Vignette of iGraphMatch R Package

—— Expected to be released in May 2018

Zihuan Qiao

11/29/2017

General Description of Graph Matching

Graph matching is an increasingly important problem which can be applied to a wide variety of fields including biology, neuroscience, pattern recognition and machine learning, to name a few. For example, as a fundamental tool in solving pattern recognition problem, graph matching is widely used in computer vision. In this problem, one seeks to find a correspondence between local features of the image, which are labeled as nodes of the graphs. Relational aspects between features are modeled by edges of the graphs in this context.

While packages such as `iGraph` and `GraphM` also have graph matching functionality our goal is to provide a single centralized repository for graph matching which attempts to confront many of the additional pathologies of graph matching. While the `iGraph` package provides versatile options on descriptive network analysis and graph visualization based on igraph objects in R, Python and C/C++, it doesn't focus on implementation of the most commonly used and cutting edge algorithms of graph matching. In contrast to `iGraph`, the `iGraphMatch` package is also more flexible in dealing with different type of graph objects, ranging from igraph objects to matrix objects. The `GraphM` [1] provides tools for approximately soloving large scale graph matching problems. It implements a variety of graph methods that were proposed between 1999 and 2009, including Umeyama algorithm, Linear programming approach, Rank algorithm, QCV (Quadratic convex relaxation) algorithm and PATH (A path following) algorithm in C/C++. Its corresponding R package version `RGraphM` hasn't been published yet.

What Is the Graph Matching Problem

The graph matching problem seeks to find an alignment between the vertices of two graphs with shuffled labels. To formulate the problem, given two adjacency matrices A and B corresponding to graphs $G1$ and $G2$ with n nodes, the target is to find the true permutation matrix $P = \text{argmin}_{P\in\Pi}\|A - PBP^T\|_F$, where Π denotes the set of all the $n \times n$ permutation matrices, $\| \bullet \|$ denotes the Froebenius norm. In a mathematical sense, the graph matching problem is a quadratic assignment problem. The goal of research in graph matching has been focused on developing more accurate or faster algorithms to approximately match two graphs since the problem is NP-hard.

Seeded graph matching has been extensively studied in the literature [2]. Suppose that we know the correspondence between a subset of vertices in both graphs. With some available seeds, the problem is to minimize $\|A - (I \oplus P)B(I \oplus P)^T\|_F$ over all m-by-m permutation matrices P, where $m := n - s$, and $\oplus$ is the direct sum, and I is the identity matrix.

What Can iGraphMatch Package Do

The `iGraphMatch` package aims to provide useful and convenient tools to users who are dealing with problems related to graph matching and working with either igraph objects or matrix objects. Among the capabilities of this package, you can do the following:

- Implement the FAQ algorithm and convex relaxed algorithm to match two given undirected, weighted or unweighted graphs.
- Incorporate prior information such as known correspondences and probabilistic estimates of the correspondences.
- Evaluate goodness of matching for each vertex.
- Initialize different start matrix for the graph matching iteration.

- Generate pairs of graph samples according to a specific graph model, e.g. Erdős-Rényi graph model, random dot product graph model and stochastic block model, etc.

Outline of This Vignette

This document introduces `iGraphMatch`'s basic set of tools by showing you how to apply them to some problems in the graph matching field:

- Graph matching with different initialization of start matrix.
- Find core vertices in the presence of junk vertices.
- Graph matching with adaptive seeds.

Background

Graph Mathcing Algorithms

The graph matching algorithms currently implemented in `iGraphMatch` are the FAQ algorithm [3] (`graph_match_FW`) and the convex relaxed algorithm (`graph_match_convex`) [4]. Seeded graph matching algorithm is an extension of FAQ algorithm which incorporates seeds [2].

Formally, for any two n-by-n adjacency matrices A and B corresponding to two graphs $G1$ and $G2$ with n nodes, the graph matching problem is to minimize $\|A - PBP^T\|_F^2$ over all n-by-n permutation matrices, and $\|\bullet\|_F$ denotes the Froebenius norm, which is to minimize the edge disagreements between $G1$ and $G2$. For any $P \in \Pi$, where Π is the set of all the permutation matrices, the original objective function of the graph matching problem can be expanded

$$\|A - PBP^T\|_F^2 = \|AP - PB\|_F^2 = \|A\|_F^2 + \|B\|_F^2 - 2\langle AP, PB\rangle.$$

The first equality holds due to unitarity of the permutation matrices. Optimizing any of these equivalent forms of the objective function over all the permutation matrices is a NP-hard quadratic assignment problem due to the combinatorial complexity of the constraints $P \in \Pi$.

Relaxation techniques can be applied to solve the optimization problem by replacing the constraints from all the permutation matrices to all the doubly stochastic matrices, which is the convex hull of Π. There are two ways to relax the original graph matching problem, which correspond to two equivalent forms of the graph matching objective function. When relaxation is applied to the middle term, we get the convex relaxed graph matching problem, which is to minimize $\|AD - DB\|_F^2$ over all the n-by-n doubly stochastic matrices. When relaxation is applied to the third term, the graph matching problem becomes maximizing $\mathrm{tr}BDAD^T$ over all the n-by-n doubly stochastic matrices, we call it the indefinite relaxed graph matching problem. The Hessian of $\mathrm{tr}BDAD^T$ is not necessarily positive definite, this is why it got the name. In order to clearly present the relationships between the original graph matching problem and these two different relaxation techniques, we list the descriptions and properties of them in the table below.

Graph Matching Problem	Objective Functions	Constraints	Optimization Guarantees
Original Form	$\|A - PBP^T\|_F^2$	$P \in \Pi$	NP-hard
Indefinite Relaxed Form	$\mathrm{tr}BDAD^T$	$D \in \mathcal{D}$	Local Convergence
Convexed Relaxed Form	$\|AD - DB\|_F^2$	$D \in \mathcal{D}$	Global Convergence

The FAQ algorithm [3] is an efficient approximate graph matching algorithm based on Frank-Wolfe methodology, which is a gradient acent approach. The Frank-Wolfe methodology is described in algorithm 1. Experimental studies yield that it's reasonable to set the number of iteration less than 25. In the `graph_match_FW` function, the default setting for `max_iter` parameter is 100. Previous studies have proved that if $n \geq 100$ and $G2$, which is an isomorphic copy of $G1$, has $V(G2)$ being a discrete-uniform random permutation of $V(G1)$,

then the probability of FAQ yields the correct alignment is nearly 1.

Algorithm 1: Frank-Wolfe Methodology

Initialization: specify one n-by-n doubly stochastic matrix D^0, a tolerance ϵ;
Set $i = 0$;
while $\|D^i - D^{i-1}\|_F^2 \geq \epsilon$ **do**
    $P^i = \text{argmax}_{P \in \mathcal{P}} \text{tr} \nabla f(D^i)^T P$;
    $D^{i+1} = \text{argmax}_{D \in \mathcal{D}} f(D)$ over line segment from D^i to P^i, where $\mathcal{D}$ is the set of all doubly stochastic matrices;
    i=i+1;
end
Project D^i to the nearest P by maximizing $\text{tr}(P^T D)$, we get $\hat{P}$;

Finally, let's look at the optimization guarantees for different relaxation methods. The convex relaxed problem can achieve the glocal minimum. But we can only employ the tools of continuous optimization to find the local maximum for the indefinite relaxed problem, and therefore initialization is significant for the problem. These global optima or local optima is then projected back to Π, yielding an approximate solution to the original optimization problem.

In the context of seeded graph matching [2], without loss of generality, assume the correct matching is I and suppose the first s nodes are seeds. Suppose A and B are adjacency matrices of graphs $G1$ and $G2$ with n nodes. To formulate the seeded graph matching algorithm, denote the number of nonseed $m := n - s$, and partition adjacency matrices A and B

$$A = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{21}^T \\ B_{21} & B_{22} \end{bmatrix}$$

where $A_{11}, B_{11} \in \{0,1\}^{s \times s}$ denotes the adjacency matrices corresponding to seeds, $A_{22}, B_{22} \in \{0,1\}^{m \times m}$ denotes the adjacency matrices corresponding to nonseeds, and $A_{21}, B_{21} \in \{0,1\}^{m \times s}$ corresponds to the nonseed to seed information. Then the seeded graph matching problem is to find $P = \text{argmax}_{P \in \Pi} \text{tr} A^T (I \oplus P) B (I \oplus P)^T$, where Π is the set of all permutation matrices. To apply the Frank-Wolfe methodology, relax the maximization of $\text{tr} A^T (I \oplus P) B (I \oplus P)^T$ over all permutation matrices to the maximization of the same objective function over all doubly stochastic matrices to form a convex optimization problem. Further simplification yields the objective function

$$f(D) = \text{tr} A_{11} B_{11} + 2\text{tr} D^T A_{21} B_{21}^T + \text{tr} A_{22} D B_{22} P^T$$

with gradient

$$\nabla(D) = 2A_{21}B_{21}^T + 2A_{22}DB_{22}.$$

where D denotes the doubly stochastic matrix. Then apply the Frank-Wolfe methodology with the specified objective function and gradient function, and optimize over all the m-by-m doubly stochastic matrices. Therefore, to understand the seeded graph matching, it is still based on the Frank-Wolfe methodology while using different objective function to incorporate seeds.

Suppose we are matching two graphs with cardinality n, the running times of both the FAQ algorithm and convex relaxed algorithm are $O(n^3)$ (per iteration). In `iGraphMatch` package, for both `graph_match_FW` (corresponds to the FAQ algorithm) and `graph_match_convex` (corresponds to convex relaxed algorithm) functions, the argument `seeds` incorporates seeds information.

Correlated Erdős-Rényi Random Graphs

Presently, we describe Correlated Erdős-Rényi random graphs model [2]. This model is an extensively used theoretical framework within which many theorems are proved. This model is frequently used in simulations as well.

Correlated Erdős-Rényi random graphs model is specified by parameters including a positive number n, a real number $p \in (0,1)$ and a real number $\rho \in [0,1]$. Suppose we have two graphs $G1$ and $G2$ whose common cardinality of vertex set is n. For each pair of vertices $\{v, v'\} \in \binom{V}{2}$, let $\mathbb{1}\{\{v, v'\} \in E(Gi)\}$ denote the indicator variable for the event $\{v, v'\} \in E(Gi)$, where $i = 1, 2$. Correlated Erdős-Rényi Random Graphs model assumes that the indicator variable $\mathbb{1}\{\{v, v'\} \in E(Gi)\}$ follows Bernoulli(p) distribution. We call p edge probability, because it is the probability of there existing an edge between $\{\{v, v'\} \in E(Gi)\}$. Note that the random indicator variables are all independent within a graph, but there is Pearson product-moment correlation coefficient ρ between each pair of vertices $\{v, v'\} \in \binom{V}{2}$ across graphs. If ρ is 0, then $G1$ and $G2$ are independent, and at the other extreme, if ρ is 1, then $G1$ and $G2$ are identical almost surely.

Random dot product graphs (RDPG) model is another frequently used random graphs model, which can be regarded as an extension of correlated Erdős-Rényi Random Graphs model. For the RDPG model, for each pair of vertices $\{v, v'\} \in \binom{V}{2}$, the edge probability p between them is specified by the dot product of two vectors in $\mathbb{R}^d$ drawn from distribution $\mathbb{F}^d$, representing latent positions. Given this, the probability of an edge occuring between $\{v, v'\}$ is determined by the Bernoulli trail with probability given by the dot product of two latent positions vectors.

In `iGraphMatch` package, `sample_correlated_gnp_pair` and `sample_correlated_gnp_pair_w_junk` are two useful functions to generate graph pairs from Correlated Erdős-Rényi Random Graphs model. `sample_correlated_rdpg` function is for sampling a pair of graphs from random dot product graphs model.

Initialization Methods For Soft Seeding

Soft Seeding Graph Matching

In the previous studies on seeded graph matching, authors often assume that there is prior knowledge about the vertex correspondence. This could be that part of the bijection between the two vertex sets is known. However, in many cases, knowledge on seeds is quite limited, e.g. there might be errors in the seeds or we may only know the range of possible matches to a seed. If we still incorporate these information as hard seeds and keep them fixed during graph matching, incorrect information can yield bad matching results. As a result, we propose another way to make use of such information by incorporating them into the initialization of D^0 in the first step of the FAQ algorithm. We call such kind of information soft seeds.

Soft seeds can improve graph matching performance when the prior information is uncertaint since soft seeds are not fixed. Suppose n_{ss} is the number of soft seeds, n_{ns} is the number of nonseeds. Soft seeding algorithm is based on the Frank-Wolfe methodology which is described in algorithm 1. Without loss of generality, assume the first n_{ss} of the non-hard-seeds are soft seeds. In step 1, soft seeding initializes the start matrix at $D^0 = \begin{bmatrix} D^0_{n_{ss}} & \mathbf{0}^T \\ \mathbf{0} & D^0_{n_{ns}} \end{bmatrix} P$, where $D_{n_{ss}}$ is n_{ss}-by-n_{ss} doubly stochastic matrix, which denotes matrix of m soft seeds from graph A to graph B. $D_{n_{ss}}$ can be an identity matrix or any doubly stochatic matrix to represent many-to-many soft seeds information. $\mathbf{0}$ is the n_{ns}-by-n_{ss} zero matrix. $D_{n_{ns}}$, which is n_{ns}-by-n_{ns} doubly stochastic matrix, denotes matrix without soft seeds. P, which is a n-by-n permutation matrix multiplied on the right side of the block matrix, denotes permutation to columns.

Initialization Methods

Soft seeds are implemented by choosing a specific initialization for the Frank-Wolfe iterations, which is to initialize matrix D^0. Notice that D^0 is composed of two doubly stochastic matrices, $D^0_{n_{ss}}$ and $D^0_{n_{ns}}$ should be initialized seperately. $D^0_{n_{ss}}$ is specified by incorporating the information in soft seeds. If the soft seeds are one-to-one, which means each soft seed is matched to a specific vertex in the other graph, then $D^0_{n_{ss}} \in \{0,1\}^{n_{ss} \times n_{ss}}$. Or if the soft seeds are many-to-many, that is we only know probabilistic estimates of the correspondences, $D^0_{n_{ss}}$ should still be a doubly stochastic matrix. While $D^0_{n_{ss}}$ can be uniquely specified by the information in soft seeds, there are multiple ways to initialize $D^0_{n_{ns}}$. Presently, we discuss different methods of initialization of $D^0_{n_{ns}}$ which affects the performance of soft seeding. Here we will discuss three

initialization methods: barycenter start, random doubly stochastic start and convex start. A good choice of the start matrix contributes to finding the global optimum permutation matrix and to higher speed of reaching the result.

The most straightforward way is to initialize at the barycenter, indicating that each nonseed vertex is equally likely to be matched to other nonseed vertices. Suppose n_{ns} is the number of non seeds and n_{ss} is the number of soft seeds. The formula for barycenter start is:

$$D_{n_{ns}} = D_{bari} = \frac{1}{n_{ns} - n_{ss}} \mathbb{1}\mathbb{1}^T_{(n_{ns}-n_{ss})\times(n_{ns}-n_{ss})}.$$

Another approach to initialize the start matrix is to use a random doubly stochastic matrix. The algorithm to generate a random doubly stochastic matrix incorporates the sinkhorn iterative renormalization approach. [5] The basic idea is to normalize rows and columns of the matrix iteratively. Algorithm 2 describes how the algorithm works. Note that the expectation of the random doubly stochastic matrix is barycenter.

Algorithm 2: Initialization of a random doubly stochastic matrix

Input : Dimension of the matrix n ($:= n_{ns} - n_{ss}$), distribution $\mathbb{F}$, maximum number of iteration num.

Generate $D_{ij} \sim \mathbb{F}, i, j \in \{1, ..., n\}$;
Initialize $m = 0$;
while $m \leq num$ **do**
  $D_{i,\bullet} = D_{i,\bullet} / \sum_{i=1}^{n} D_{i,j}, i \in \{1, ..., n\}$;
  $D_{\bullet,j} = D_{\bullet,j} / \sum_{i=1}^{n} D_{i,j}, j \in \{1, ..., n\}$;
end

Output: A n-by-n doubly stochastic matrix D.

Here we illustrate the convex initialization method in the general setting where we have both hard seeds and soft seeds. Suppose hard seeds provide the correct bijection between two vertex sets. To generate the convex start matrix, we first use the convex relaxed graph matching algorithm, regarding all the soft seeds as hard seeds. The correspondence of soft seeds won't change during this initial matching process. As a result, the convex initialization method can only be applied to one-to-one soft seeds, the convex initialization method for many-to-many soft seeds doesn't make sense. We have $n_s + n_{ss}$ seeds, where n_s denotes the number of hard seeds and n_{ss} denotes the number of soft seeds. Denote the number of nonseeds as $n_{ns} := n - n_s - n_{ss}$. Then we follow the seeded graph matching algorithm to find the doubly stochatic matrix corresponding to nonseeded vertices: $\hat{D_{n_{ns}}} = \text{argmin}_{D_{n_{ns}} \in \mathcal{D}'} \|A - (I_{n_s} \oplus (I_{n_{ss}} \oplus D_{n_{ns}})P)B(I_{n_s} \oplus (I_{n_{ss}} \oplus D_{n_{ns}})P)^T\|_F^2$, where $\mathcal{D}'$ denotes the set of all n_{ns}-by-n_{ns} doubly stochastic matrices. I_{n_s}, which is a n_s-by-n_s identity matrix, denotes matrix of hard seeds. $I_{n_{ss}}$, which is a n_{ss}-by-n_{ss} identity matrix, denotes matrix of one-to-one soft seeds. $D_{n_{ns}}$, which is a n_{ns}-by-n_{ns} matrix, denotes matrix without seeds. P, which is a $(n - n_s)$-by-$(n - n_s)$ permutation matrix multiplied on the right side of the block matrix, denotes permutation to columns. The convex start matrix is defined as $D_{convex} := (D_{n_{ss}} \oplus \hat{D_{n_{ns}}})P$ which is a $(n - n_s)$-by-$(n - n_s)$ matrix.

Simulations

Soft seeding algorithm can be easily implemented in R using the existing functions in `iGraphMatch` package. Basically, the main function used is `graph_match_FW` with specification of the start method and seeds which correspond to the hard seeds. For example, now we try to compare random doubly stochastic start, bari start and convex start which are discussed above and do simulations on the correlated Erdős-Rényi graphs model. To make the result easy to be shown, we sample a graph pair with a small cardinality 10 from the correlated Erdős-Rényi graphs model with the edge probability equal to 0.5 and the correlation between two graphs to be 0.5. Set the first three pairs of vertices to be hard seeds.

```
library(iGraphMatch)
library(dplyr)
library(purrr)
```

```
set.seed(8)
cgnp_pair <- sample_correlated_gnp_pair(n = 10, corr = .5, p = .5)
g1 <- cgnp_pair$graph1
g2 <- cgnp_pair$graph2

(seeds <- 1:10 <= 3)
```

```
##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Then just randomly pick two pairs of vertices as soft seeds. Note that soft seeds may not be correct, but soft seeds may improve during the matching procedure. Let's first pick two bad soft seeds {4,4} and {8,6} which is a combination of good soft seed and bad soft seed:

```
bad_soft_seeds <- rbind(c(4,4),c(8,6))
```

Initialization of different start matrix is very convenient in R with `iGraphPackage`. Function `init_start` returns a n_{ns}-by-n_{ns} matrix where n_{ns} denotes the number of nonseeds vertices.

```
nns <- 7
ns <- 3
options(digits = 3)
(start_bari <- init_start(start = "bari", nns = nns, ns = ns, soft_seeds = bad_soft_seeds))
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]    1  0.0    0  0.0  0.0  0.0  0.0
## [2,]    0  0.2    0  0.2  0.2  0.2  0.2
## [3,]    0  0.2    0  0.2  0.2  0.2  0.2
## [4,]    0  0.2    0  0.2  0.2  0.2  0.2
## [5,]    0  0.0    1  0.0  0.0  0.0  0.0
## [6,]    0  0.2    0  0.2  0.2  0.2  0.2
## [7,]    0  0.2    0  0.2  0.2  0.2  0.2
```

```
set.seed(123)
(start_rds <- init_start(start = "rds", nns = nns, ns = ns, soft_seeds = bad_soft_seeds))
```

```
##      [,1]   [,2] [,3]   [,4]   [,5]   [,6]  [,7]
## [1,]    1 0.0000    0 0.0000 0.0000 0.0000 0.000
## [2,]    0 0.0822    0 0.0168 0.3238 0.3594 0.218
## [3,]    0 0.2680    0 0.2313 0.1824 0.1168 0.202
## [4,]    0 0.1378    0 0.3875 0.2702 0.0198 0.185
## [5,]    0 0.0000    1 0.0000 0.0000 0.0000 0.000
## [6,]    0 0.2467    0 0.1985 0.1893 0.1279 0.238
## [7,]    0 0.2653    0 0.1660 0.0344 0.3761 0.158
```

```
(start_convex <- init_start(start = "convex", nns = nns, ns = ns, soft_seeds = bad_soft_seeds,
                            A = g1, B = g2, seeds = seeds))
```

```
## 7 x 7 sparse Matrix of class "dgCMatrix"
##
## [1,] 1 . . . . . .
## [2,] . 1 . . . . .
## [3,] . . . 1 . . .
## [4,] . . . . . . 1
## [5,] . . 1 . . . .
## [6,] . . . . 1 . .
## [7,] . . . . . 1 .
```

Then implement seeded graph matching using the FAQ algorithm in R by using `graph_match_FW` function.

To incorporate soft seeds, we specify the `start` argument in `graph_match_FW` function accordingly. Note that in this example, we assume the true permutation to be the identity matrix.

```
set.seed(123)
match_bari <- graph_match_FW(g1, g2, seeds, start = start_bari)
match_rds <- graph_match_FW(g1, g2, seeds, start = start_rds)
match_convex <- graph_match_FW(g1, g2, seeds, start = start_convex)
err_bari_bad <- mean(match_bari$corr$corr_A[!seeds] != match_bari$corr$corr_B[!seeds])
err_rds_bad <- mean(match_rds$corr$corr_A[!seeds] != match_rds$corr$corr_B[!seeds])
err_convex_bad <- mean(match_convex$corr$corr_A[!seeds] != match_convex$corr$corr_B[!seeds])
```

Since in this example both of the soft seeds are incorrect, intuitively not incorporating such incorrect information might yield better matching results. This motivates us to perform a comparative experiment on whether or not to incorporate such incorrect soft seeds. The R code for performing graph matching with only the hard seeds is similar to soft seeding except for different specification of the `start` argument.

```
set.seed(123)
match_nss_bari <- graph_match_FW(g1, g2, seeds, start = "bari")
match_nss_rds <- graph_match_FW(g1, g2, seeds, start = "rds")
match_nss_convex <- graph_match_FW(g1, g2, seeds, start = "convex")
```

For completeness, we also include an example of good soft seeds, say {4,4} and {8,8}. Then we follow the same procedure as in the bad soft seeds case, and yield the matching results for good soft seeds when matching the same graphs. Since the codes corresponding soft seeding implementation are the same, we'll skip the codes and only show the results comparing the three cases.

```
good_soft_seeds <- rbind(c(4,4),c(8,8))
```

Table 2: Matching Errors With Various Initialization Methods

	bari	rds	convex
good soft seeds	0.714	0.286	0.000
bad soft seeds	0.857	0.286	0.857
non soft seeds	1.000	0.571	0.286

Table 2 presents the matching error of nonseeded vertex sets with various initialization methods. Compared with using a mixture of good soft seeds and bad soft seeds and matching without soft seeds, incporating good soft seeds decrease or maintain the same matching error for all the initialization methods. Note that for the convex initialization method, we can successfully uncover the true correspondence of two graphs by using good soft seeds. Using partial good soft seeds can also improve the matching performances except for the convex case. In general, random doubly stochastic initialization yields the most stable matching result while the convex intialization method using good soft seeds achieves the highest matching accuracy among all the methods.

We also illuminate the difference between various initialization methods by presenting the experiment results on larger scale graphs under more parameter settings and with more monte carlo replicates. The experiment is relatively time consuming in case of larger scale graphs, we recommend to run larger simulations on server, or consider using statistical computing methods, e.g. divide and conquer algorithm \cite{D&C} to enhance the speed of experiments. Here we just show the result figure and skip the R code.

Figure 1 shows the average performance for 300 graph pairs sampled from Erdős-Rényi graph model with the settings: $p \in \{0.1, 0.2, 0.5\}$, $\rho = 0.5$, $n_v = 250$ and $n_s = 10$. Wrong soft seeds are randomly sampled from the nonseed vertices. We observe that with a moderate number of incorrect soft seeds, convex start outpreforms the other two start methods. Converting from a random doubly stochastic start to a convex start can reduce the matching error by 0.1 to 0.3. As a result, when implementing a soft seeding graph matching, if we know

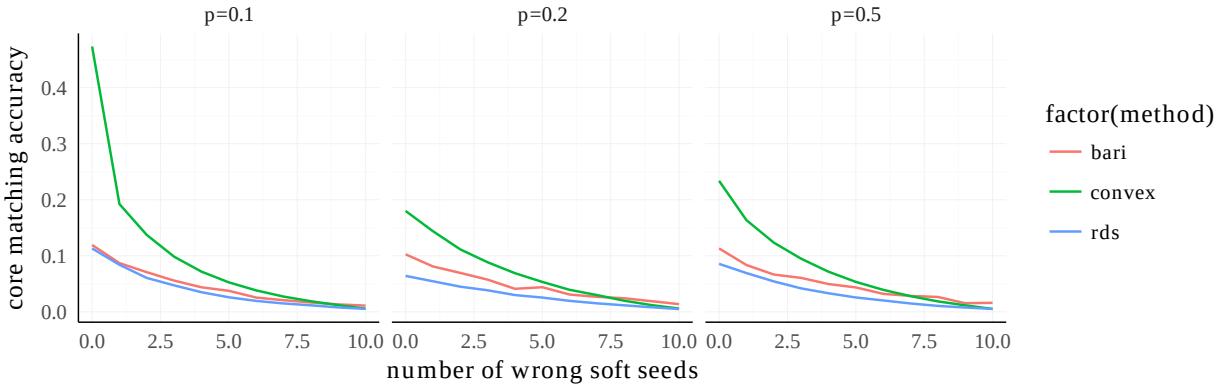

Figure 1: Core matching accuracy versus the number of wrong soft seeds, for various initialization methods for soft seeding and edge probability. Fix $\rho = 0.5$, $n_v = 250$.

the proportion of incorrect soft seeds is not too high (approximately $\leq 70\%$), it's recommendated to choose a convex start.

Identification of Core Vertices

Graph Matching with Junk Vertices Setting

The previous discussions are all based on the setting that there is always a corresponding vertex in $G2$ for each vertex in $G1$. however, this is not always the case. Social networks offer a compelling example for this, where matching across different social platforms (or within a single time varying social network) requires the understanding that not all users will be participants in both networks.

Junk vertices refer to the vertices that don't have true alignments in the other graph. This could be modeled by correlated heterogeneous Erdős-Rényi random graphs [7].

Definition For R and Λ symmetric, hollow matrices in $[0,1]^{n\times n}$, wesay A, B are R-correlated heterogeneous Erdős-Rényi(Λ) random graphs (abbreviated $CorrER(\Lambda, R)$) if:

i. A and B are marginally $ER(\Lambda)$; i.e., for all u, $v \in [n]$, $u < v$, $A_{uv} \overset{\text{iid}}{\sim} Bern(\Lambda_{uv})$ and $B_{uv} \overset{\text{iid}}{\sim} Bern(\Lambda_{uv})$, with $A_{uv} = A_{vu}$ and $B_{uv} = B_{vu}$.

ii. For all u,v,w,$r \in [n]$, $u < v$, $w < r$, it holds that A_{uv} and B_{wr} are independent unless $u = w$ and $v = r$, in which case the correlation between A_{uv} and B_{uv} is $R_{u,v} \geq 0$.

At one extreme, if $R = [0]_n$ then the graphs are independent $ER(\Lambda)$, and at the other extreme, if $R = J_n$ then A and B are isomorphic almost surely. This model is a generalization of the homogeneous correlated ER model discussed earlier, and similarly allows for the addition of "junk" vertices—those without a probabilistic match across graphs—by setting $R = R_k \oplus [0]_{n-k}$ for some $k \leq n$.

$(A, B) \sim CorrER(\Lambda, R)$ are matchable, if $\text{argmin}_{P\in\Pi(n)}\|AP - PB\|_F = I_n$, where I_n denotes the identity matrix. Our goals are to uncover the alignment between core vertices, i.e those where $R_{u,v} > 0$ for some $u, v \in [n]$, and to detect which vertices are core versus junk vertices.

Different Measures for Goodness of Matching

Having a measure for goodness of matching is important. The measure can be used in finding core vertices in the setting with junk vertices. Since we can rank all the vertices in the order of goodness of matching based on the measure, it's also useful for choosing the best matched vertices. The best matched core vertices can then be used as additional seeds in the next iteration of graph matching, and we call this process adaptive seeding which is introduced in the next section. Here we list three different measures for goodness of matching, row permutation statistics, row difference and row correlation.

Row permutation statstics [7] is based on a graph matching variant of the permutation test. In terms of a vertex v, test the hypotheses

$$H_0^{(v)} : \forall P \in \mathcal{P}, u \neq v, corr(A_{vu}, (PBP^T)_{vu}) = 0,$$

$$H_A^{(v)} : \exists P \in \mathcal{P}, u \neq v, corr(A_{vu}, (PBP^T)_{vu}) > 0$$

To test, we will make use of the following relationship between edge-wise correlation and the number of induced error between A and P^*BP^{*T}. Namely, if v is a core vertex (or v is correctly matched), then the number of errors induced by P^* across the neighborhoods of v in A and B (i.e., $\|(AP^* - P^*B)_{v,\bullet}\|_1$) should be significantly smaller than the number of errors induced by a randomly chosen permutation P (i.e., $\|(AP - PB)_{v,\bullet}\|_1$). With this in mind, we define $\Delta_v(P) = \|(AP - PB)_{v,\bullet}\|_1$ and let $\mathbb{E}_P$ and Var_P denote the conditional expection and variance of these quantities with respect to uniform sampling of P over all permutation matrices. Inspired by the permutation-test, we define the row permutation statistic as:

$$T_p(v, P^*) := \frac{\Delta_v(P^*) - \mathbb{E}_P \Delta_v(P)}{\sqrt{Var_P \Delta_v(P)}}$$

Intuitively, the larger $T_p(v)$, the more likely v is to be a core vertex (or the more likely we find the true alignment to v).

Row difference is defined as the L-1 norm of A and P^*BP^{*T}, which is

$$T_d(v, P^*) := \|A_{v\bullet} - (P^*BP^{*T})_{v\bullet}\|_1$$

Intuitively, correctly matched vertex v (or core vertex v) should induce smaller $T_d(v, P^*)$.

Row correlation is a statistics for the correlation between A and P^*BP^{*T} which is defined as

$$T_c(v, P^*) := 1 - corr(A_{v\bullet}, (P^*BP^{*T})_{v\bullet})$$

If v is correctly matched, then the correlation between the neighborhoods of v in A and B should be smaller. Thus, larger $T_c(v)$ value indicates higher chance of vertex v being correctly matched (or more likely v is to be a core vertex).

Algorithm of Finding Core Vertices

We next develop an approach to identify core vertices correctly. The main tool that we utilize is a graph matching variant of permutation test. We will use the test statistic $T(\bullet)$ to rank vertices based on the likelihood they are core vertices. Suppose $A, B \in \{0,1\}^{(n_c+n_j)\times(n_c+n_j)}$ are the adjacency matrices corresponding to graphs $G1$ and $G2$, where n_c denotes the number of core vertices and n_j denotes the number of junk vertices. Denote the available seeded vertices as S. Algorithm of finding core vertices consists of the following steps. First, use the available seeded vertices S to match A and B yielding the optimal permutation P^*. Based on the matching result, then plug in P^* to compute the value of $T(v, P^*)$ for each vertex v. Finally rank all the vertices via the decreasing value of $T(v, P^*)$ with increasing likelihood of being core vertices, that is the first n_j vertices are identified as junk vertices with decreasing likelihood.

Simulations

Now implement algorithm of finding core vertices in R with `iGraphMatch` package. First we sample a pair of graphs with 50 vertices from the correlated Erdős-Rényi graph model and let 5 out of the total vertices to be junk vertices using function `sample_correlated_gnp_pair_w_junk`. We also assume the first 10 vertices to be seeds. Then apply the FAQ algorithm to match two graphs.

```
set.seed(5)
cgnp_pair <- sample_correlated_gnp_pair_w_junk(n = 50, corr = .5, p = .5, ncore = 45)
g1 <- cgnp_pair$graph1
g2 <- cgnp_pair$graph2

seeds <- 1 : 50 <= 10
core <- 1 : 50 <= 45
junk <- !core
non_seeds <- !seeds

match <- graph_match_FW(g1, g2, seeds, start = "rds")
```

Then implement the algorithm for finding core vertices, we use row permutation statistic as our measure in this example. The main steps of the algorithm involves calculating values of the row permutation statistic for each nonseed vertex and rank them in the order of increasing value of the statistic. These steps can be implemented by using the `best_matches` function. The `best_matches` function has the functionality of finding best matched vertices and identifying core vertices. Since we want to rank all the non-seed vertices in terms of their row permutation statistics, set argument `x` to be `non_seeds`, which denotes the vertices we are interested in. Also set argument `num` to be the number of non-seeds, which denotes the number of top ranked vertices we want to get. By returning the result in `A_best`, we get the indices corresponding to vertices in $G1$ in the order of decreasing likelihood of being core vertices.

```
r <- best_matches(g1, g2, measure = "row_perm_stat", num = sum(non_seeds),
                  x = non_seeds, match_corr = match$corr)$A_best
r
```

```
##  [1] 18 27 13 32 37 30 31 14 43 17 24 39 50 12 42 40 34 44 26 48 11 20 29
## [24] 36 35 19 28 21 46 22 38 16 25 33 15 41 45 23 47 49
```

Table 3: Summarization Table for Core Identification Precisions

	k1	k5	k10	k25	k35
core identification precision	1	1	1	0.944	0.914

Table 4: Summarization Table for Junk Identification Precisions

	k1	k2	k3	k4	k5
junk identification precision	0.5	0.373	0.44	0.5	0

Since there are 5 junk vertices, the last 5 vertices in the output are identified as junk vertices. Note that seeded vertices are not ranked. In order to evaluate the performance of core identification algorithm in this example, we calculate the precision of classification for each vertex and summarize the precisions in table 3 and table 4. Table 3 shows the identification precision for $k = 1, 5, 10, 25, 35$, where k denotes the rank of identified core vertices. Higher rank indicates the vertex has higher probability to be a core vertex. From table 3, the core identification algorithm achieves pretty high precision for identifying the core vertices. In contrast, the identification precision for junk vertices drops quickly as we averaging over more high rank junk

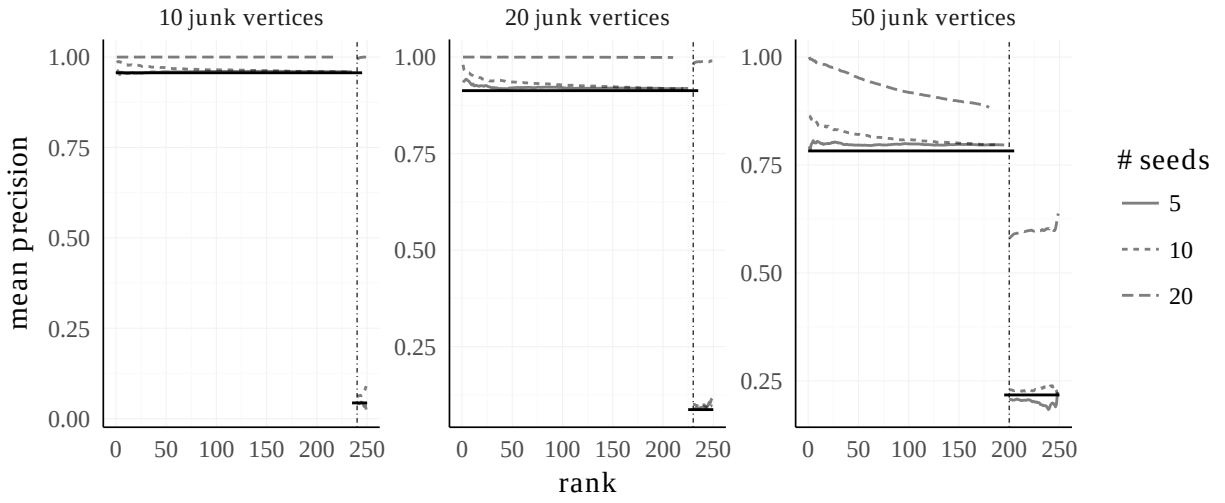

Figure 2: Mean precision for classifying vertices as core or junk by using the row permutation statistic. Graphs are generated from $\text{CorrER}(0.2, 0.5J_{n_c} \oplus 0_{n_j})$. The dashed vertical lines seperate core vertices from junk vertices. The horizontal lines represent the performance of a random classifier.

vertices.

Figure 2 shows the simulation results for two random graphs sampled from $\text{CorrER}(0.2, 0.5J_{n_c} \oplus 0_{n_j})$ with $n_j \in \{20, 50\}$ and $n_s \in \{5, 10, 20\}$ [7]. The figure averages the results of 1000 monte carlo replicates by plotting the mean precision at each rank with lower ranks indicating core vertices and higher ranks near n indicating junk vertices. When there are a small number of seeded vertices, the core identification algorithm doesn't outperform the random algorithm much, which is indicated by the horizontal lines. But when we have 20 seeds, the core identification algorithm achieves substantially higher precision than the random algorithm, especially for junk vertices. As expected, when there are smaller number of junk vertices, the algorithm is able to have better performance.

Graph Matching with Adaptive Seeds

In this part, we will show you how to conduct FW graph matching method with adding adaptive seeds iteratively. The idea is motivated by the fact that in many realistic applications, we only know a small number of seeds while the number of vertices to be matched is large. Since there are costs and difficulties in acquiring seeds, the following algorithm will provide a feasible approach to acquire additional seeds and use them in graph matching to achieve higher matching accuracy.

The inputs are the adjacency matrices of two graphs A and B. Both graphs are composed of n_c matchable core vertices and n_j junk vertices, vertices that don't have a bijection in the other graph, and some available seeds S. The detailed algorithm is given in algorithm 3.

Simulations

Now implement the algorithm in R with `iGraphMatch` package on the correlated stochastic block model. First use function `sample_correlated_sbm_pair_w_junk` to sample a pair of graphs from the correlated stochastic block model with 50 nodes in each graph and the correlation between two graphs is 0.5. We set the number of blocks in each graph to be 2 of sizes 15 and 35 and specify the edge probabilities between vertices within the smaller block, the bigger block and across blocks to be 0.3, 0.5 and 0.7 respectively. In

Algorithm 3: Core matching with added seeds

Input : Adjacency matrices $A, B \in \{0,1\}^{(n_c+n_j)\times(n_c+n_j)}$; available seeded vertices S; number of best matches to choose num

1 Match the vertices from A and B using available seeds S, yielding the optimal permutation P^{0^*};

2 Use P^{0^*} to compute $T(v, P^{0^*})$ for each vertex v in graph A;

3 Rank the vertices from graph A via decreasing value of $T(v, P^{0^*})$;

4 Choose num top ranking vertices from graph A and their matched vertices from graph B, denoted as S_{add}, and add the node pairs to seeds;

5 Match the non-seed vertices from graph A and graph B with seeds, $S = S \oplus S_{add}$. Calculate the updated core matching error;

Output: Matching accuracies of core vertices before and after adding additional seeds

this example, we consider a core-junk setting by setting the first 10 and 30 vertices in each block to be core vertices, while the rest being junk vertices. Finally let the first 5 core vertices in block be known seeds.

```
set.seed(6)
pm <- cbind( c(.3, .5), c(.5, .7) )
sbm_pair <- sample_correlated_sbm_pair_w_junk(n = 50, pref.matrix = pm, rho = 0.5,
                                              block.sizes = c(15,35),
                                              core.block.sizes = c(10,30))
g1 <- sbm_pair$graph1
g2 <- sbm_pair$graph2

seeds <- 1 : 50 <= 5
seeds[16:20] <- TRUE
```

Perform FW graph matching with the first 10 nodes to be seeds by using `graph_match_FW` function.

```
match <- graph_match_FW(g1, g2, seeds = seeds, start = "convex")
err <- mean(match$corr$corr_A[!seeds] != match$corr$corr_B[!seeds])
```

Analyze goodness of matching by using the row permutation statistics measure and select top 3 best matched pairs of vertices as adaptive seeds. Step 3 and 4 can be implemented by using the `best_matches` function as the following:

```
seeds_adp <- best_matches(A = g1, B = g2, measure = row_perm_stat, num = 3, x = !seeds,
                          match_corr = match$corr)
seeds_adp
```

```
##   A_best B_best
## 1     29     14
## 2     28     12
## 3     41     50
```

By using row permutation statistics, all of the three seeds selected are correct. Then we combine the adaptive seeds with the original seeds we have and redo the FW graph matching with the updated seeds:

```
seeds <- rbind(as.matrix(check_seeds(seeds, nv = 50)$seeds), as.matrix(seeds_adp))
match_adp <- graph_match_FW(g1, g2, seeds=seeds, start = "convex")
seeds <- 1 : 50 <= 10
err_adp <- mean(match_adp$corr$corr_A[!seeds] != match_adp$corr$corr_B[!seeds])
```

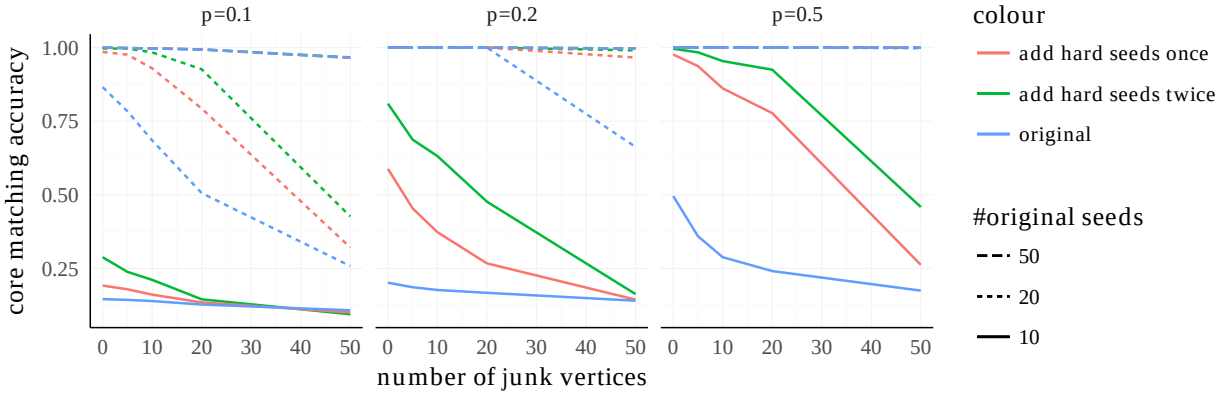

Figure 3: Matching accuracy when adding seeds adaptively across two iterations as a function of number of seeds, number of junk vertices and edge probability. Fix $\rho = 0.5$, n=250. 7 adaptive seeds are added in each iteration.

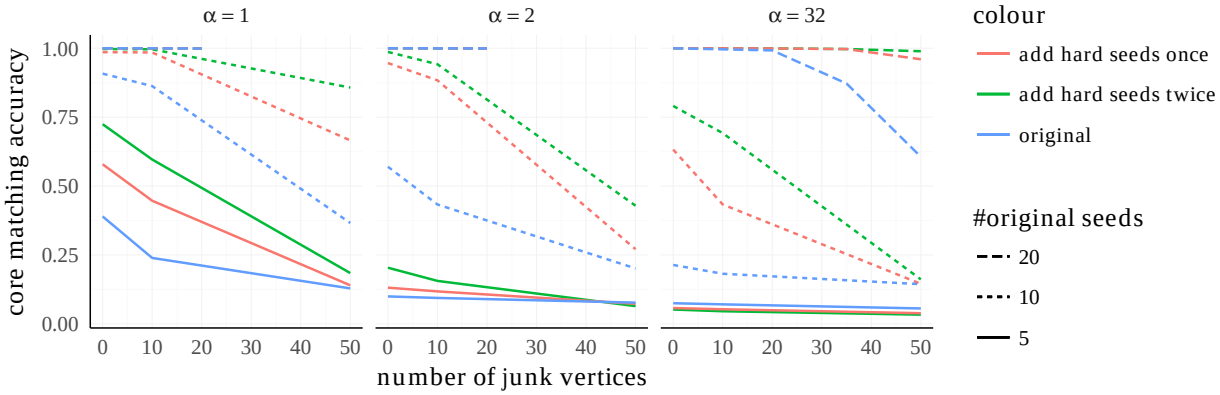

Figure 4: Matching accuracy when adding seeds adaptively across two iterations as a function of number of seeds, number of junk vertices and edge probability. Fix $\rho = 0.5$, n=250. 10 adaptive seeds are added in each iteration.

Table 5: Matching Errors for Adaptive Seeding

err_orig	err_adp
0.8	0.775

Table 5 shows the matching errors with only the original seeds and after adding 3 seeds adaptively, the matching error is reduced from 0.8 to 0.775 in one iteration. Again we also present the result figures from simulations on larger graphs and with more monte carlo replicates to illustrate the performance of adaptive seeds. Figure 3, 4 and figure 5 show the average results of 300 monte carlo replicates.

In the Erdős-Rényi regime, we consider matching two graphs with 250 nodes, for $p \in \{0.1, 0.2, 0.5\}$. Set the correlation between two graphs to be 0.5 and consider matching two graphs $G1$ and $G2$ using the SGM algorithm [2] run with $n_s \in \{10, 20, 50\}$, $n_j \in \{0, 5, 10, 20, 50\}$. From figure 3, we see algorithm 3 improves the performance of graph matching under each of $n_s \in \{10, 20, 50\}, n_j \in \{0, 5, 10, 20, 50\}$, except for one case when edge probability is small ($p = 0.1$) and number of junk vertices is big ($n_j \geq 20$). The figure also clearly shows that adding adaptive seeds iteratively contributes to improving core matching performance.

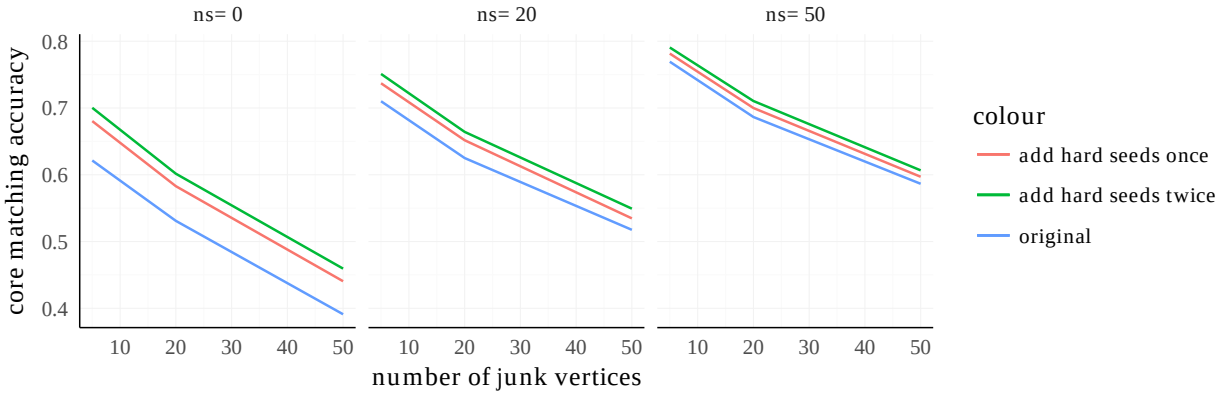

Figure 5: Matching accuracy when adding seeds adaptively across two iterations as a function of number of seeds and number of junk vertices. Fix $\rho = 0.5$ and $n_{as} = 7$. 20 adaptive seeds are added in each iteration.

To analyze the effect of vertex heterogeneity including degree heterogeneity and mixed membership community structure, we also simulated correlated graphs from the random dot product graph distributions. For this case, we have $A, B \sim CorrER(XX^T, R)$ where $X = [X_1, X_2, ..., X_n]^T \in \mathcal{R}^{n \times d}$ is the matrix of latent positions and $X_1, ..., X_n$ are independently and identically distributed for some distribution F satisfying $\mathbb{P}[X_i^T X_j \in [0, 1]] = 1$ for all i, j. Note that $A_{ij}, B_{ij} \sim Bern(X_i^T, X_j)$. In this exapmle, we take F to be a 2-dimensional marginal of a Dirichlet distribution on three dimensions with scale parameter α and mean vector $(1/3, 1/3, 1/3)$. Note that when α is large, the distribution will approximate the homogeneous Erdős-Rényi case with $p = \frac{2}{9}$.

Figure 4 shows the result of matching accuracy as a function of number of original seeds, number of junk vertices and various scale for the random dot product graphs model. Each line in the figure is the average of 300 monte carlo replicates for each of $\alpha \in \{1, 2, 32\}$, $n_s \in \{5, 10, 20\}$ and $n_j \in \{0, 10, 40, 50\}$. Note that in the experiment, we skip the cases when $\alpha \in \{1, 2\}$ and the number of original hard seeds is 20, because the graph matching accuracy is already very high in the first iteration when no seeds are added adaptively. From the figure, we can see that the algorithm is more effective in improving core matching performance with smaller scale, less junk vertices and more original seeds. Notice that in the setting $\alpha = 1, ns = 20, nj \leq 10$, and the setting $\alpha = 32, n_s = 20, n_j \leq 50$ adding hard seeds twice can improve the core matching performance to almost perfect.

In order to evaluate the performance the algorithm 3 with real data, we also did experiments with real Twitter data. We took the most active users from April and May 2014 to be the nodes of the graphs. The graphs are then weighted by means of taking log of the times a user mentioned another user during the given month. The empirical Pearson correlation between the entries in the two weighted adjacency matrices are approximately 0.91. We finally have the same group of users of size 431 in each graph by keeping the largest common connected ones.

To mimick the core-junk setting in the context of Twitter data, we sample three disjoint sets of users from 431 users, namely C, J_{April} and J_{May} of sizes n_c, n_j and n_j respectively. Then combine C and J_{April} to get an induced subgraph of Twitter users from April. Similarly, Combine C and J_{May} to get the second graph, the induced subgraph of Twitter users from May. Since three vertex sets are disjoint, vertices J_{April} and J_{May} don't have an alignment in the other graph. Hence J_{April} and J_{May} are the junk vertices, and vertices C are the core vertices.

Figure 5 shows the average matching accuracy across 300 monte carlo replicates of graphs sampled with $n = 250$, $n_s \in \{0, 20, 50\}$ and $n_j \in \{5, 20, 50\}$. Adding hard seeds effectively increase matching accuracy in each iteration. Although the original matching accuracy is lower with smaller number of original seeds, matching accuracy increases quicker when we add adaptive seeds.

##Discussion The primary goal of this package is to facilitate research related to graph matching by providing a variety of useful tools in several aspects of graph matching problem, including graph matching algorithm, quality measure, random graph models etc. In `iGraphMatch` package we implement two graph matching algorithms, the FAQ algorithm and the convex relaxed algorithm. Both algorithms are applicable to the setting with junk vertice which is a more general case. Moreover, the package is capable of handling weighted graphs and graphs with different orders (by adding padding [7]). For future works, we target to implement more graph matching algorithms aiming at enhancing the computational efficiency, which is especially significant to matching two large scale graphs.

References

[1] M.Zaslavskiy and F.Bach and J-P.Vert. *A path following algorithm for the graph matching problem.* IEEE Transactions on Pattern Analysis and Machine Intelligence, DOI: 10.1109/TPAMI.2008.245, (2009).

[2] Vince Lyzinski, Donniell E. Fishkind, Carey E. Priebe. *Seeded Graph Matching for Correlated Erdős-Rényi Graphs.* Journal of Machine Learning Research, 15, pp 3513-3540 (2014).

[3] J. T. Vogelstein, J. M. Conroy, L. J. Podrazik, S. G. Kratzer, E. T. Harley, D. E. Fishkind, R. J. Vogelstein, and C. E. Priebe. *Large (brain) graph matching via fast approximate quadratic programming.* arXiv preprint arXiv:1112.5507 (2011).

[4] V. Lyzinski, D. E. Fishkind, M. Fiori, J. T. Vogelstein, C. E. Priebe and G. Sapiro. *Graph Matching: Relax at Your Own Risk.* IEEE Transactions on Pattern Analysis and Machine Intelligence, DOI: 10.1109/TPAMI.2015.2424894, (2015).

[5] Philip A. Knight *The Sinkhorn-Knopp Algorithm: Convergence and Applications* SIAM Journal on Matrix Analysis and Applications, 30(1), 261–275 (2008).

[6] V. Lyzinski, D. Sussman, D. E. Fishkind, H. Pao, L. Chen, J. Vogelstein, Y. Park and C. E. Priebe. *Spectral Clustering for Divide-and-Conquer Graph Matching.* Parallel Computing, doi:10.1016/j.parco.2015.03.004 (2015).

[7] V. Lyzinski, D. L. Sussman. *Graph matching the matchable nodes when some nodes are unmatchable.* arXiv 1705.02294 (2017).