Topics in Graph Deep Learning

Radu Balan

Department of Mathematics, CSCAMM and NWC
University of Maryland, College Park, MD

October 15, 2019
Robot Learning Workshop
Lehigh University, Bethlehem PA
Acknowledgments

"This material is based upon work partially supported by the National Science Foundation under grant no. DMS-1816608 and LTS under grant H9823013D00560049. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation."

Joint works with:
Naveed Haghani (UMD)  Debdeep Bhattacharya (GWU)
Maneesh Singh (Verisk)
Table of Contents:

1. Permutation Invariant Representations

2. Optimizations using Deep Learning
Permutation Invariant induced Representations

Consider the equivalence relation $\sim$ on $\mathbb{R}^{n\times d}$ induced by the group of permutation $S_n$: for any $X, X' \in \mathbb{R}^{n\times d}$,

$$X \sim X' \iff X' = PX,$$

for some $P \in S_n$

Let $M = \mathbb{R}^{n\times d}/\sim$ be the quotient space endowed with the natural distance induced by Frobenius norm $\| \cdot \|_F$

$$d(\hat{X}_1, \hat{X}_2) = \min_{P \in S_n} \| X_1 - PX_2 \|_F, \quad \hat{X}_1, \hat{X}_2 \in M.$$
Permutation Invariant induced Representations

Consider the equivalence relation $\sim$ on $\mathbb{R}^{n \times d}$ induced by the group of permutation $S_n$: for any $X, X' \in \mathbb{R}^{n \times d}$,

$$X \sim X' \iff X' = PX, \text{ for some } P \in S_n$$

Let $M = \mathbb{R}^{n \times d}/\sim$ be the quotient space endowed with the natural distance induced by Frobenius norm $\| \cdot \|_F$

$$d(\hat{X}_1, \hat{X}_2) = \min_{P \in S_n} \| X_1 - PX_2 \|_F, \quad \hat{X}_1, \hat{X}_2 \in M.$$ 

The Problem: Construct a Lipschitz embedding $\hat{\alpha} : M \to \mathbb{R}^m$, i.e., an integer $m = m(n, d)$, a map $\alpha : \mathbb{R}^{n \times d} \to \mathbb{R}^m$ and a constant $L = L(\alpha) > 0$ so that for any $X, X' \in \mathbb{R}^{n \times d}$,

1. If $X \sim X'$ then $\alpha(X) = \alpha(X')$
2. If $\alpha(X) = \alpha(X')$ then $X \sim X'$
3. $\| \alpha(X) - \alpha(X') \|_2 \leq L d(\hat{X}, \hat{X}')$
Motivation (1)

Graph Learning Problems

Consider data graphs such as: social networks, transportation networks, citation networks, chemical networks, protein networks, biological networks, etc. Each such network is modeled as a (weighted) graph \((\mathcal{V}, \mathcal{E}, A)\) of \(n\) nodes, and a set of feature vectors \(\{x_1^T, \ldots, x_n^T\} \subset \mathbb{R}^d\) that form the matrix \(X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix} \in \mathbb{R}^{n \times d}\).

Two important problems involving a map \(f : (A, X) \rightarrow f(A, X)\):

1. **classification**: \(f(A, X) \in \{1, 2, \ldots, c\}\)
2. **regression/prediction**: \(f(A, X) \in \mathbb{R}\).

In each case we expect the task to be invariant to vertices permutation: \(f(PAP^T, PX) = f(A, X)\), for every \(P \in S_n\).
Motivation (2)
Graph Convolutive Networks (GCN), Graph Neural Networks (GNN)

GCN (Kipf and Welling ('16)) uses $\tilde{A} = I + A$, where $A$ is the adjacency matrix, or the graph weight matrix; GNN uses $\tilde{A} = p(A)$, polynomial in adjacency matrix, or weight matrix. $L$-layer GCN has parameters $(p_1, W_1, B_1, \cdots, p_L, W_L, B_L)$.
Motivation (2)
Graph Convolutive Networks (GCN), Graph Neural Networks (GNN)

GCN (Kipf and Welling ('16)) uses $\tilde{A} = I + A$, where $A$ is the adjacency matrix, or the graph weight matrix; GNN uses $\tilde{A} = p(A)$, polynomial in adjacency matrix, or weight matrix. $L$-layer GCN has parameters $(p_1, W_1, B_1, \cdots, p_L, W_L, B_L)$.

Note the covariance property: for any $P \in O(n)$ (including $S_n$), $(A, X) \mapsto (PAP^T, PX)$ and $B_i \mapsto PB_i$ then $Y \mapsto PY$. 
Motivation (3)
Deep Learning with GCN

The two learning tasks (classification or regression) can be solved by the following scheme:

where $Ext$ is a permutation invariant feature extractor, and SVM/NN is a single-layer or a deep neural network (Support Vector Machine or a Fully Connected Neural Network).

The purpose of this (part of the) talk is to analyze the $Ext$ component.
Motivation (4)
Enzyme Classification Example

Protein Dataset where task is classification into enzyme vs. non-enzyme.
Dataset: 450 enzymes and 450 non-enzymes.

Architecture (ReLU activation):
- GCN with $L = 3$ layers and $d = 25$ feature vectors in each layer;
- No Permutation Invariant Component: $Ext = \text{Identity}$
- Fully connected NN with dense 3-layers and 120 internal units.
The Measure Theoretic Embedding

First approach: Consider the map

\[ \mu : \mathbb{M} \rightarrow \mathcal{P}(\mathbb{R}^d) \ , \ \mu(X)(x) = \frac{1}{n} \sum_{k=1}^{n} \delta(x - x_k) \]

where \( \mathcal{P}(\mathbb{R}^d) \) denotes the convex set of probability measures over \( \mathbb{R}^d \), and \( \delta \) denotes the Dirac measure.

Clearly \( \mu(X') = \mu(X) \) iff \( X' = PX \) for some \( P \in S_n \).

Main drawback: \( \mathcal{P}(\mathbb{R}^d) \) is infinite dimensional!
Two classes of extractors [Zaheer et.al.17’ -’Deep Sets’]:

1. Pooling Map – based on Max pooling
2. Readout Map – based on Sum pooling
Finite Dimensional Embeddings
Architectures

Two classes of extractors [Zaheer et.al.17’ -’Deep Sets’]:

1. Pooling Map – based on Max pooling
2. Readout Map – based on Sum pooling

**Intuition** in the case $d = 1$:

**Max pooling**:

$$
\lambda : \mathbb{R}^n \rightarrow \mathbb{R}^n , \ \lambda(x) = x^{\uparrow} := (x_{\pi(k)})_{k=1}^n , \ x_{\pi(1)} \geq x_{\pi(2)} \geq \cdots \geq x_{\pi(n)}
$$
Two classes of extractors [Zaheer et al. 17’ - ’Deep Sets’]:

1. Pooling Map – based on Max pooling
2. Readout Map – based on Sum pooling

**Intuition** in the case $d = 1$:

**Max pooling:**

$$\lambda : \mathbb{R}^n \rightarrow \mathbb{R}^n, \quad \lambda(x) = x^\downarrow := (x_{\pi(k)})_{k=1}^n, \quad x_{\pi(1)} \geq x_{\pi(2)} \geq \cdots \geq x_{\pi(n)}$$

**Sum pooling:**

$$\sigma : \mathbb{R}^n \rightarrow \mathbb{R}^n, \quad \sigma(x) = (y_k)_{k=1}^n, \quad y_k = \sum_{j=1}^n \nu(a_k, x_j)$$

where kernel $\nu : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, e.g. $\nu(a, t) = e^{-(a-t)^2}$, or $\nu(a = k, t) = t^k$. 
Pooling Mapping Approach

Fix a matrix $R \in \mathbb{R}^{d \times D}$. Consider the map:

$$\Lambda : \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times D} \equiv \mathbb{R}^{nD}, \quad \Lambda(X) = \lambda(XR)$$

where $\lambda$ acts columnwise (reorders monotonically decreasing each column). Since $\Lambda(\Pi X) = \Lambda(X)$, then $\Lambda : \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times D}$.

**Theorem**

For any matrix $R \in \mathbb{R}^{n, d+1}$ so that any $n \times n$ submatrix is invertible, there is a subset $Z \subset \mathbb{R}^{n \times d}$ of zero measure so that $\Lambda : \mathbb{R}^{n \times d} \setminus Z \rightarrow \mathbb{R}^{n \times d+1}$ is faithful (i.e., injective).

No known tight bound yet as to the minimum $D = D(n, d)$ so that there is a matrix $R$ so that $\Lambda$ is faithful (injective). However, due to local linearity, if $\Lambda$ is faithful (injective), then it is stable.
Enzyme Classification Example
Extraction with Hadamard Matrix

Protein Dataset where task is classification into enzyme vs. non-enzyme. Dataset: 450 enzymes and 450 non-enzymes.

Architecture (ReLU activation):

- GCN with $L = 3$ layers and $d = 25$ feature vectors in each layer;
- $Ext = \Lambda$, $Z = \lambda(YR)$ with $R = [I \text{ Hadamard}]$. $D = 50$, $m = 50$.
- Fully connected NN with dense 3-layers and 120 internal units.
Consider:

$$\Phi : \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^m \ , \ (\Phi(X))_j = \sum_{k=1}^{n} \nu(a_j, x_k) \text{ or } (\Phi(X))_j = \prod_{k=1}^{n} \nu(a_j, x_k)$$

where $\nu : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is a kernel, and $x_1, \cdots, x_n$ denote the rows of matrix $X$.

Known solutions: If $m = \infty$, then there exists a $\Phi$ that is globally faithful (injective) and stable on compacts.

Interesting mathematical connexion: On compacts, some kernels $\nu$ define Reproducing Kernel Hilberts Spaces (RKHSs) and yield a decomposition

$$\begin{align*}
(\Phi(X))_j &= \sum_{p \geq 1} \sigma_p f_p(a_j) g_p(X)
\end{align*}$$
Enzyme Classification Example
Feature Extraction with Exponential Kernel Sampling

Protein Dataset where task is classification into enzyme vs. non-enzyme.
Dataset: 450 enzymes and 450 non-enzymes.
Architecture (ReLU activation):
- GCN with $L = 3$ layers and $d = 25$ feature vectors in each layer;
- $Ext : Z_j = \sum_{k=1}^{n} \exp(-\pi \| y_k - z_j \|)$ with $m = 120$ and $z_j$ random.
- Fully connected NN with dense 3-layers and 120 internal units.
Readout Mapping Approach
Polynomial Expansion - Quadratics

Another interpretation of the moments for $d = 1$: coefficients of linear expansion

$$P(X) = \frac{1}{n} \sum_{k=1}^{n} (X - x_k)^n = X^n + \sum_{k=1}^{n} a_k X^{n-k}$$
Another interpretation of the moments for $d = 1$: coefficients of linear expansion

$$P(X) = \frac{1}{n} \sum_{k=1}^{n} (X - x_k)^n = X^n + \sum_{k=1}^{n} a_k X^{n-k}$$

For $d > 1$, consider the quadratic $d$-variate polynomial:

$$P(Z_1, \cdots, Z_d) = \prod_{k=1}^{n} \left( (Z_1 - x_k(1))^2 + \cdots + (Z_d - x_k(d))^2 \right)$$

$$= \sum_{p_1, \ldots, p_d = 0}^{2n} a_{p_1, \ldots, p_d} Z_1^{p_1} \cdots Z_d^{p_d}$$

Encoding complexity:

$$m = O \left( \binom{2n + d}{d} \right) \sim (2n)^d.$$

A more careful analysis of $P(Z_1, \ldots, Z_d)$ reveals a form:

$$P(Z_1, \ldots, Z_d) = t^n + Q_1(Z_1, \ldots, Z_d)t^{n-1} + \cdots + Q_{n-1}(Z_1, \ldots, Z_d)t + Q_n(Z_1, \ldots, Z_d)$$

where $t = Z_1^2 + \cdots + Z_d^2$ and each $Q_k(Z_1, \ldots, Z_d) \in \mathbb{R}_k[Z_1, \ldots, Z_d]$. Hence one needs to encode:

$$m = \binom{d + 1}{1} + \binom{d + 2}{2} + \cdots + \binom{d + n}{n} = \binom{d + n + 1}{n} - 1$$

number of coefficients.
One can do even better than that!
Consider the \( n \) linear forms \( \lambda_k(Z_1, \ldots, Z_d) = x_k(1)Z_1 + \cdots x_k(d)Z_d \).
Consider the polynomial in variable \( t \) with coefficients in \( \mathbb{R}[Z_1, \ldots, Z_d] \):

\[
P(t) = \prod_{k=1}^{n} (t - \lambda_k(Z_1, \ldots, Z_d)) = t^n - e_1(Z_1, \ldots, Z_d) t^{n-1} + \cdots (-1)^n e_n(Z_1, \ldots, Z_d)
\]

The elementary symmetric polynomials \((e_1, \ldots, e_n)\) are in 1-1 correspondence (Newton-Girard theorem) with the moments:

\[
\mu_p = \sum_{k=1}^{n} \lambda_k^p(Z_1, \ldots, Z_d) , \quad 1 \leq p \leq n
\]
Each $\mu_p$ is a homogeneous polynomial of degree $p$ in $d$ variables. Hence to encode each of them one needs $\binom{d+p-1}{p}$ coefficients. Hence the total embedding dimension is

$$m = \binom{d}{1} + \binom{d+1}{2} + \cdots + \binom{d+n-1}{n} = \binom{d+n}{n} - 1$$
Each $\mu_p$ is a homogeneous polynomial of degree $p$ in $d$ variables. Hence to encode each of them one needs $\binom{d + p - 1}{p}$ coefficients. Hence the total embedding dimension is

$$m = \binom{d}{1} + \binom{d + 1}{2} + \cdots + \binom{d + n - 1}{n} = \binom{d + n}{n} - 1$$

For $d = 1$, $m = n$ which is optimal.

For $d = 2$, $m = \frac{n^2 + 3n}{2}$. Is this optimal?
Idea: Consider the case $d = 2$. Then each $x_1, \cdots, x_n \in \mathbb{R}^2$ can be replaced by $n$ complex numbers $z_1, \cdots, z_n \in \mathbb{C}$, $z_k = x_k(1) + ix_k(2)$.

Consider the complex polynomial:

$$Q(z) = \prod_{k=1}^{n} (z - z_k) = z^n + \sum_{k=1}^{n} \sigma_k z^{n-k}$$

which requires $n$ complex numbers, or $2n$ real numbers.
Idea: Consider the case $d = 2$. Then each $x_1, \cdots, x_n \in \mathbb{R}^2$ can be replaced by $n$ complex numbers $z_1, \cdots, z_n \in \mathbb{C}$, $z_k = x_k(1) + ix_k(2)$.
Consider the complex polynomial:

$$Q(z) = \prod_{k=1}^{n} (z - z_k) = z^n + \sum_{k=1}^{n} \sigma_k z^{n-k}$$

which requires $n$ complex numbers, or $2n$ real numbers.

Open problem: Can this construction be extended to $d \geq 3$?
Remark: A drawback of polynomial (algebraic) embeddings: [Cahill’19] showed that polynomial embeddings of translation invariant spaces cannot be bi-Lipschitz.
Quadratic Optimization Problems

Approach

Consider two symmetric (and positive semidefinite) matrices $A, B \in \mathbb{R}^{n \times n}$. The quadratic assignment problem asks for the solution of

$$
\text{maximize} \quad \text{trace}(\Pi A \Pi^T B)
$$

subject to:

$$
\Pi \in S_n
$$

where *Input* stands for a given set input data, and $S_n$ denotes the symmetric group of permutation matrices.

**Idea:** Use a two-step procedure:

1. Perform a latent representation of the Input Data using a Graph Convolutive Network (or Graph Neural Network);
2. Solve the Linear Assignment Problem for an appropriate cost matrix to obtain an estimate of the optimal $\Pi$. 
Consider two $n \times n$ symmetric matrices $A$, $B$. In the alignment problem for quadratic forms one seeks an orthogonal matrix $U \in \text{O}(n)$ that minimizes

$$
\|UAU^T - B\|_F^2 := \text{trace}((UAU^T - B)^2) = \|A\|_F^2 + \|B\|_F^2 - 2\text{trace}(UAU^T B).
$$

The solution is well-known and depends on the eigendecomposition of matrices $A$, $B$: if $A = U_1 D_1 U_1^T$, $B = U_2 D_2 U_2^T$ then

$$
U_{opt} = U_2 U_1^T, \quad \|U_{opt} A U_{opt}^T - B\|_F^2 = \sum_{k=1}^n |\lambda_k - \mu_k|^2,
$$

where $D_1 = \text{diag}(\lambda_k)$ and $D_2 = \text{diag}(\mu_k)$ are diagonal matrices with eigenvalues ordered monotonically.
The challenging case is when $U$ is constrained to belong to the permutation group. In this case, the previous minimization problem

$$\min_{U \in S_n} \|UAU^T - B\|_F$$

turns into the QAP:

$$\max_{U \in S_n} \text{trace}(UAU^TB).$$

In the case $A, B$ are graph Laplacians (or adjacency matrices), an efficient solution to this optimization problem would solve the graph isomorphism problem, one of the remaining millennium problems: decide if two given graphs are the same modulo vertex labelling.
Prior work to discrete optimizations using deep learning

- Direct approach to discrete optimization: Pointer Networks (Ptr-Nets) utilize sequence-to-sequence Recurrent Neural Networks [Vinyals’15];
- Reinforcement learning and policy gradients: [Bello’16]
- Graph embedding and deep Q-learning: [Dai’17]
- QAP using graph deep learning: [Nowak’17] utilizes siamese graph neural networks that act on $A$ and $B$ independently to produce embeddings $E_1$ and $E_2$; then the product $E_1 E_2^T$ is transformed into a permutation matrix through soft-max and cross-entropy loss.

Shift Invariance Properties

Consider $A = A^T$ and $B = B^T$ (no positivity assumption).

**Lemma**

The QAP associated to $(A, B)$ has the same optimizer as the QAP associated to $(A - \lambda I, B - \mu I)$, where $\lambda, \mu \in \mathbb{R}$.

Indeed, the proof of this lemma is based on the following direct computation:

$$\text{trace}(\Pi(A - \lambda I)\Pi^T(B - \mu I)) = \text{trace}(\Pi A\Pi^T B) - \mu \text{trace}(A) - \lambda \text{trace}(B) + n\lambda \mu$$

A consequence of this lemma is that, without loss of generality, we can assume $A, B \geq 0$. In fact, we can shift the spectrum to vanish the smallest eigenvalues of $A, B$. 
The case of Rank One

Assume now \( A = aa^T \) and \( B = bb^T \) are non-negative rank one matrices. Then:

\[
\text{trace}(\Pi A \Pi^T B) = |b^T \Pi a|^2 = (\text{trace}(\Pi ab^T))^2 = \frac{1}{\text{trace}(AB)}(\text{trace}(\Pi AB))^2
\]

In this case we obtain the explicit solution to the QAP:

**Lemma**

Assume \( A = aa^T \) and \( B = bb^T \) are rank one. Then the QAP optimizer is the optimizer of one of the following two optimization problems:

\[
\begin{align*}
\text{maximize} & \quad \text{trace}(\Pi C) \\
\text{subject to:} & \quad \Pi \in S_n
\end{align*}
\]

or

\[
\begin{align*}
\text{minimize} & \quad \text{trace}(\Pi C) \\
\text{subject to:} & \quad \Pi \in S_n
\end{align*}
\]

where \( C = AB \).
Linear Assignment Problems

Given a cost matrix $C \in \mathbb{R}^{n \times n}$, the Linear Assignment Problem (LAP) is defined by:

$$\begin{align*}
\text{maximize} & \quad \text{trace}(\Pi C) \\
\text{subject to:} & \quad \Pi \in S_n
\end{align*}$$

Without loss of generality, max can be replace by min, for instance by solving LAP for $-C$. 

Radu Balan (UMD)
Linear Assignment Problems

Given a cost matrix $C \in \mathbb{R}^{n \times n}$, the Linear Assignment Problem (LAP) is defined by:

$$\begin{align*}
& \text{maximize} \quad \text{trace}(\Pi C) \\
& \text{subject to:} \\
& \quad \Pi \in S_n
\end{align*}$$

Without loss of generality, max can be replace by min, for instance by solving LAP for $-C$.

The key observation is that LAP can be solved efficiently by a linear program. Specifically, the convexification of LAP produces the same optimizer:

$$\begin{align*}
& \text{maximize} \quad \text{trace}(WC) \\
& \text{subject to:} \\
& \quad W_{i,j} \geq 0 \ , \ 1 \leq i, j \leq n \\
& \quad \sum_{i=1}^{n} W_{i,j} = 1 \ , \ 1 \leq j \leq n \\
& \quad \sum_{j=1}^{n} W_{i,j} = 1 \ , \ 1 \leq i \leq n
\end{align*}$$
Diagonal Matrices

Another case when we know the exact solution is when $A$ and $B$ are diagonal matrices. Say $A = \text{diag}(a)$ and $B = \text{diag}(b)$. Then

$$\text{trace}(\Pi A \Pi^T B) = \text{trace}(\text{diag}(\Pi a) \text{diag}(b)) = \text{trace}(\Pi ab^T) = \text{trace}(\Pi C)$$

where $C = ab^T$.

**Lemma**

If $A = \text{diag}(a)$ and $B = \text{diag}(b)$ then the solution of the QAP is given by the solution of the LAP

$$\text{maximize} \quad \text{trace}(\Pi C)$$

$$\text{subject to:}$$

$$\Pi \in S_n$$

where $C = ab^T$.
Graph Deep-Learning Based Approach: First convert the input data \((A, B)\) into a cost matrix \(C\), and then solve two LAPs, one associated to \(C\) the other associated to \(-C\). Finally choose the permutation that produces the larger objective function.
The conversion step \((A, B) \mapsto C\) is performed by a Graph Convolutional Network (GCN).
Kipf and Welling (2016) introduced a network structure that performs local processing according to a modified adjacency matrix:

Here \( \tilde{T} = I + T \), where \( T \) is an input adjacency matrix, or graph weight matrix. The \( L \)-layer GCN has parameters \((W_1, B_1, W_2, B_2, \cdots, W_L, B_L)\). As activation map \( \sigma \) we choose the ReLU (Rectified Linear Unit).
The Specific GCN Architecture

For the QAP associated to matrices \((A, B)\) we design a specific GCN architecture:

\[
X = \begin{bmatrix} A & 0 \\ B & 0 \end{bmatrix}, \quad \tilde{T} = \begin{bmatrix} I_n & 1 \\ \frac{1}{\|A\|_F \|B\|_F} BA & I_n \end{bmatrix}
\]

(2.1)

where the 0 matrices in \(X\) are designed to fit the appropriate size of \(W_1\). For \(\sigma\) we choose the ReLU (Rectified Linear Unit) function in each layer except for the last one; in the last layer we do not use any activation function (i.e., \(\sigma = \text{Identity}\)). The biases \(B_1, \cdots, B_L\) are chosen of the form \(B_k = 1 \cdot \beta_k^T\), i.e., each row \(\beta_k^T\) is repeated.
The following result applies to this network.

**Theorem**

Assume $A = aa^T$ and $B = bb^T$ are rank one with $a, b \geq 0$, and consider the GCN with $L$ layers and activation map ReLU as described above. Then for any nontrivial weights $W_1, \cdots, W_L$ and zero biases $B_1, \cdots, B_L = 0$ the network output $Y$ partitioned $Y = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}$ into two blocks of $n$ rows each, satisfies $Y_1Y_2^T = \gamma AB$, for some constant $\gamma \in \mathbb{R}$. In particular, the max-LAP and min-LAP applied to the latent representation matrix $C = Y_1Y_2^T$ are guaranteed to produce the optimal solution of the QAP.
Reference Algorithms

We compare the GCN based optimizer with two different algorithms.
1. The *AB Method* bypasses the GCN block. Thus $Y = X$ and the cost matrix inputted into the LAP solver is simply $C = AB$ (hence the name of the method). Similar to the GCN approach, the AB Method is exact on rank 1 inputs. But there is no adaptation of the cost matrix for other input matrices.
2. The *Iterative* algorithm is based on alternating max-LAP or min-LAP as follows:

$$\Pi_{k+1} \in \left\{ \begin{array}{c}
\arg\max_{\Pi \in S_n} \text{trace}(\Pi A \Pi_k^T B), \\
\arg\min_{\Pi \in S_n} \text{trace}(\Pi A \Pi_k^T B)
\end{array} \right\}$$

where $\Pi_0 = I$ (identity), and the choice of permutation at each $k$ is based on which permutation produces a larger $\text{trace}(\Pi A \Pi_k^T B)$. 
Comparison with Ground Truth
Results for $2 \leq n \leq 10$ and raw data normal distributed

Average relative difference w.r.t. maximum objective function:

Figure: Top left: ABMethod, Top right: Iterative algorithm, Bottom left: GCN with $L=2$ layers and bias, Bottom right: GCN with $L=3$ layers and bias
Comparison with Ground Truth
Results for $2 \leq n \leq 10$ and raw data uniform distributed

Average relative difference w.r.t. maximum objective function:

**Figure**: Top left: ABMethod, Top right: Iterative algorithm, Bottom left: GCN with $L=2$ layers and bias, Bottom right: GCN with $L=3$ layers and bias
Relative Comparison
Results for $n = 100$ and $n = 200$ with raw data normal distributed

Figure: Top row: Frequency of optimal algorithm for $n = 100$ (left), and $n = 200$ (right). Bottom row: Relative performance [%] to the best algorithm for $n = 100$ (left) and $n = 200$ (right)
Relative Comparison
Results for $n = 100$ and $n = 200$ with raw data normal distributed

Figure: Top row: Frequency of optimal algorithm for $n = 100$ (left), and $n = 200$ (right). Bottom row: Relative performance [%] to the best algorithm for $n = 100$ (left) and $n = 200$ (right).
Bibliography

Bibliography

Bibliography