Meeting notes 28-9-2017: Graph Matching

Geoff Iyer

Quick Summary

Using graph match, we can come up with some kind of feature comparison algorithm. I'm not sure what to call it. It's not exactly change detection but it's similar. Here is a rough overview. Given coregistered datasets X, Y (so x_i corresponds to y_i), we do a graph match to get a second registration $\rho : X \to Y$. We then compare the different registrations. Specifically, look at the matches $x_i \to y_{\rho(i)}$ where the calculated match weight is strong. Then for those particular i, compare x_i to $x_{\rho^{-1}}(i)$ and y_i to $y_{\rho(i)}$. In other words, look for points where the graphs match well, but it does not agree with the actual coregistration. See examples at the end.

Problem Statement and Background Theory

First we recall the problem statement. Let $G = (X, W_X)$, $H = (Y, W_Y)$ be undirected, weighted graphs. Here X, Y represent the nodes of G, H (respectively), and W_X, W_Y are the corresponding matrix of edge weights. For convenience of notation we will assume $|X| = |Y| = N$. The extension to the general case is quite straightforward. The goal of the Weighted Graph Matching Problem (WGMP) is to find a bijection $\rho : X \to Y$ that minimizes the squared difference of edge weights. Phrased in terms of matrices, our minimization problem becomes

$$\operatorname{argmin}_{P \text{ a permutation matrix}} \left\| P W_X P^T - W_Y \right\|_F^2 . \tag{1}$$

This problem is NP-hard, so we relax and look for an orthogal matrix.

$$Q^* = \operatorname{argmin}_{QQ^T = I} \left\| Q W_X Q^T - W_Y \right\|_F^2 . \tag{2}$$

This problem is solved via eigenvectors of the graph laplacian. Let $L_X = U_X \Lambda_X U_x^t$, and $L_Y = U_Y \Lambda_Y U_Y^T$. Then the solution to 2 is given by

$$Q^* = U_Y^T S U_X , \tag{3}$$

where S is a diagonal matrix with values ± 1 to account for the sign ambiguity in eigenvectors.

Right now we have some ideas of how to determine S, but we don't have a solution that we really like. The sample code uses one pre-known match and extrapolates from there to determine S. There are some other ideas in the literature that we could try out.

Current Work

Once we have Q^* we have to decide what to do. The ideas we've implemented are

1. Use Hungarian Algorithm to get a one-to-one matching

   - It's very slow to use hungarian algorithm directly. So we created the hierarchical match plan. We do the match on a coarse version of the graph, then lift to the full graph.
   - There is also code written for a semisupervised version in which some matches are given, and the algorithm fills out the rest from there.

2. Do the most naive thing: for each node in X, choose its best match in Y. Then similar from Y to X (so we get a many-to-one matching, and a one-to-many matching).

   - Surprisingly this works decently well.

Let's call the final assignment

$$\rho : \{1, 2, \ldots, N\} \to \{1, 2, \ldots, N\}. \tag{4}$$

Last time we worked through this, we took a coregistered set and looked at the differences

$$\begin{gathered} \left\| x_i - x_{\rho^{-1}(i)} \right\| \\ \left\| y_i - y_{\rho(i)} \right\| . \end{gathered} \tag{5}$$

I played with this for a while, and found that we should specifically look at the matches $i \to \rho(i)$ where the corresponding match weight in Q^* is large. That is, we are most interested in the points where Q^* gives us a good match. Heuristically this makes sense, because if graph match is weak in some area then it makes no sense to try to transfer information through this match.

Examples

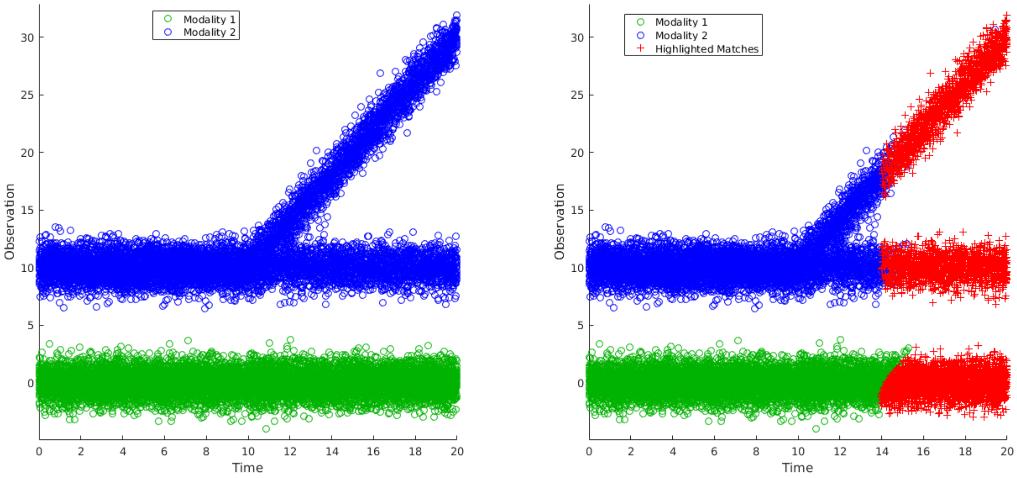

Figure 1: Synthetic Dataset

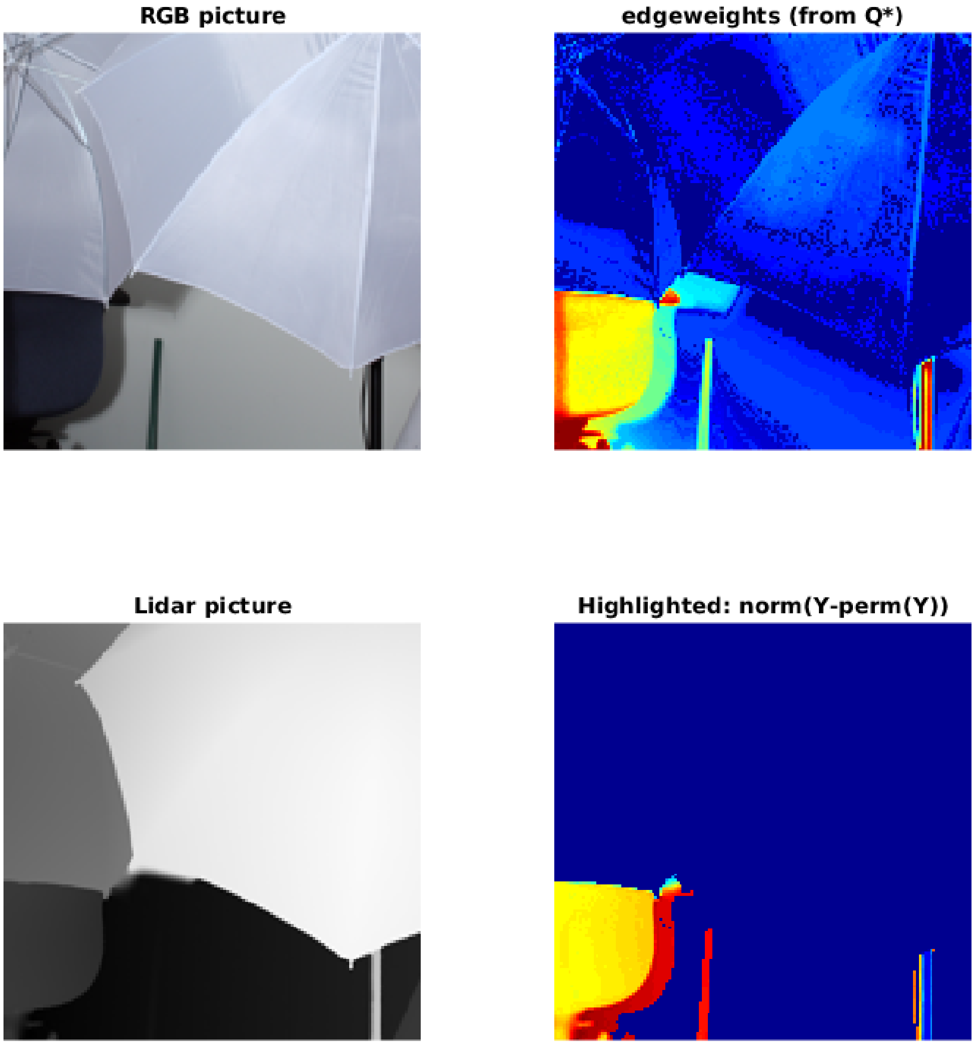

Figure 2: Umbrella Dataset

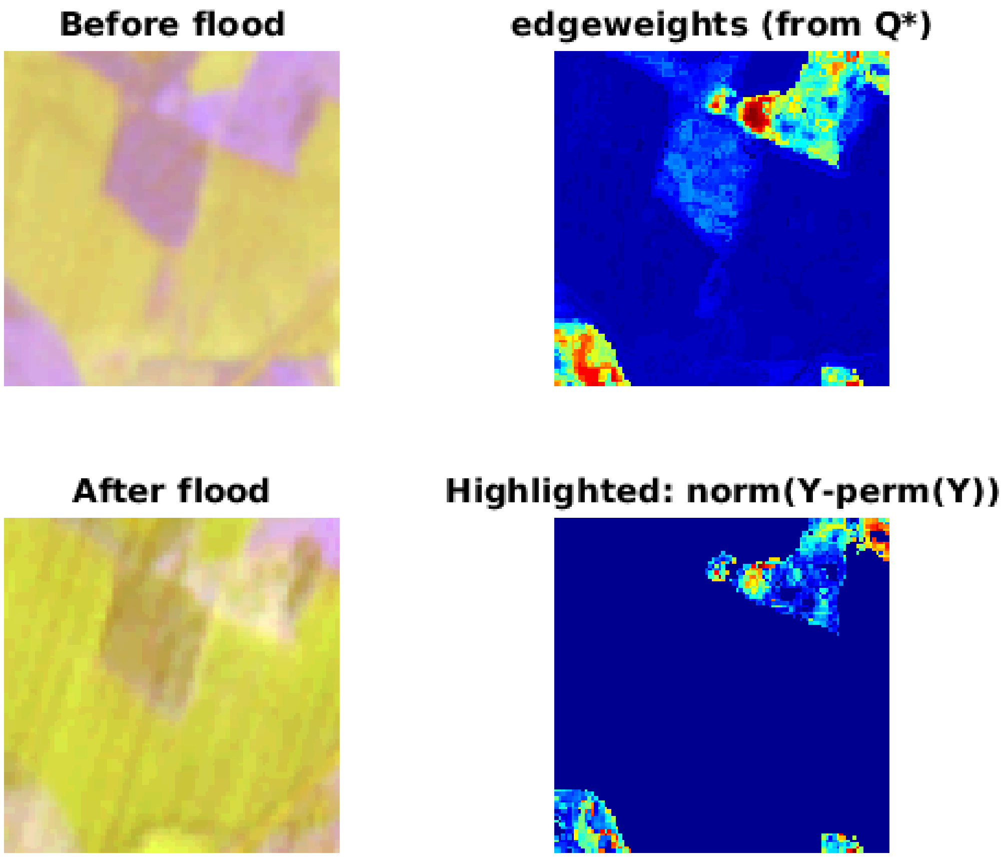

Figure 3: Flood Dataset