diffusion distance of a graph.

On the left is a graph called “AG2105.” On the right is a bar graph which models the change in diffusion distance of its nodes as time increases.





Convergence of diffusion distance



- The central 10 columns show the diffusion distance between node 1 and node 86 of the graph at the times 1 through 10.
- Diffusion distance ***decreases*** as a function of time, because the probabilities of the random walks stabilize and converge to fixed values as time goes to infinity.
- The more strongly connected a graph is, the faster the convergence to the stable distribution of probabilities.

Construction of the diffusion distance

- The adjacency, degree, and Laplacian matrices can be used to describe graphs
- The ***transition matrix*** gives the probability of hopping from node *i* to node *j* at random and can be derived from the Laplacian
- The ***diffusion distance*** uses random walks to measure a probabilistic distance between nodes on a graph, and uses time as a parameter
    - As time increases, the diffusion distance between two nodes decreases

We are now ready to discuss Graph Matching algorithms.

Part III: The Graph Matching Problem

- Consider a graph called *G1*
- We will “shuffle” or *permute* the nodes of *G1* using a randomized algorithm
- This new shuffled graph is called *G2*
- Using our Graph Matching Algorithm, we should be able to ***measure precisely how different G1 and G2 are, node-by-node***
- The algorithm should also tell us how we can ***reshuffle G2 to make it identical to G1 once more.*** Formally, this is called “recovering the original permutation.”

Generating the cost matrix with diffusion distance

The (i,j)-entry (row *i*, column *j*) of the ***cost matrix*** measures the difference in similarity (the other nodes to which *i* and *j* are connected) between node *i* of *G1* and node *j* of *G2*.

1. Decide on a time *t* for the diffusion distance measurement
2. Measure the following relationships:
- *v1* gives the distances between node *i* of *G1* and the other nodes of *G1* at time *t*
- *v2* gives the distances between node *j* of *G2* and the other nodes of *G2* at time *t*
3. Encode each bullet point as a vector in $\mathbb{R}^d$
4. Take the Euclidean distance of *v1* and *v2*
5. The result is the row-*i*, column-*j* entry of the cost matrix

The Graph Matching Problem

$$\min_{P \in \Pi} ||AP - PB||_F^2 + \beta \operatorname{tr}(C^T P)$$

$\mathbf{\Pi}$ denotes the set of ***permutation matrices***. We want to find the permutation matrix which minimizes the difference between graphs *A* and *B* plus the trace (Tr) of the permuted cost matrix.

$C^T P$ is a permuted version of the cost matrix. Tr($C^T P$) measures the ***trace*** (the sum of the eigenvalues) of this shuffled cost matrix.

β is a parameter by which we scale the trace of the cost matrix.

Thus, the Graph Matching Problem can be thought of as writing an algorithm that finds the "*optimal shuffle*" of graph *B* to make it similar in shape and structure to graph *A*.

The first objective is a *global* measure of difference; the second objective is a *local* measure of difference.

An algorithm for graph matching

$$\min_{P \in \Pi} \left|\left|AP - PB\right|\right|_F^2 + \beta \operatorname{tr}(C^T P)$$

1. Take graphs *G1* and *G2* as inputs
2. Compute their adjacency matrices, *A* and *B*
3. Use diffusion distance to construct the cost matrix *C*
4. Use convex optimization algorithms to find the optimal permutation *P*
5. Use this permutation matrix *P* to compute the optimal shuffling of the graphs

While graph matching is a well-studied field, this is the first attempt that we have found that uses diffusion distance to solve the problem.

References

- Boaz, Nadler *et al*. “Diffusion maps, spectral clustering and eigenfunctions of Fokker-Planck operators.” Neural Information Processing Systems, No. 18. December 2005.
- Chung, Fan. *Spectral Graph Theory*. CBMS No. 92. Providence: American Mathematical Society. 1997.
- Lovásv, L. “Random walks on graphs: a survey.” Bolyai Society Mathematical Studies, Vol. 2. 1993.
- Narayanan, Arvind and Shmatikov, Vitaly. “De-anonymizing social networks.” 30th IEEE Symposium on Security and Privacy. 2009.
- Nica, Bogdan. *A Brief Introduction to Spectral Graph Theory*. Zürich: European Mathematical Society. 2018.