# Latent Labeled Graph Classification

Joshua T. Vogelstein, R. Jacob Vogelstein, and Carey E. Priebe

**Abstract**—Because graphs can encode more information in their structure than vectors, they are becoming increasingly popular data structures for representing information. While the last century has witnessed the development of a plethora of statistical tools for the analysis of data, the vast majority of these tools natively operate in vector spaces, not graph spaces. Thus, algorithms for even relatively simple statistical inference tasks, such as two-class classification, are essentially absent for graph data. In this work, we propose a number of complementary algorithms to classify graphs, with special attention to the possibility of unknown vertex labels. Since exactly solving the graph-matching problem is currently computational intractable, we consider several approximate approaches. We introduce a multiple-restart Frank-Wolfe approach to solving the graph matching problem by formulating it as a quadratic assignment problem. Although this approach has superior performance than previous state-of-the-art approaches to the graph matching problem, even when it "should" do well in classification problems, it is outperformed by a graph invariant strategy. This is just the beginning.

**Index Terms**—statistical inference, graph theory, network theory, structural pattern recognition, connectome.

## 1 Introduction

THE statistical analysis of collections of graphs is becoming an increasingly popular desideratum [?]. Specifically, we consider the following idealized scenario. Let $\mathbb{G} : \Omega \mapsto \mathcal{G}$ be a graph-valued random variable taking values $G \in \mathcal{G}$. Let $\mathbb{Y}$ be a Bernoulli random variable, $\mathbb{Y} : \Omega \mapsto \mathcal{Y} \subseteq \mathbb{Z}$, such that each graph has an associated class. Given a collection of graphs and classes, we assume they were jointly sampled independently and identically from some true but unknown joint distribution, $\{(\mathbb{G}_i, \mathbb{Y}_i)\}_{i\in[n]} \overset{iid}{\sim} F_{\mathbb{G},\mathbb{Y}}[\cdot;\boldsymbol{\theta}]$, where $\boldsymbol{\theta}$ is a set of parameters. Note that $F_{\mathbb{G},\mathbb{Y}}[\cdot;\boldsymbol{\theta}]$ is but one of a (possibly infinite) set of distributions, collectively comprising the model: $\mathcal{F}_{\mathbb{G},\mathbb{Y}} = \{F_{\mathbb{G},\mathbb{Y}}[\cdot;\boldsymbol{\theta}] : \boldsymbol{\theta} \in \boldsymbol{\Theta}\}$, where $\boldsymbol{\Theta}$ is the set of feasible parameters. The goal of such an analysis is to learn about the relationship between the random variables $\mathbb{G}$ and $\mathbb{Y}$. Most standard classification techniques fail in this domain as they typically require classifying objects that live in finite dimensional Euclidean space, $\mathbb{R}^d$, whereas the object of interest here live in graph space, $\mathcal{G}$ (even finite graphs do not natively live in Euclidean space). In this work, therefore, we propose novel extensions of several classification algorithms appropriate for the graph domain.

## 2 Graph Classification

The graph classification problem may be stated thusly: given training data $\mathcal{T}_n = \{(\mathbb{G}_i, \mathbb{Y}_i)\}_{i\in[n]}$, and a new graph, $\mathbb{G}$, estimate the new graph's corresponding class, $\mathbb{Y}$, assuming each graph/class pair was sampled identically and independently from some true but unknown distribution, $\mathcal{T}_n, (\mathbb{G}, \mathbb{Y}) \overset{iid}{\sim} F_{\mathbb{G},\mathbb{Y}}[\cdot;\boldsymbol{\theta}]$ . Given an appropriately defined risk-function, such as misclassification rate: $R_h = \mathbb{P}[h(\mathbb{G}) \neq \mathbb{Y}]$, one can then search for the function $h^* \in \mathcal{H}$ that minimizes the loss function of interest:

$$h^* = \operatorname*{argmin}_{h\in\mathcal{H}} \mathbb{P}[h(\mathbb{G}) \neq \mathbb{Y}]. \tag{1}$$

In general, $h^*$ is unavailable and dependent on the true but unknown distribution, $F = F_{\mathbb{G},\mathbb{Y}}$ (which includes the vertex labels). When $h^*$ is unavailable, one can utilize training data, $\mathcal{T}_n$, to obtain $\hat{h}$, an approximation to $h^*$:

$$\hat{h} \approx \operatorname*{argmin}_{h\in\mathcal{H}} \mathbb{P}[h(\mathbb{G}) \neq \mathbb{Y}|\mathcal{T}_n], \tag{2}$$

where $\approx$ indicates that in general, we will not be able to find the actual minimum in the set $\mathcal{H}$. Regardless, any approach necessarily estimates a decision boundary in the space of graphs separating them into two classes.

## 3 Sources of Variability

Let a graph be a 4-tuple consisting of a set of (i) vertices, (ii) edges, (iii) vertex labels, and (iv) edge attributes. Assume that in our collection of graphs, each graph has the same set of vertices. Moreover, assume that edge attributes can be represented as scalar quantities, so the edges and edge attributes of a random graph are jointly encoded in an adjacency matrix representation. Thus, the joint distribution on graphs and classes can be expanded and factorized:

$$F_{\mathbb{G},\mathbb{Y}} = F_{\mathbb{L},\mathbb{A},\mathbb{Y}} = F_{\mathbb{L},\mathbb{A}|\mathbb{Y}} F_{\mathbb{Y}} \tag{3}$$

where $\mathbb{L} : \Omega \mapsto \mathcal{L} \subseteq \{L_1, L_2, \ldots, L_{n_V}\} = [L_{n_V}]$ is a random vertex labeling, $\mathbb{A} : \Omega \mapsto \mathcal{A} \subseteq \mathbb{R}^{n_V \times n_V}$ is a random adjacency matrix, and $n_V$ is the number of vertices in the graphs. The class-conditional variability therefore derives from two possible sources of variability,

• *J.T. Vogelstein and C.E. Priebe are with the Department of Applied Mathematics and Statistics, Johns Hopkins University, Baltimore, MD 21218.*
*E-mail: joshuav@jhu.edu*
• *R.J. Vogelstein is with the Johns Hopkins University Applied Physics Laboratory, Laurel, MD, 20723.*

$\mathbb{L}$ and $\mathbb{A}$. Given this insight, we consider three distinct graph classification scenarios.

### 3.1 Labeled Graph Classification

In *labeled* graph classification, we observe $(\mathbb{L}, \mathbb{A}), \{(\mathbb{L}_i, \mathbb{A}_i, \mathbb{Y}_i)\}_{i \in [n]} \overset{iid}{\sim} F_{\mathbb{L},\mathbb{A},\mathbb{Y}}$, and we desire to know $\mathbb{Y}$, the class of the as yet unclassified labeled graph. An example of such a scenario is MR connectome classification, where the vertices correspond to macro-neuroanatomical regions, such visual cortex. Because it is relatively easy to draw reasonable boundaries between regions, these inferred graphs have labeled vertices. And because the vertex labels of each graph are observed, we can compare adjacency matrices directly, and use graph classification techniques, such as those described in [?]. This facilitates factorizing the likelihood term of Eq. (3): $F_{\mathbb{L},\mathbb{A}|\mathbb{Y}} = F_{\mathbb{A}|\mathbb{L},\mathbb{Y}} F_{\mathbb{L}|\mathbb{Y}}$. Note that although adjacency matrices can be represented by objects in $\mathbb{R}^{n_v \times n_V}$, the graphs are *not* in finite Euclidean space; rather, the structure of the adjacency matrix may encode information. In other words, simply vectorizing the adjacency matrix, and performing some standard classification techniques, might be discarding class-conditional signal. A labeled graph classifier therefore maps from the joint vertex label and edge attribute space into the class space: $h_{\mathbb{L},\mathbb{A}} : \mathcal{L} \times \mathcal{A} \mapsto \mathcal{Y}$.

### 3.2 Unlabeled Graph Classification

In *unlabeled* graph classification, the labels are assumed to be uninformative with regard to our inference task; that is, it is assumed that the labels lack class-conditional signal. In this context, the likelihood term of Eq. (3) factorizes: $F_{\mathbb{L},\mathbb{A}|\mathbb{Y}} = F_{\mathbb{A}|\mathbb{Y}}$. A common scenario in which this assumption is made is in classifying chemical compounds, where only graph structure is used [?]. In this context we observe $\mathbb{A}, \{(\mathbb{A}_i, \mathbb{Y}_i)\}_{i \in [n]} \overset{iid}{\sim} F_{\mathbb{A},\mathbb{Y}}$, and we desire to know $\mathbb{Y}$, the class of the as yet unclassified unlabeled graph. Here, any arbitrary permutation of each of the adjacency matrices is equivalent to any other. Letting $Q \in \mathcal{Q}$ be a permutation matrix, pre and post-multiplying an adjacency matrix by a permutation matrix yields another adjacency matrix representation, $\widetilde{A} = QAQ^{\mathsf{T}}$. Thus, in the unlabeled graph classification setting, each graph implies a quotient space, $G = \{QAQ^{\mathsf{T}} : Q \in \mathcal{Q}\}$. An unlabeled graph classifier therefore maps from this quotient space, which is just the edge attribute space, into the class space: $h_{\mathbb{A}} : \mathcal{A} \mapsto \mathcal{Y}$.

### 3.3 Latent Labeled Graph Classification

In *latent labeled* graph classification, the labels latent, and it is unknown whether they contain any class-conditional signal. In this context, the likelihood term of Eq. (3) does not necessarily factorize. Moreover, we observe $\mathbb{A}, \{(\mathbb{A}_i, \mathbb{Y}_i)\}_{i \in [n]} \overset{iid}{\sim} F_{\mathbb{L},\mathbb{A},\mathbb{Y}}$, and we desire to know $\mathbb{Y}$, the class of the as yet unclassified unlabeled graph, and we have not observed any labels, although they may contain class-conditional signal (they may not). Given such a scenario, one could consider at least two complementary approaches:

1) Try to impute the missing labels, and use a labeled graph classifier, $h_{\mathbb{L},\mathbb{A}}$.
2) Ignore the missing labels, and use an unlabeled graph classifier, $h_{\mathbb{A}}$.

The preferred strategy will depend on the assumed model and the data. Specifically, if one has good reason to believe that the vertex labels lack much class-conditional signal, then one “should” ignore them. Alternately, if one believes that the vertex labels might contain some class-conditional signal, then one “could” try to use them. However, whether those efforts are fruitful will depend on a bias-variance trade-off. Certainly, the class-conditional entropy for the joint vertex labels and edge attributes is at least as great as the class-conditional entropy for the edge attributes alone, $\mathcal{I}(\mathbb{L}, \mathbb{A}|\mathbb{Y}) \geq \mathcal{I}(\mathbb{A}|\mathbb{Y})$. But if the vertex have labels, ignoring them will induce more bias.

Another view of how to determine whether to try using the labels concerns the relative effective signal-to-noise-ratios. Specifically, if given the vertex labels, the adjacency matrices are all very similar, then $F_{\mathbb{A}|\mathbb{L},\mathbb{Y}}$ has relatively low entropy. Therefore, one can try to impute the missing labels with perhaps great success. Alternately, if $F_{\mathbb{A}|\mathbb{L},\mathbb{Y}}$ has relatively high entropy, then imputing the missing labels might be too difficult, and we are better off using only $F_{\overset{\circ}{A}|\mathbb{Y}}$.

It is our contention that many interesting graph classification problems can be cast as latent labeled graph classification problems.

The sequel considers this latent labeled graph classification setting from a statistical perspective. We propose two complementary approaches, one that tries to impute the missing labels, and one that does not. Via synthetic data analysis, we demonstrate scenarios in which the above intuition regarding the bias-variance trade-off suggests which strategy is likely more effective.

## 4 Latent Labeled Graph Classifiers

### 4.1 Trying to use the labels

Deciding the try to use the labels to classify

#### *4.1.1 Graph Matching*

A graph can be represented by its adjacency matrix, $A$, assuming the edge attributes are univariate. Unlabeled graphs, on the other hand, can be represented by a set of adjacency matrices, $\{QAQ^{\mathsf{T}} : Q \in \mathcal{Q}\}$, where $Q$ is any permutation matrix. Given a pair of unlabeled graphs, determining whether they are isomorphic with respect to one another is equivalent to determining whether one can find an adjacency matrix of one graph that is identical to the other's. This problem can be cast as a *quadratic assignment problem* (QAP):

$$Q_{QAP} \triangleq Q_{QAP}(A,B) = \underset{Q \in \mathcal{Q}}{\operatorname{argmin}} \left\| QAQ^{\mathsf{T}} - B \right\|_F^2, \quad (4)$$

where $A$ and $B$ are adjacency matrix representations of two different graphs. A bit of linear algebra simplifies Eq. (4):

$$\begin{aligned} &\underset{Q \in \mathcal{Q}}{\operatorname{argmin}} \left\| QAQ^{\mathsf{T}} - B \right\|_F^2 \\ &\qquad = \underset{Q \in \mathcal{Q}}{\operatorname{argmin}} -tr(B^{\mathsf{T}} QAQ^{\mathsf{T}}) - tr(QAQ^{\mathsf{T}} B), \quad (5) \end{aligned}$$

which is equivalent to the standard representation of the quadratic assignment problem [**?**]:

$$\hat{\sigma} = \underset{\sigma}{\operatorname{argmin}}\, a_{\sigma(i),\sigma(j)} b_{ij} = \underset{q \in \mathcal{Q}}{\operatorname{argmin}}\, q_{ij} a_{ij}, q_{ji} b_{ij} \quad (6)$$

where $\sigma$ is a permutation function, that is, $\sigma : [n] \mapsto [n]$. Unfortunately, Eq. (4) is an NP-complete problem [**?**]. The primary difficulty in solving Eq. (4) is the discrete non-convex constraint set. Thus, it is natural to consider an approximation with the constraints relaxed. Since the convex hull of permutation matrices is the set of doubly stochastic matrices, we define the approximate quadratic assignment problem:

$$Q_{AQAP} \triangleq Q_{AQAP}(A,B) = \underset{Q \in \mathcal{D}}{\operatorname{argmin}} \left\| QAQ^{\mathsf{T}} - B \right\|_F^2, \quad (7)$$

where $\mathcal{D}$ is the set of doubly stochastic matrices. When the permutation matrix constraint is relaxed, the equivalence relation shown in Eq. (5) no longer holds. Nonetheless, we proceed by attempting to solve:

$$\hat{Q}_{AQAP} \approx \underset{Q \in \mathcal{D}}{\operatorname{argmin}} -tr(B^{\mathsf{T}} QAQ^{\mathsf{T}}) - tr(QAQ^{\mathsf{T}} B), \quad (8)$$

considering it an auxiliary function for which we can compute gradients and ascend a likelihood, unlike the permutation constrained case.

The Frank-Wolfe (FW) algorithm is a successive linear programming algorithm for nonlinear programming problems; specifically, for quadratic problems with linear (equality and/or inequality) constraints. Let $f(Q) = -tr(B^{\mathsf{T}} QAQ^{\mathsf{T}}) - tr(QAQ^{\mathsf{T}} B)$. With each iteration $j$, the FW algorithm takes the following steps:

**Step 1: Compute the gradient** The gradient of $f$ with respect to $Q$ is given by:

$$\nabla_Q^{(j)} = \partial f / \partial Q^{(j)} = AQ^{(j)} B^{\mathsf{T}} + A^{\mathsf{T}} Q^{(j)} B. \quad (9)$$

**Step 2: Find the closest doubly stochastic matrix** Instead of directly descending this gradient, we search for the direction of the doubly stochastic matrix closest to this gradient. Noting that that direction may be computed by the dot-product operator, we have:

$$W^{(j)} = \underset{W^{(j)} \in \mathcal{D}}{\operatorname{argmin}} \langle \nabla_Q^{(j)}, W^{(j)} \rangle. \quad (10)$$

Eq. (10) can be solved as a Linear Assignment Problem (LAP). More specifically, a LAP can be written as:

$$Q_{\mathrm{LAP}} \triangleq Q_{\mathrm{LAP}}(A,B) = \underset{Q \in \mathcal{Q}}{\operatorname{argmin}} \left\| QA - B \right\|_F^2, \quad (11)$$

which, when $B = I$, can be simplified:

$$\begin{aligned} Q_{\mathrm{LAP}}(A, I) &= \underset{Q \in \mathcal{Q}}{\operatorname{argmin}} \left\| QA - I \right\|_F^2 \\ &= \underset{Q \in \mathcal{Q}}{\operatorname{argmin}} (QA - I)^{\mathsf{T}} (QA - I) \\ &= \underset{Q \in \mathcal{Q}}{\operatorname{argmin}}\, A^{\mathsf{T}} Q^{\mathsf{T}} QA - 2QA - II \\ &= \underset{Q \in \mathcal{Q}}{\operatorname{argmin}} -\langle Q, A \rangle. \quad (12) \end{aligned}$$

In other words, letting $B = I$, the projection of a matrix onto its nearest doubly stochastic matrix is a LAP problem. While Eq. (12) cannot be solved directly, as above, we can relax the permutation matrix constraint to the doubly stochastic matrix constraint:

$$Q_{\mathrm{LAP}}(A, I) = \underset{Q \in \mathcal{D}}{\operatorname{argmin}} -\langle Q, A \rangle. \quad (13)$$

Since the permutation matrices are the vertices of the set of doubly stochastic matrices, finding the minimum of Eq. (13) is guaranteed to yield a permutation matrix (as minima are necessarily at the vertices). Thus, letting $A = \nabla_Q^{(j)}$, solving Eq. (13)—which is a linear problem with linear and non-negative constraints—is equivalent to solving Eq. (10). Fortunately, the Hungarian algorithm solves any LAP in $\mathcal{O}(n^3)$ [**?**], thus this projection is relatively efficient.[1]

**Step 3: Update the direction** Given $W^{(j)}$, the new direction is given by:

$$d^{(j)} = W^{(j)} - Q^{(j)}. \quad (14)$$

**Step 4: Line search** Given this direction, one can then perform a line search to find the doubly stochastic matrix that minimizes the objective function along that direction:

$$\alpha^{(j)} = \underset{\alpha \in [0,1]}{\operatorname{argmin}} f(Q^{(j)} + \alpha^{(j)} d^{(j)}). \quad (15)$$

This can be performed exactly, because $f$ is a quadratic function.

**Step 5: Update** $Q$ Finally, the new estimated doubly stochastic matrix is given by:

$$Q^{(j+1)} = Q^{(j)} + \alpha^{(j)} d^{(j)}. \quad (16)$$

**The grand finale** Steps 1–5 are iterated until convergence, computational budget limits, or some other stopping criterion is met. These 5 steps collective comprise the FW algorithm. Note that while $Q^{(j)}$ will generally not be a permutation matrix, we do not project $Q^{(j)}$ back onto the set of permutation matrices between each iteration, as that projection requires $\mathcal{O}(n^3)$ time. After

1. More efficient algorithms are available for certain special cases, that is, whenever the matrix-vector multiplication operation is fast (for example, when both $A$ and $B$ are sparse).

the final iteration, however, we have $\hat{Q}_{AQAP}$, which we project onto the set of permutation matrices:

$$\hat{Q}_{QAP} = \operatorname*{argmin}_{Q \in \mathcal{Q}} \langle \hat{Q}_{AQAP}, Q \rangle, \tag{17}$$

which is a LAP, and yields approximate solution to QAP. Let `QAP` indicate this algorithm: FW appended with a projection onto the permutation matrices.

**Multiple restarts** Note that `QAP` will not generally achieve the global optimum even of Eq. (7), because $f$ is not necessarily positive definite. This is clear upon computing the Hessian of $f$ with respect to $Q$:

$$\nabla^2_Q = B \otimes A + B^\mathsf{T} \otimes A^\mathsf{T}, \tag{18}$$

where $\otimes$ indicates the Kronecker product. This means that the initialization, $Q^{(0)}$, will be important. While any doubly stochastic matrix would be a feasible initial point, two choices seem natural: (i) the "flat doubly stochastic matrix," $J = \mathbf{1}^\mathsf{T}\mathbf{1}/n_V$, which is the middle of the feasible region, and (ii) the identity matrix, which is a permutation matrix. Therefore, if we run the FW algorithm once, we always start with one of those two. If we use multiple restarts, each initial point is "near" the flat matrix. Specifically, we sample $J'$, a random doubly stochastic matrix using 10 iterations of Sinkhorn balancing [?], and let $Q^{(0)} = (J + J')/2$. We refer to multiple re-starts of `QAP` with subscripts, that is, the performance of `QAP`$_n$ is the best result of $n$ pseudo-random re-starts of `QAP`. Note that `QAP` natively operates on matrices, which could correspond to either weighted or unweighted graphs.

### 4.2 Graph Invariants

A graph invariant (GI) is any function that maps a graph to a scalar whose value is independent of the vertex labels, $T : (\mathcal{V}, \mathcal{E}, \mathcal{A}) \mapsto \mathbb{R}$ (note that often graph invariants are defined to operate only on unweighted graphs, that is, $T : (\mathcal{V}, \mathcal{E}) \mapsto \mathbb{R}$). By defining a set of GIs, one can embed a collection of graphs into a quotient space invariant to vertex labels. Whenever this quotient space is a subset of finite dimensional Euclidean space, all standard machine learning classifiers may be implemented to solve the classification problem.

## 5 Unlabeled Graph Classification Algorithms

Below we provide an example, and briefly discuss, three complementary approaches to solving unlabeled graph classification problems. Each strategy uses one of the two above approaches to dealing with the graph isomorphism problem as a subroutine.

### 5.1 A graph dissimilarity approach: $k$NN $\circ$ QAP

Define a dissimilarity on graph spaces: $d : \mathcal{G} \times \mathcal{G} \mapsto \mathbb{R}_+$, in which one can compute the dissimilarity between any pair of graphs. Given an adjacency matrix representation, many such dissimilarities are possible (e.g., graph edit distance, Hamming distance, etc.). It is becoming increasingly popular to use a *graph kernel*, $\kappa(G, G') = \langle \phi(G), \phi(G') \rangle$, where $\phi(\cdot) : \mathcal{G} \mapsto \mathbb{E}$, as the dissimilarity [?]. Graph kernels have a number of desirable properties, perhaps most notably, that one can then use standard *kernel machines* to classify [?]. Regardless, given any dissimilarity, one can apply at least two different kinds of classifiers.

First, one could implement a nearest neighbor style classifiers, such as the $k_n$ nearest neighbor ($k$NN) classifier. In addition to being universally consistent[2], $k$NN classifiers are computationally efficient, in that they only require $n{+}1$ graph dissimilarity computations (assuming a single test graph).

Alternately, one could implement an interpoint-dissimilarity matrix based algorithm [?]. This strategy has the advantage of using all available information to generate a class prediction, but a disadvantage that it requires $(n+1)^2$ graph dissimilarity computations. Moreover, it may be more sensitive to outliers.

For simplicity, we only consider the $k$NN strategy here. Specifically, given a test adjacency matrix, $A$, find $\hat{Q}^A_i = \hat{Q}_{QAP}(A, B_i)$ for all $n$ training adjacency matrices, $\{B_i\}_{i \in [n]}$. Given these solutions, let $\widetilde{A}_i = \hat{Q}^A_i A \hat{Q}^{A\mathsf{T}}_i$ for all $i$. Given a suitable dissimilarity $d : \mathcal{A} \times \mathcal{A} \mapsto \mathbb{R}_+$, one can compute $d(\widetilde{A}_i, B_i)$ for all $i \in [n]$, and sort them: $d_{(1)} \leq d_{(2)} \leq \cdots \leq d_{(n)}$. Let the $k_n$ nearest neighbors of $A$ be the graphs with the $k_n$ smallest distances, $\{d_{(1)}, \ldots, d_{(j)}\}$. The estimated class of the training sample $A$ is then the plurality class of the $k_n$ nearest neighbors: $\hat{y} = \operatorname{argmax}_y \mathbb{I}\{\sum_{i \in [k_n]} y_{(i)} = y\}$. Call this algorithm $k$`NN` $\circ$ `QAP`.

### 5.2 A graph model approach: BPI$\circ$QAP

First assume a model, $\mathcal{F}_{\mathbb{G},\mathbb{Y}}$. The model of graphs and classes may be factorized into (i) a class-conditional random graph model: $\mathcal{F}_{\mathbb{G}|\mathbb{Y}} = \{F_{\mathbb{G}|\mathbb{Y}}[\cdot; \boldsymbol{\theta}_\mathbb{Y}] : \boldsymbol{\theta}_\mathbb{Y} \in \boldsymbol{\Theta}\}$, and (ii) a class prior model, $\mathcal{F}_\mathbb{Y} = \{F_\mathbb{Y}[\cdot; \pi_\mathbb{Y}] : \pi_\mathbb{Y} \in (0,1)\}$. Given such a factorization, one could then, for instance, estimate $\{\boldsymbol{\theta}, \pi\}$ and then use standard model-based classifiers, such as the Bayes plugin (BPI) classifier.

However, if all the training graphs are unlabeled, unless the class-conditional signal is independent of the vertex labels, one must somehow deal with the absent labels. Sometimes a "canonical" labeling is somehow natural, and can be searched for. Otherwise, one could `QAP` all the training graphs to the test graph. This approach has the unfortunate consequence of making all the training graphs more similar to one another, disregarding the class labels. Another option would be to select a "prototype" from each class. Although several prototype selection strategies have been studied, prototype selection is still problematic [?]. Finally, one could

2. A sequence of $k$NN classifiers is guaranteed to converge to the Bayes optimal classifier if as $n \to \infty$, $k \to \infty$ but $k/n \to 0$ [?].

generate prototype for each class, using all available information. This approach is intriguing but complicated. Each of these alternatives essentially defines a canonical labeling per class.

Regardless of how a canonical labeling is chosen per class, given this canonical labeling, a graph model approach takes the following steps. First, align each graph in each class to its class prototype using $\texttt{QAP}_n$. Second, estimate the likelihood parameters for each class, $\{\boldsymbol{\theta}_{\mathbb{Y}}\}_{\mathbb{Y}\in\{0,1\}}$, as well as the class-priors. Third, the test graph can then be $\texttt{QAP}$ed to the prototype of each class. Fourth, compute the posterior probability of the test graph coming from each class, by plugging in the parameters to the Bayes classifier. Finally, let the estimated class be the *maximum a posteriori* class. Call this algorithm $\texttt{BPI} \circ \texttt{QAP}$.

### 5.3 A graph invariant approach: $\texttt{CW}\circ\texttt{GI}$

To use graph invariants to classify, one first must choose as set of graph invariants. Unfortunately, there is no known set of graph invariants that collectively solve the graph isomorphism problem [?]. Fortunately, some recent theoretical work shows that certain graph invariants have greater discriminability with regard to certain graph inference tasks [?]. With that in mind, we generalize a standard set of unweighted graph invariants—described fully in [?]—developing their weighted graph invariant counterparts. We describe each briefly:

- $T_{weight}$: total weight of all the edges in the graph
- $T_{maxweight}$, is the max over $d(v)$ for all $v \in \mathcal{V}$, where $d(v)$ is the sum of all weights incident to vertex $v$
- $T_{MAW_g}$: a greedy approximation of the maximum average weight (MAW), akin to maximum average degree
- $T_{MAW_e}$: an eigenvalue approximation of MAW
- $T_{wS_1}$: the maximum weighted locality statistic, akin to the typical scan statistic, but sums the weight of edges in each neighborhood rather than just the number of edges.
- $T_{Dijkstra}$: the average Dijkstra path distance between each pair of vertices

For each graph $G_i$ in the training set, we compute a graph invariant vector: $\boldsymbol{T}_i : \mathcal{G} \mapsto \mathbb{R}^d$. We stack these $n$ $d$-dimensional vectors to form a matrix $\boldsymbol{T} \in \mathbb{R}^{n\times d}$. Letting $T_{ij}$ indicate the $i^{th}$ graph invariant of the $j^{th}$ graph, and normalize each element to be between zero and one according to: $T_{ij} \leftarrow \frac{T_{ij}-\min_i(T_{ij})}{\max_i(T_{ij})-\min_i(T_{ij})}$.

Now, to estimate the class of a test graph, we first compute its invariant vector, $\boldsymbol{t}$, and normalize it appropriately. We then apply a variety of machine learning algorithms, including $k$NN, linear classifiers, and SVMs. For the below connectome data, the best performing algorithm is the exact confidence weighted classifier [?]. Call this algorithm $\texttt{CW}\circ\texttt{GI}$.

## 6 Results

### 6.1 QAP benchmarks vs. PATH algorithm

We first compare the performance of $\texttt{QAP}_n$ with recent state-of-the-art approaches on the QAP benchmark library [?]. Specifically, [?] reported improved performance in all but two cases, in which the QPB method of Cremers et al. [?] achieved a lower minimum. We compare $\texttt{QAP}_n$ with the previous best performing algorithm. In *all* cases, $\texttt{QAP}_3$ outperforms the previous best result, often by orders of magnitude in terms of relative error. In three cases, $\texttt{QAP}_{100}$ achieves the absolute minimum. In 12 out of 16 cases, 75%, the simple $\texttt{QAP}_1$ algorithm outperforms the others (starting with the flat doubly stochastic matrix). See Figure 1 for quantitative comparisons.

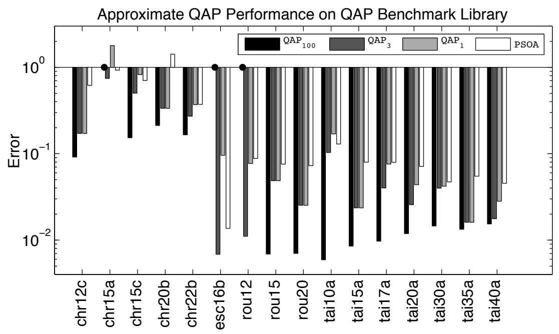

Fig. 1. $\texttt{QAP}_3$ outperforms PSOA on all 16 benchmark graph matching problems. Moreover, $\texttt{QAP}_1$ outperforms PSOA on 12 of 16 tests. For 3 of 16 tests, $\texttt{QAP}_{100}$ achieves the minimum (none of the other algorithms ever find the absolute minimum), as indicated by a black dot. Let $f_*$ be the minimum and $\hat{f}_x$ be the minimum achieved by algorithm $x$. Error is $f_*/\hat{f}_x - 1$.

### 6.2 Unweighted Graph Simulation

$\texttt{QAP}_n$'s near perfect performance gave us hope for using $\texttt{QAP}$ as part of an unlabeled graph classifier. To investigate further, we generated some simulations using the following assumptions. First, assume an independent edge random graph model for each class: $F_y = \prod_{(u,v)\in\mathcal{E}} F_{uv|y}$. For this simple graph scenario, each edge is a Bernoulli random variable, $F_{uv|y} = \text{Bern}(a_{uv}; p_{uv|y})$, where $\{p_{uv|y}\}$ are the likelihood parameters. Then, assume class prior probabilities are equal, $\mathbb{P}[\mathbb{Y} = 1] = \mathbb{P}[\mathbb{Y} = 0] = 1/2$. For simplicity, we sample one graph from each class, meaning $n = 2$, and a single training graph sampled according the class priors. Thus, each simulation is defined by $\mathcal{M} = (P_0, P_1, n_V)$, where $P_0$ and $P_1$ are the class-conditional likelihoods, and $n_V$ is the number of vertices per graph. Given a model, $\mathcal{M}$, we generate $n_{MC}$ Monte Carlo trials. For each model, $R_{chance} \approx 0.5$, as estimated by using BPI to classify without first implementing $\texttt{QAP}$. We estimate Bayes error, $R_*$, by using the true parameters and a Bayes plugin classifier (that is, we use all the labels). We then implement $\texttt{BPI}\circ\texttt{QAP}_1$, and plot both the misclassification rate and objective function $f(Q^{(j)})$ as a function of iteration number (not number of restarts, which we hold fixed at one). Figure 2 shows a model (right panel) and results (left panel) of one such simulation. Intriguingly, the first iteration of $\texttt{QAP}_1$ seems to basically do the trick. This led us to investigate the relationship between LAP and QAP.

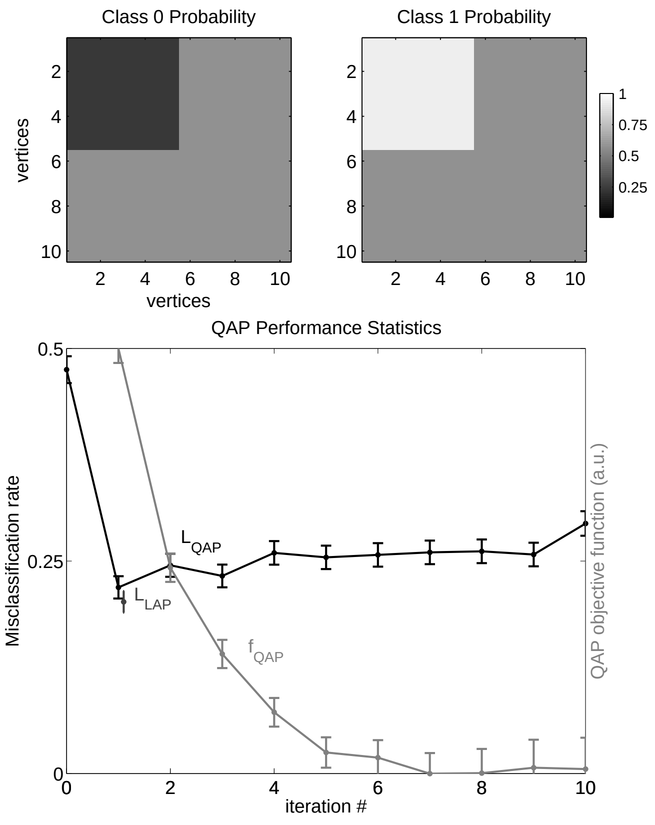

Fig. 2. Homogeneous-kidney-egg model simulation. The left and middle panels show the model parameters for class 0 and 1, respectively. Each edge in the "kidney" in both classes has probability $0.5$; in the egg, class 0 edges are sampled with probability $0.25$, and class 1 edges are sampled with probability $0.75$. The right panel shows the QAP objective function (gray) and misclassification rate (black) as a function of iteration number. LAP does approximately as well as QAP on this (and other) simple graph simulation. Note that $R_{chance} \approx 0.5$ and $R_* \approx 0$ for this simulation. The units of the right-side ordinate are arbitrary. The QAP objective function evaluation prior to any QAP iterations is beyond the bounds of this figure.

## 6.3 LAP vs. QAP

Much like the QAP objective function from Eq. (4) can be simplified to Eq. (5), the LAP objective function can be similarly simplified:

$$\operatorname*{argmin}_{Q\in\mathcal{Q}} \|QA-B\|_F^2 = \operatorname*{argmin}_{Q\in\mathcal{Q}} tr(QAB^\mathsf{T}). \qquad (19)$$

Letting $f_{LAP}(Q) = tr(QAB^\mathsf{T})$, the gradient is:

$$\nabla_{LAP} = 2AB^\mathsf{T}. \qquad (20)$$

Comparing this gradient to that of QAP—Eq. (9)—one can see that when $Q^{(j)}$ is the identity matrix, the two gradients are identical. Thus, if QAP is initialized at the identity matrix, the first permutation matrix—output of Step 2—is identical to $\hat{Q}_{LAP}$; although the line search will make $Q^{(1)} \neq \hat{Q}_{LAP}$, in general. In the above simulation, the first iteration of $\texttt{QAP}$ is essentially the only useful one. Thus, we compare the performance of $\texttt{BPI}\circ\texttt{LAP}$ (dark gray). The performance of LAP and QAP are not statistically different for this simulation. This suggests that for certain problems, LAP (which is $\mathcal{O}(n^3)$) is both an efficient and useful approximation to solving NP-hard graph matching problems. We were unable to find a model for simple graphs in which multiple iterations of $\texttt{QAP}$ improved performance over LAP.

## 6.4 Weighted simulation

Because $\texttt{QAP}_1$ works for both weighted and unweighted, we next simulated multi-graphs (that is, integer valued weights). The below simulation is identical to the above except for $F_{uv|y}$—the edge random variable—was Bernoulli before and is now Poisson: $F_{uv|y} = \text{Poisson}(a_{uv}; \lambda_{uv|y})$. Figure 3 shows misclassification rate steadily decreasing with each iteration.

## 6.5 Connectome Classification

A "connectome" is a graph in which vertices correspond to biological neural units, and edges correspond to connections between the units. Diffusion Magnetic Resonance (MR) Imaging and related technologies are making the acquisition of MR connectomes routine [**?**]. We use 49 subjects from the Baltimore Longitudinal Study on Aging, with acquisition and connectome inference details as reported in [**?**]. For each connectome, we obtain a $70 \times 70$ element adjacency matrix, where each element of the matrix encodes the number of streamlines between a pair of regions, ranging between 0 and about 65,000. Associated with each graph is class label based on the gender of the individual (24 males, 25 females). Because the vertices are labeled, we can compare the results of having the labels and not having the labels. As such, we implement the following classification strategies. In each case, we use a leave-one-out strategy to evaluate performance:

- `N/A-QAP` Using the vertex labels, implement a standard 1NN classifier, where distance is the norm of the difference between any pair of adjacency matrices.
- `1-QAP` Permute only the vertex labels of the test graph, and then implement $\texttt{1NN}\circ\texttt{QAP}_1$ .
- `48-QAP` Permuting the vertex labels, then implement $\texttt{1NN}\circ\texttt{QAP}_1$ .
- `AVG-QAP` Permuting the vertex labels, $\texttt{QAP}_1$ each of the 48 training graphs to the test graph. Then, given

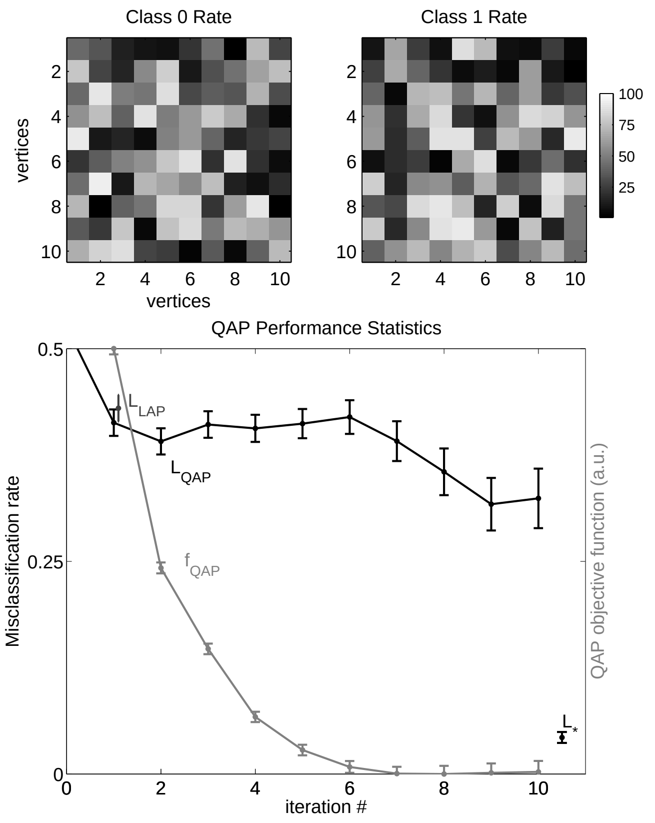

Fig. 3. Multigraph model simulation. The left and middle panels show the parameters for class 0 and 1, respectively. The right panel shows the QAP objective function (gray) and misclassification rate (black) as a function of iteration number; both clearly descending. For comparison, both $\hat{R}_{LAP}$ and $\hat{R}_*$ are shown.

those permuted adjacency matrices, compute the average, and then implement a standard 1NN classifier.

`1NN-GI` Use the graph invariant approach as described above. We provide the normalized graph invariants as inputs into a number of standard classifiers, including $k$NN, linear classifiers, support vector machines, random forests, and CW. On this data, the CW classifier performed best; we therefore only report its results.

Table 1 shows leave-one-out misclassification rates for the various strategies.

TABLE 1
MR Connectome Leave-One-Out Misclassification Rates

| N/A-QAP | 1-QAP | 48-QAP | AVG-QAP | 1NN-GI |
|---|---|---|---|---|
| 20% | 31% | 45% | ?? | 25% |

## 7 Discussion

In this work, we have presented a number of approaches one could take to classifier graphs. Importantly, when the vertex labeling function is unavailable, one must deal with this uncertainty somehow. We compare a number of approaches on both simulated and connectome data. A multiple-restart Frank-Wolfe approach to approximating QAP outperforms previous state-of-the-art approaches in terms of approximating the graph matching problem. Simulations demonstrate that only the first iteration of such an iterative algorithm, starting from the identity matrix, yields classification performance better than chance. Moreover, the first iteration is identical to LAP, which is a linear problem with linear and non-negativity constraints, and therefore can be solved quite easily.

On a connectome dataset, we compare the performance of various `QAP` classification algorithms with several graph invariant (GI) based strategies. Of the algorithms that we tried, a graph invariant approach was most effective, even though, in theory, a QAP based approach could have done better (compare the first and last columns of Table 1).

These analyses leave many open questions. Perhaps most interestingly, when might one expect a QAP-based approach to outperform a GI-based approach? Resorting to a generative model, it should be clear that if the class conditional difference is independent of the vertex labels, then there is no reason to even try to implement graph matching. However, if one believes that the labeling function might convey some class-conditional signal (as in the connectome data), then QAP-based approaches could outperform any approach that ignores the labeling function. Which QAP-based approach to use in such a scenario, however, will depend on many factors, including the assumed model and computational resources.

## Acknowledgments

PLACE PHOTO HERE

**Joshua T. Vogelstein** Joshua T. Vogelstein is a spritely young man, engorphed in a novel post-buddhist metaphor.

**William R. Gray** William R. Gray graduated from Vanderbilt University in 2003 with a Bachelors degree in electrical engineering, and received his MS in electrical engineering in 2005 from the University of Southern California. Currently, Will is a PhD student in electrical engineering at The Johns Hopkins University, where he is conducting research in the areas of connectivity, signal and image processing, and machine learning. He is also a member of the technical staff at the Johns Hopkins University Applied Physics Laboratory, where he manages projects in the Biomedicine and Undersea Warfare business areas. Will is a member of IEEE, Eta Kappa Nu, and Tau Beta Pi

**R. Jacob Vogelstein** R. Jacob Vogelstein received the Sc.B. degree in neuroengineering from Brown University, Providence, RI, and the Ph.D. degree in biomedical engineering from the Johns Hopkins University School of Medicine, Baltimore, MD. He currently oversees the Applied Neuroscience programs at the Johns Hopkins University (JHU) Applied Physics Laboratory as an Assistant Program Manager, and has an appointment as an Assistant Research Professor at the JHU Whiting School of Engineerings Department of Electrical and Computer Engineering. He has worked on neuroscience technology for over a decade, focusing primarily on neuromorphic systems and closed-loop brainmachine interfaces. His research has been featured in a number of prominent scientific and engineering journals including the IEEE Transactions on Neural Systems and Rehabilitation Engineering, the IEEE Transactions on Biomedical Circuits and Systems, and the IEEE Transactions on Neural Networks.

PLACE
PHOTO
HERE

**Carey E. Priebe** Buddha in training.