# Bachelor Thesis about Partially Labeled Classification in Graphs

## Using Network Propagation Methods and Applications in Protein Function Annotaions

Yiftach Kolb

Instructor

Prof. Dr. Martin Vingron

November 5, 2020

**Abstract**

This is an abstract . . .

## Contents

# 1 Introduction

This Thesis deals with applying propagation methods for protein function prediction. Our goal is to propose a prediction algorithm. Its input is a bidirectional graph and a labeling of a subset of the vertices —that's the partial classification which we wish to extend. The algorithm outputs for each unlabeled vertex a suggested suitable label, and the label must be one of the existing labels. We are going to require just one label for each vertex. That is for the sake of simplification, although in reality when we deal with protein-protein interaction networks (PPI), each vertex (protein) can have multiple labels (functions or annotations in the Jargon). We call this problem the partially labeled classification problem (PLC) as it was named by [15].

Our input network is a PPI network, which is a bidirectional graph where vertices represent a protein type in the cells of a certain organism and an edge represent some sort of an observed 'interaction' between them that is unlikely to be observed that frequently just by happenstance.

However some edges may be interactions that don't actually occur in the organism of research, yet they are observed in the test tube due to the method that produces the PPI data.

Suppose that some of the proteins are well researched and the life science guys have provided us with further information about their biological functions. This information is represented in the network in the form of labels associated with some of the vertices. But this information is incomplete and many vertices are unlabeled. We want to extend this labeling to all the nodes, by guessing for each unlabeled vertex which of labels could fit it best.

Estimating the accuracy of the prediction is another issue of concern. In this paper we use a PPI network where all the proteins are annotated with labels. We remove the labels of some of them, try to predict their labels, and verify. In the end we know how many labels we got right and how many wrong. This gives us an estimation of the total accuracy.

We would also like to eastimate or score the likelihood for each specific prediction (perhaps as part of a future paper or in the master thesis. We would need for example to come up with a statistical way to quantify the posterior probability that a vertex has a certain label.

We use propagation methods in attempt to find a prediction algorithm for the PLC. Specifically we use random walk with restart (RWR) approach. Essentially what we do is picking an unlabeled vertex and starting randomly walking from it with restarts at a certain probability back to the starting vertex. In this walk we sometimes visit labeled vertices. We predict that the label that we most frequently encounter in this walk is the most likely label of the starting vertex.

There are 2 papers that we know of which deal with the PLC. Deng et al. [3] Specifically deals with protein function prediction in a PPI network. They use a method which is based on "Markov random fields" to estimate for each function category separately the likelihood that a protein in the network has that function or not. In their experiments of predicting the functions of annotated yeast proteins they reached specificity and sensitivities of 45-64%.

The other paper is Szummer and Jaakkola [15] and it is purely mathematical and from it we took the name partially labeled classification (PLC) for this problem. They use time scaled random walk without restart to calculate the posterior probability that an unlabeled vertex has a certain label. This method may work for the purpose of protein function prediction but in the way it is tested in the article it is hard to compare it to a PPI network scenario. For example they tested the algorithm on a network that fixed connectivity of $k = 5$ (5 neighbors for each vertex) and they labeled just 2 initial starting vertices. In short the testing scenario doesn't resemble a partially annotated PPI network type of situation but the method and the definition of the problem do fit.

We dedicate sections 2 and 3 of this Thesis to explain the underlying mathematical theories of stochastic matrices and random walk in graphs. We think that this theoretical introduction is required in order to develop an understanding of how RWR actually works, the relationship between graphs and stochastic matrices and spectral theory, what is the diffusion kernel, and for which type of graphs we can use these methods.

In the following 2 sections (4 and 5) We will deal with the subjects of community structure and spectral clustering in graphs. There is allot of overlap between the PLC problem and community structure. In the later problem there is no initial labeling and we just use the graph topology to find a sensible way of dividing it into tightly clustered groups, for example by using the modularity method as proposed by Newman [10]. With the PLC problem, we are given additional constraints. The labels which we are given dictate how many groups there are. We cannot switch labels for the labeled vertices even if it looks from a community structure perspective that 'these two don't belong together'. However we still assume the experimentally substantiated 'dogma' that proteins of the same function tend to cluster together in the PPI network.

In section 6 we are going to propose an algorithm using propagation methods for the PLC problem and show some initial experiments with it using a subnetwork of the yeast interactome.

Finally in section 7 we are going to wrap things up with some nice words and say good bye.

## 2 Linear algebra primer

**Definition 2.1.** A **Transition Matrix**, sometimes also called a **stochastic matrix**, is a real valued non negative square ($n^2$) matrix $A$ such that each of its columns sums up to one:

$$A \geq 0,\ \forall j \sum_i A_{i,j} = 1$$

$A$ acts from the left as a linear mapping: $A : v \to A \cdot v$. In the following script we treat matrices by default as acting from the left ($Av$).

A transition is **positive**, designated by $A > 0$ if all its entries are positive.

A transition is **primitive** if for some $k >$ $A^k$ is positive. The same property is called **regular** in some other sources.

A transition is **irreducible** if for every entry $A_{i,j}$ there is some $k$ such that $A^k_{i,j} > 0$.

Equivalently it can be shown that A matrix $A$ is irreducible by if and only if it is NOT similar by permutations matrices to a block upper triangular matrix which means $\nexists P : A = P \begin{pmatrix} B & C \\ 0 & D \end{pmatrix} P^{-1}$

Where $P$ is a permutation matrix and $B, C$ are square matrices of size greater than 0.

**Definition 2.2.** A **State** is a non-negative vector $v \in R^n$ s.t $\sum_i v_i = 1$.

*Remark* 1. If $v$ is a state and $A$ is a transition as defined in 2.1, then $Av$ is also a state because:

$$\sum_i (Av)_i = \sum_j v_j (\sum_k A_{j,k}) = \sum_j v_j \cdot 1 = 1$$

Also its easy to confirm by multiplying with $e_i$, that if $Av$ is a state for every state $v$ each column $Ae_i$ must sum to 1. Therefore this is an equivalent definition for a transition.

If $A$ is a transition and $x, y$ are two states such that $x \leq Ay$ then $x = y$.

**Definition 2.3.** Given $A \in \mathbb{C}^{n\times n}$ We let $|A| \in \mathbb{C}^{n\times n}$ be the resulting matrix from applying $|\cdot|$ element wise. Given a vector $v \in \mathbb{C}^n$ we let $|v| \in \mathbb{C}^n$ the corresponding non-negative vector.

We also let $A > 0, v > 0$ mean that it holds coordinate wise.

Here is a very useful lemma for non-negative matrices which we will need later:

**Lemma 2.1.** *Let* $0 \leq A \leq B \in \mathbb{R}^{n\times n}$ *and let* $0 < v \in \mathbb{R}^n$*. If* $Av = Bv$ *then* $A = B$*.*

*Proof.* Trivial. □

*Remark* 2. If $u \in \mathbb{C}^n$ is on the unit circle and $T$ a transition then $|u|$ is a state, meaning $\||u|\|_1 = 1$, so $T|u|$ is also a state so $\|T|u|\|_1 = 1$.

We have (component-wise) $|Tu| \leq T|u|$. If $T > 0$ and $u$ has negative or non-real entries, then this inequality must be strict and then $\|Tu\|_1 < \|T|u|\|_1 = 1$.

**Lemma 2.2.** *If $T$ is a transition, then there is a state $u$ such that $Tu = u$.*

*Proof.* Because the columns of $A$ all sum to 1, the columns of $A - I$ all sum to 0. Therefore $(1, 1, \ldots, 1)$ is an (left) eigenvector of the rows with eigenvalue 1. Therefore there is also some real (right) column eigenvector with eigenvalue 1. (Also it follows from the Brouer fixed point theorem because $T$ maps the $l_1$ sphere to itself).

Let $u \in R^n$ be such vector: $Au = u$. Let $u = u^+ + u^-$ such that $u^- = \min(u_i, 0)$ and the other one defined similarly for the non-negative components.

Because $A$ is non-negative $A(u^+) \geq (Au)^+ \geq 0$, and $A(u^-) \leq (Au)^- \leq 0$.

From $A$ being non-negative and $(Au)^+ + (Au)^- = Au = u = u^+ + u^-$ And also $(Au)^+ = (u)^+$, so we must have $Au^+ \geq (Au)^+ = u^+$ (component wise). But if we had a strict inequality we would get: $\|A(u^+/\|u^+\|_1)\|_1 > 1$ which is a contradiction to $A$ being a transition matrix.

Then $Au^+ = u^+, Au^- = u^-$ and one of them must be non-zero. It follows that $A$ has a non-negative eigenvector with eigenvalue 1 (one of $u^+, -u^-$ which is non-zero). If we $l_1$-normalize that eigenvector it becomes a state. □

**Lemma 2.3.** *If a transition $A > 0$ (or primitive) , then it has exactly one eigenvector $v$ with eigenvalue 1 and in addition it can be chosen so that $v > 0$ Any for any other eigenvalue $\lambda$ it holds that $|\lambda| < 1$.*

*If $A$ is just irreducible then then again $v > 0$ and is unique but there may be additional eigenvalues on the unit circle.*

*Proof.* Let $A > 0$ be a transition. We already know that there exists at least one such eigenvector. Let $u, v$ s.t $Au = u, Av = v$. We can assume these are real vectors because $A$ has only real entries. Therefore we can choose $u, v$ to be states as we have already proven above.

Then let $w = u - v$. So $Aw = A(u - v) = u - v = w$. And $\sum_i w_i = 1 - 1 = 0$ by choice of $u, v$.

Like before $Aw^+ = w^+, Aw^- = w^-$ and because $w \neq 0$ but sums up to 0, both $w^+, -w^- > 0$. Because $w^-$ is non zero exactly on entries where $w^+$ is zero and vice versa, each of them must have at least one 0 entry (and one none zero). But because $A$ is strictly positive and $w^+$ is non-negative, $Aw^+$ must have ONLY positive entries, contradicting $Aw^+ = w^+$. It follows then that $u - v = 0$ is the only possibility, hence the eigenvector with 1 as eigenvalue is unique.

Suppose there is $Aw = \lambda w$ where $|\lambda| = 1$. Choose $w$ so it is $l_1$-normalized. Then $|w| = |Aw| \leq A \cdot |w|$ If $w$ has any negative or complex coordinantes, then $|Aw| < A|w|$ and therefore $1 = \||Aw|\|_1 < \|A|w|\|_1 = 1$, a contradiction. Therefore there cannot be any other eigenvalues on the unit circle.

Extending this for primitive matrix is easy because for some sufficiently big $k$ $A^k > 0$.

To prove the uniqueness of the 1-eigenvector for the irreducible case, we have $(\forall k \in \mathbb{N}) A^k w^+ = w^+$ and from that with some more work left undone it follows that $w^+ > 0$ or $w^+ = 0$. □

□

*Remark* 3. The lemmas and theorems in this section are phrased in term of transitions. They hold true in the more general case of positive/non-negative linear transformations and one just replaces 1 with the **spectral radius** $\rho = \rho(A)$.

In general a non-negative linear transformation has a **spectral radius** $\rho = \rho(A)$ which is the absolute value of its greatest eigenvalue. In the case of positive maps there is a unique single eigenvector with $\rho$ as the unique greatest eigenvalue and so forth .... When we deal with a transition map, lemma 2.3 guaranties it has a spectral value of $\rho(A) = 1$.

**Theorem 2.1.** *Let $T$ be a positive (or primitive) transition. Then*

*1. 1 is the greatest eigenvalue of $T$ and it has one unique eigenvector which is also positive, so there exists a unique stationary state.*

*2. All the other eigenvalues have absolute value strictly less than 1.*

*3. For every state $u$, $T^k u$ converges to the stationary state $\pi$. In particular the columns of $T^k$ converge to $\pi$.*

*Proof.* 1 and 2. We already know.

3. There is a Jordan decomposition $T = PJP^{-1}$, such that $J$ is a Jordan matrix, $J_{1,1} = 1$ and the rest of the main diagonal $|J_{i,i}| < 1$. So now the convergence is clear $J^k \to (e_1|0\dots|0)$. For the matrix $P$ it must hold that $P = (P_1|\dots|P_n)$ and $P_1$ is the column eigenvector of $T$ corresponding to 1 which we are free to $l_1$ normalize and the first row of $P^{-1}$ is the left eigenvector corresponding to 1 and so force ....

Some more work or literature check should confirm that $T^k \to (v|v\dots|v) = V$. Then one can verify $Vu = v$ for any state $u$. □

**Theorem 2.2.** *Let $T$ be an irreducible positive (or primitive) transition. Then:*

*1. Then 1 is the greatest eigenvalue of $T$ and it has one unique eigenvector which is also positive, so there exists a unique stationary state.*

*2. If there are other other eigenvalues on the unit circle then their algebaic multiplicity is equal their geometric multiplicity.*

*3. For every state $u$, the* ***Cesaro sums*** *$\frac{1}{n}\sum_{k=1}^{n} T^k u$ converge to the stationary state $\pi$.*

*Proof.* 1 We already know.

2. There is a Jordan decomposition $T = PJP^{-1}$. such that $J$ is a Jordan matrix, $j_{1,1} = 1$ and the rest of the main diagonal $|J_{i,i}| \leq 1$.

If we had a Jordan block of size greater than 1 for an eigenvalue $\lambda$, Then on the superdiagonal of $J^k$ we would have $k\lambda$. If $|\lambda| = 1$ then $J^k$ and hence $T^k$ would be unbounded, but that is impossible since $T^k$ is a transition. If follows that all eigenvalues on the unit circle must be semi simple (alebgaic multiplicity equals geometric).

3. For the convergence, it follows from calculations on the Jordan blocks, which I omit. See Meyer [7] or Serre [13] for rigorous proof. □

What differs irreducible non-primitive matrices from primitive is that they are periodical on their eigenvectors with complex eigenvalues on the unit cycle. There is, in fact a wonderful theorem from Wielandt which characterizes these Matrices:

**Theorem 2.3** (Wielandt (1950))**.** *Let $A, B \in \mathbb{C}^{n\times n}$ such that $A \geq 0$ is irreducible and $|B| \leq A$ (component-wise). Then $\rho(B) \leq \rho(A)$. If $\rho(B) = \rho(A)$ then $|B| = A$, $B$ has an eigenvalue of the form $\mu = \rho(A)e^{i\phi}$ and:*

$$B = e^{i\phi} DAD^{-1}$$

*where $D$ has the form:*

$$D = \begin{pmatrix} e^{\theta_1} & & & \\ & e^{\theta_2} & & \\ & & \ddots & \\ & & & e^{\theta_2} \end{pmatrix} \tag{1}$$

*And conversely any $B$ of the form 1 has $\rho(B) = \rho(A)$, $|B| = A$ and $\mu$ is an eigenvalue of $B$ which corresponds to the eigenvalue $\rho(A) = |\mu|$ the greatest eigenvalue of $A$.*

*Proof.* To see a rigorous proof I suggest Meyer [7].

The keys for proving this theorem are: First WLG assume $A$ is a transition. This is possible because we may replace $A$ with $AW^{-1}$, and $B$ with $BW^{-1}$, where $W$ is the diagonal matrix that has the column-sums of $A$. Since $A$ is irreducible it cannot have a column or a row that is all 0 so this diagonal is positive $W$ is indeed invertible and later we can cancel out the $W$'s and return to the general case.

Let $v$ be the $\mu$-eigenvector $Bv = \mu v$, $\|\mu| = 1$ and choose it so that $\|v\|_1 = 1$.

Then

$$|v| = |\mu v| = |Bv| \leq |B||v| \leq A|v| \tag{2}$$

Since $A$ is a transition by remark 1 $A|v| = |v|$. Since $A$ is irreducible and $A|v| = |v|$ we must have by 2.2 that $|v| > 0$, and then by lemma 2.1 $A = |B|$ so that proves the first part.

Now let $w = v/|v|$ (component-wise division) and let

$$D = \operatorname{diag}(w) = \begin{pmatrix} e^{\theta_1} & & & \\ & e^{\theta_2} & & \\ & & \ddots & \\ & & & e^{\theta_2} \end{pmatrix}$$

Then $v = D|v|$, $|v| = D^{-1}v$ and we have:

$$\begin{aligned} A|v| = |v| = D^{-1}v = \\ = D^{-1}\mu^{-1}Bv = \mu^{-1}D^{-1}BD|v| \end{aligned} \tag{3}$$

We know already that $|v| > 0$. If $C := \mu^{-1}D^{-1}BD$ contains any negative or complex entries, then 3 cannot hold. It follows that

$$A = C = \mu^{-1}D^{-1}BD$$

This proves the harder direction of the claim, the other direction is easy □.

□

The amazing consequence from 2.3:

**Theorem 2.4** (Corollarly)**.** *If $A$ is irreducible transition with $h$ eigenvalues on the unit circle then its eigenvalues are exactly the $h$th unit roots $\lambda_k = e^{2\pi ik/h}, k = 0 \dots n-1$ and $A$ is similar to $\lambda A$ by a diagonal matrix for any such eigenvalue.*

*Proof.* Use theorem 2.3 with $B = A$. If $|\lambda| = 1$ is an eigenvalue then $A = \lambda DAD^{-1}$. Since similarity doesn't change the eigenvalues, $A$ and $\lambda A$ must have the same eigenvalues with the same multiplicity. Since 1 is a simple eigenvalue of $A$ and hence of $\lambda A$, and its corresponding eigenvalue $\lambda$ is simple in $\lambda A$ and therefore in $A$ as well.

The matrices $\lambda_k A$, $k = 0 \dots h$ are all similar, with $\lambda_0 = 1$ and $|\lambda_k| = 1, k = 0\dot{h} - 1$ all simple eigenvalues. The only way for this to hold is if $\{\lambda_k\}_0^{h-1}$ form a multiplicative group of order $h$ on the unit circle, in other words, the eigenvalues are exactly all the $h$th unit roots. □

*Remark* 4. If $A$ is irreducible transition with exactly $h$ eigenvalues on the unit circle then $h$ is called the **period** of $A$. $A \geq 0$ is primitive if and only if it is irreducible and aperiodic ($h = 1$).

If $\omega = e^{2\pi i/h}$ then 2.4 shows that $A = \omega DAD^{-1}$. So if $\lambda$ is any eigenvalue not necessarily on the unit circle, then $\omega\lambda$ is also an eigenvalue with the same multiplicity and rotation with $\omega$ is an automorphism on the eigenvalues.

We may choose $D$ so that $D[1,1] = 1$. If we reindex the dimension we can make $D$ look like

$$D = [D_0|\omega D_1|\dots\omega^{h-1}D_{h-1}]$$

So Indexes corresponding to the same phase of the period appear sequentially.

Then use the identify $\forall k D^h A^k = A^k D^h$ and the fact that $A$ is irreducible to show that $D^h = I$. Then use the identity $\omega DA = AD$ to show that in the new indexing $A$ has the following periodical block structure (0 on the main diagonal):

$$A = \begin{pmatrix} 0 & M_1 & 0 & \dots & 0 \\ 0 & 0 & M_2 & 0\dots & 0 \\ & & \ddots & \ddots & \\ 0 & \dots & 0 & 0 & M_{h-1} \\ M_h & 0 & \dots & 0 & 0 \end{pmatrix}$$

Now we will just present the Perron-Frobenius theorems. The main parts that are important to our work have appeared in the previous theorems.

**Theorem 2.5** (Perron-Frobenius [7])**.** *Let $0 < A \in \mathbb{R}^{n\times n}$ with spectral radius $\rho := \rho(A)$, then the following are all true:*

- $\rho > 0$
- *$\rho$ is a simple root of the characteristic polynomial of $A$, in other words its algebraic multiplicity is* 1.
- $(\exists v > 0) Av = \rho v$
- *If $Au = \lambda u$ and $\|u\| = \rho$ then $\lambda = \rho$ namely, $\rho$ is the unique eigenvalue on the spectral circle.*
- *(Collatz-Wielandt Formula) $\rho = \max_{x\in\Gamma} \min_{i:x_i\neq 0}[Ax]_i/x_i$ with $\Gamma = \{x|x \geq 0, x \neq 0\}$*

**Theorem 2.6** (Perron-Frobenius for irreducible matrices [7])**.** *Let $0 \leq A \in \mathbb{R}^{n\times n}$ be irreducible with spectral radius $\rho := \rho(A)$, then the following are all true:*

- $\rho > 0$
- *$\rho$ is a simple root of the characteristic polynomial of $A$, in other words its algebraic multiplicity is* 1*.*
- $(\exists v > 0) Av = \rho v$
- *There are no additional non-negative unit eigenvectors of $A$ other than $v$.*
- *(Collatz-Wielandt Formula)* $\rho = \max_{x \in \Gamma} \min_{i:x_i \neq 0} [Ax]_i / x_i$ *with* $\Gamma = \{x | x \geq 0, x \neq 0\}$

## 3 More on matrices, graphs and stochastics

A directed non-weighted graph $G$ can be uniquely represented by its **adjacency matrix**, $A_{i,j} := 1$ if and only if there is a directed edge from $i$ to $j$ (if we want to use it for transitioning on columns as done above). It's possible to assign different edge weights rather than only 1 and 0. If the graph is undirected each edge would be counted in both directions and the matrix is symmetric. Relabeling the vertices of a graph yields an adjacency matrix that is similar by permutations ($PAP^{-1}$, where $P$ is a permutation matrix) and vice versa.

To turn $A$ into a transition, normalize each column by dividing it with its out going rank, so let $D_{i,i} = \text{out rank}(i)$, $T := AD^{-1}$ is the transition matrix of this graph (because right-multiplying by $D$ normalizes each column by its rank). If the graph was stochastic to begin with then the adjacency matrix as we defined it is already column-normalized.

**Definition 3.1.** A graph is $G$ **strongly connected** if there is a directed path form any edge to any edge. Equivalently $G$ is strongly connected if and only if its adjacency matrix is irreducible.

We say that the **Period of a vertex** $v \in V(G)$ is the greatest common divisor of all closed paths on $v$. If the periods of all vertices are equal (spoiler: in the case that $G$ is strongly connected they are), we call it the **Period of the graph** $G$.

*Remark* 5. If $G$ is strongly connected then the periods of all vertices are indeed equal and its easy to prove. The corresponding adjacency matrix $A$ is irreducible so it too has a period $h$ as defined in 4 and it is equal to the graph period (can be shown using 2.4 and 4).

So if the graph $G$ is strongly connected and has period 1 then the adjacency matrix is **aperiodic** and hence primitive, and vice versa.

From all the above we have seen that a Markov process can be represented in two equivalent ways —as a transition matrix and as the corresponding weighted directed graph.

If the graph $G$ is strongly connected and aperiodic, its corresponding adjacency matrix is primitive. We know from 2.5 that there is a unique stationary distribution $p$ and that the Markov converges to $p$ no matter from which distribution it starts. We may calculate $p$ using the **power method** which is efficient because it can be done in a matrix-free method. We don't need to know the matrix itself just the dot product of it with a state vector.

If the graph $G$ is strongly connected, then 2.6 assures us the existence and uniqueness of a stationary distribution $p$. But if $G$ is not aperiodic, the corresponding adjacency matrix is not primitive. We cannot use the efficient power method to calculate $p$. Also the process itself doesn't stabilize on $p$. It is periodic and cycles between the $h$ eigenvectors on the unit circle.

We are therefore interested to find how to convert an imprimitive matrix (= irreducible but not primitive) to a primitive matrix or equivalently to turn a strongly connected but periodic graph into an aperiodic graph.

**Lemma 3.1.** *Let $A \geq 0$ be irreducible. Then $(A+I)^{n-1} > 0$, and therefore $A+I$ is primitive.*

*Proof.* Notice that the $I$ represents self loops and it absorbs all the lower powers so if $(A^k)_{i,j} > 0$ for some $k < n$ then so is $(A+I)^{n-11}$. Let $G := G_A$ be the corresponding graph to $A$. Then $G$ is strongly connected. For every $i \neq j$ there is a directed **shortest path** $\sigma$ in $G$ from $i$ to $j$. Its length must be $|\sigma| < n$. Otherwise $\sigma$ would have to contain a cycle and not be of minimal length. This shows that we have for every $i \neq j$ some $k$ sucht that $A^k_{i,j} > 0$ and therefore $(A+I)^{n-1} > 0$. □

The properties of irreducibility, primitiveness and positivity only depend on the sign $(-, 0, +)$ of the entries and not on their size. So we use the following definition to test matrices for these properties:

**Definition 3.2.** Let $A \in \mathbb{R}^{n \times n}$, then its binary form is the matrix $\beta(A) := \operatorname{sgn}(A)$ where sgn is applied element-wise.

The following trivial lemmas would help as to construct primitive transitions:

**Lemma 3.2.** *$A$ is positive/primitive/irreducibly if and only if $\beta(A)$ is.*

*Let $0 \leq \beta(A) \leq \beta(B)$. Then If $A$ is positive/primitive/irreducibly so is $B$.*

*Let $0 < \alpha < 1$. If $T, S$ are transitions the so is $W = (1-\alpha)T + \alpha S$. If one of $S, T$ is positive/primitive/irreducible so is $W$*

Let $G$ be any weighted graph and $A$ its adjacency transition matrix. Some vertices may be unreachable from other vertices and there might not exist a single and unique stationary distribution. A random walk on this graph is generally dependent on the initial starting distribution $p_0$ and has the form $p_{k+1} = Ap_k = \cdots = A^k p_0$, where $p_k$ is the distribution after $k$ steps.

However if we allow the possibility of 'random restart' from any state, this graph becomes totally connected it is guarantied to have a unique stationary distribution to which any random walk converges regardless of the initial state.

When we talk about **random walk with restart (RWR)** we set a restart state $q$ and a restart parameter $\alpha \in [0, 1]$. At each step, we either restart over to the state $q$, with probability of *alpha*, or continue to walk using the normalized adjacency matrix $A$. The state sequence is therefore

$$p_{k+1} = \alpha q + (1-\alpha)A \cdot p_k \tag{4}$$

It turns out that this random walk with restart is actually a normal random walk but with an adjusted matrix (and respectively, an adjusted weighted directional graph).

**Lemma 3.3.** *Let $q$ be any state, let $\alpha \in [0, 1]$ and let $Q = (q|q|\ldots|q)$ (The square matrix whose every column equals $q$). Then $Q$ is a transition and for any state $p$ we have $Qp = Q$.*

*In addition if $T$ is any transition then $W = \alpha Q + (1-\alpha)T$ is also a transition, And we can rewrite the RWR from 4 as*

$$p_{k+1} = \alpha q + (1-\alpha)Ap_k = [\alpha Q + (1-\alpha)A]p_k = Wp_k$$

*Proof.* Trivial and uses 3.2 □

The matrix $W$ represents a graph $G'$ where each edge of the original graph $G$ is rescaled by a factor $1-\alpha$ (and if $G$ is undirected we treat each undirected edge as 2 directed edges in $G'$. In addition from each vertex $v$ we add edges to every other vertex and the weight of the additional edge is $\alpha q[u]$.

If we pick the restart state $q$ in a way that makes $W$ primitive, then 2.5 assures us that the RWR will converge, $\lim_{k\to\infty} p_k = \lim_{k\to\infty} W^k p_0 = p$, where $p$ is the unique stationary distribution of $W$. This means in particualr that we can use the power method to find out the distribution $p$ by calulated $p_k$ until it sufficiently converges, and it will converge to $p$ from any intial state $p_0$ which we choose.

Also we have

$$\begin{gathered} p = Wp = \alpha q + (1-alpha)A \\ [I-(1-\alpha)A]p = \alpha q \end{gathered} \tag{5}$$

We will see later that we can invert the matrix in the second equation of **??** and use a direct solution for $p$ instead of the power method. The power method has the advantage that we can use the matrix $A$ implictly, because we only need to know $A \cdot p_k$ to compute $p_{k+1}$. $A$ is usually a sparse matrix because each vertex usually only has few neighbors and so we can use matrix free methods efficiently to calculate $p$ but that is beyong the scope of this thesis.

In the particular case of pageRank, we choose a uniform restart state $q = \frac{\mathbf{1}}{\mathbf{n}}$. The corresponding matrix $Q = (q|\dots|q) > 0$ is strictly positive, and therefore $W = \alpha Q + (1-\alpha)A > 0$ is positive and therefore primitive.

The stationary distribution which corresponds to this uniform restart state $q$ is called the **PageRnak** for $G$ with restart parameter $\alpha$. It is used to order the vertices according to their 'relevance' in the network.

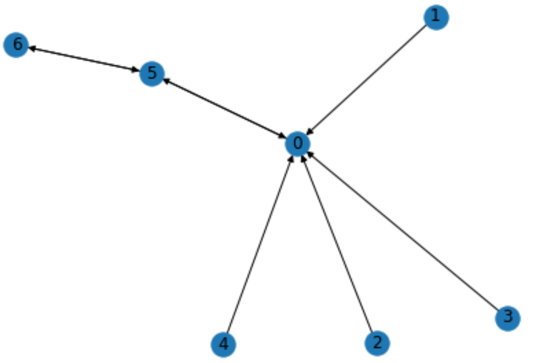


Figure 1: This is an example of a weakly connected graph. If we start walking from 0, 5 or 6 we can never reach vertices $1-4$. It is also not immediately clear which vertex 0 or 5 is more 'important'. While 0 is directly connected to more vertices, 5 may get more 'flow' through it since every path of length 2 or more passes though it.

The PageRank is the stationary distribution of the process when we use unbiased restart—A restart is equally likely from any vertex. But we want more. We want to find out what happens when we restart, for example, always from one single vertex $u$. We think of the stationary distribution $p_u$ that results from such process as the heat (or flow) which propagates out of $u$. If we

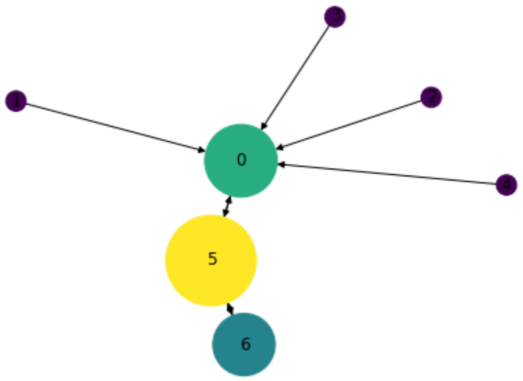


Figure 2: The same graph with vertex size and color indicating its PageRank with parameter $\alpha = 0.15$. We see that 5 is ranked first, followed by 0. The smaller the restart parameter, the more important 5 will get because we allow for longer paths with fewer restarts.

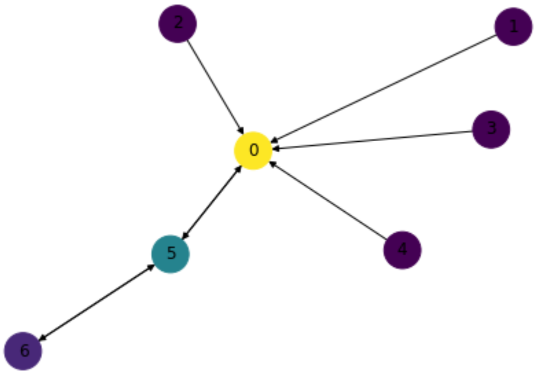


Figure 3: When the restart parameter is too large, the rank becomes almost uniform because the convex combination of the original graph with the $n$-clique graph of the restarts is weighted too heavily on the latter. It also shows that for high restart values 0 becomes hotter than 5. That is because paths of length $> 2$ (before restart) become very unlikely in this random walk.

take another vertex $v$ we think of $p_u[v]$ as a measure of how close $v$ is to $u$, or how much heat $u$ sends to $v$. Note that this is not symmetric $p_v[u] \neq p_u[v]$ in general.

**Lemma 3.4.** *Let $A \geq 0$ be irreducible. Let $B \geq 0$ have a positive row (or column), then $A + B$ is primitive.*

*Proof.* Suppose WLG that $B_1 > 0$ (first row). Then $\forall k > (B^k)_1 > 0$. Fix $i, j$. Since $A$ represents a strongly connected graph, there is a path from $i \curvearrowright 1 \curvearrowright j$ of some length $k$. So $A^k_{i,j} > 0$. Then $\forall l \geq 0 (A + B)^{k+l} > 0$ because we can go along this path, then self loop $l$ times in 1 and keep going to $j$ on that path.

So we have shown that $(A + B)^k_{i,j}$ stays positive once it becomes positive and it always turns

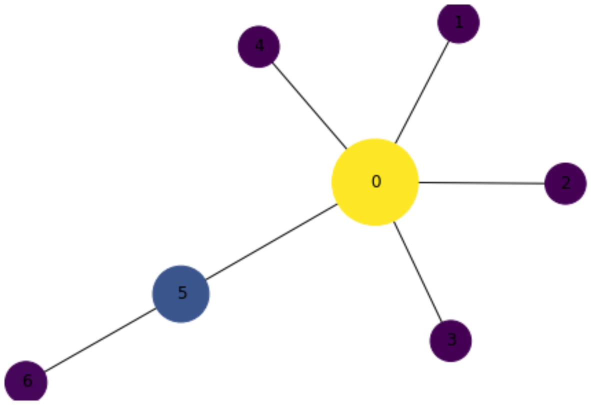


Figure 4: When we treat the graph as undirected and calculate the PageRanks ($\alpha = 0.15$), Vertex 0 comes this time on top. It is more central than 5 and more random paths intersect it than any other vertex.

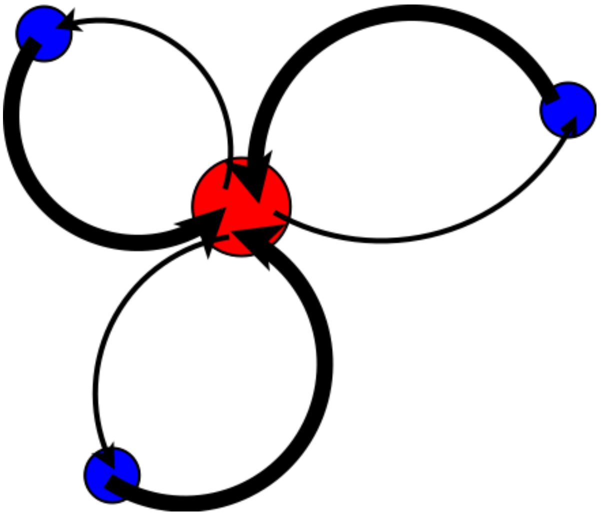


Figure 5: Stars have hot center because it spreads its heat on many sources, while each of the orbiting vertices sends all its heat into the star center.

positive at some point by irreducibility of $A$, so that means $A + B$ is primitive. □

*Remark* 6. Lemma 3.4 proves that we can propagate (namely do RWR) from any arbitrary restart state $q$, including a single vertex and the combined transition matrix will be primitive if the adjacency matrix $A$ is irreducible, or equivalently, the corresponding graph $G$ is strongly connected.

Assume that $A$ is a normalized adjacency transition of a strongly connected graph $G$. Let $q$ be any state column vector, for example $e_1$ if we restart from a single vertex, and let $Q = (q|q|\dots|q)$. So $Q$ has a positive row and therefore $(1-\alpha)A + \alpha Q$ is a primitive transition according to 3.4.

**Definition 3.3.** Let $G$ be a graph with adjacency matrix $A$, and let $D$ be the diagonal matrix of the out ranks of the vertices of $G$. Then we can column normalize $A$ and create the transition

$T = AD^{-1}$. We define **the transition matrix with restart parameter $\alpha$ and bias $q$** as

$$T_{\alpha,q} := (1-\alpha)T + \alpha Q$$

In general, the matrix $T_{\alpha,q}$ may have more than one unique stationary distribution $p$ (I think there is one for each strongly connected component of its corresponding graph), so lets require that $G$ be strongly connected (which means $A$ is irreducible), And use 6:

$$p = Ip = [(1-\alpha)T + \alpha Q]p = (1-\alpha)Tp + q$$

We can rearrange it now

$$(I - (1-\alpha)T)p = \alpha q \tag{6}$$

We want to invert the matrix in 6 and use the following lemma to justify it (The proof is easy. See for example Serre [13]):

**Lemma 3.5.** *Let $X$ be a contracting matrix, Then $(I - X)$ is invertible and the power sum of $X$ converges to it:*

$$(I-X)^{-1} = \sum_{k=0}^{\infty} X^k$$

So now we may apply lemma 3.5 on equation 6 because $(1-\alpha)A$ is contracting, and we have:

$$p = \alpha[I - (1-\alpha)T]^{-1}q := Kq = \alpha \sum_{k=0}^{\infty}(1-\alpha)^k T^k \tag{7}$$

$K$ is called **diffusion kernel** of $T$ with parameter $\alpha$ It turns out that $K$ itself is a transition because it maps the arbitrary transition $q$ to the transition $p$. In addition, $K > 0$ because of 7 and since $T$ is irreducible.

What about the eigenvectors and eigenvalues of $K$? If a matrix $A$ is invertible then $Av = \lambda v \iff A^{-1}v = \lambda^{-1}$. For any matrix $Av = \lambda v \iff (I + A)v = (1+\lambda)v$.

It turns out then that $T$ and $K$ have the same eigenvectors and if $Tv = \lambda v$ then $Kv = \alpha[1-(1-\alpha)\lambda]^{-1}v$. And in particular if it turns out that if all the eigenvalues are real (spoiler- they are), then they have the same linear order as eigenvalues of $K$ or $T$ for the same eigenvector. In particular we see that the choice of $\alpha$, the restart parameter, NEITHER changes the eigenvectors NOR does it change the order of the eigenvalues.

To sum up the important facts that we would need later:

**Theorem 3.1.** *Let $G$ be strongly connected undirected graph. Let $A$ be its adjacency matrix and $D$ the diagonal matrix which has the ranks of the vertices on its diagonal. Let $T = AD^{-1}$, let $0 < \alpha < 1$ and $K = \alpha[I - (1-\alpha)T]^{-1}$. Then the following are all true:*

- *$A \geq 0$ is symmetric therefore its eigenvalues are all real.*
- *$T = AD^{-1} = D^{1/2}[D^{-1/2}AD^{-1/2}]D^{-1/2}$ is similar to $A$. Therefore it has the same eigenvalues as $A$, which are all real: $\lambda_1 \geq \dots \lambda_n$.*

- *There exists for all $0 < \alpha < 1$ the invertible matrix: $K := \alpha \sum_{k=0}^{\infty} (1-\alpha)^k T^k = \alpha[I - (1-\alpha)T]^{-1}$. $K > 0$ is a transition. It has the same eigenvectors as $T$ and their corresponding eigenvalues are all real and they have the same order as the corresponding eigenvalues for $T$ have.*

Finally as a side note the name diffusion kernel originates from the heat equation which associated with the graph, considering the graph as perfectly insulated system. We won't get deeper into that at this time.

# 4 A naive yet not useless propagation method for clustering a graph

In this section we come up with a very simple simple clustering algorithm which usese RWR to divide the well known Zachary's Karate Club graph. The results can be verified by the actual division of the subjects of the research into 2 clubs.

**Definition 4.1.** Let $G$ be a strongly connected graph and $K$ the diffusion kernel as defined in 3.5, end let $u, v \in \{1 \dots n\}$ be two vertices and $e_u, e_v \in \mathbb{R}^n$ their corresponding index vectors.

$p_u := Ke_u$, the stationary distribution with random restart from $u$, is called the **influence vector** of vertex $u$. We say that $p_u[v]$ is the **influence of** $u$ on $v$.

*Remark* 7. We use the definition of influence as presented in the Hotnet and Hotnet2 artticles [16] and [6].

A $p_u$ is a measure of how 'heat' which is pumped into $u$ propagates in the graph. When we try to cluster the graph, it is natural to think that If vertex $v$ is the top receiver from $u$, namely $v = \text{argmax}(p_u)$ then maybe these 2 belong in the same cluster.

Informally we say that a vertex is 'hot' or 'cold' if it has a high (hot) or low (cold) PageRank. Remember that the PageRank is the unbiased stationary distribution $p = K \cdot (1/n, \dots, 1/n)^t$.

We suggest that it makes sense to pick up a cold vertex and associate it with the vertex to which it sends most of its heat. If we start from a hot vertex, it usually has many neighbours and it doesn't send allot of heat down a single vertex.

The algorithm works as follows:

```
function coolWarmClustering(G, k)
  # Input G = (V,E) : a directed strongly connected graph.
  # Input m : The desired number of clusters
  K <- diffusion_kernel(G)
  H <- Disconnected_undirected_graph_on(V) # each vertex starts in its own cluster
  p <- pageRank(G)
  vlist <- arg_sort(p) # vertex-list sorted up by PageRank
  While |connected_components(H)| > k do
    x <- vlist.pop() # take the coldest remaining vertex and remove it from the
                      # list
    p_x <-K * e_x # Influence vector of x
    y <- argmax(p_x[i : i in vlist]) # exclude the already visited
```

```
                                      # nodes
    H.add_edge(x,y)
  return H
```

The algorithm clusters the vertices by constructing a new graph on the same vertices, successively joining vertices. It picks the coldest remaining vertex and joins it with the vertex to which it propagates the most heat, among the yet unvisited vertices.

In every iteration of the algorithm adds an edge from the currently selected vertex to one of the remaining vertices which haven't been selected yet.The number of connected components by 1. Because $G$ is strongly connected the algorithm will reduce the number of connected components in $H$ until it reaches the desired number of components $k$.

It is essential to run the algorith in the order from lower pageranked vertices upwards. If we start for example the other way around, from the warmest node downwards, the list of remaining nodes contains the coldest and least 'relevant' nodes.

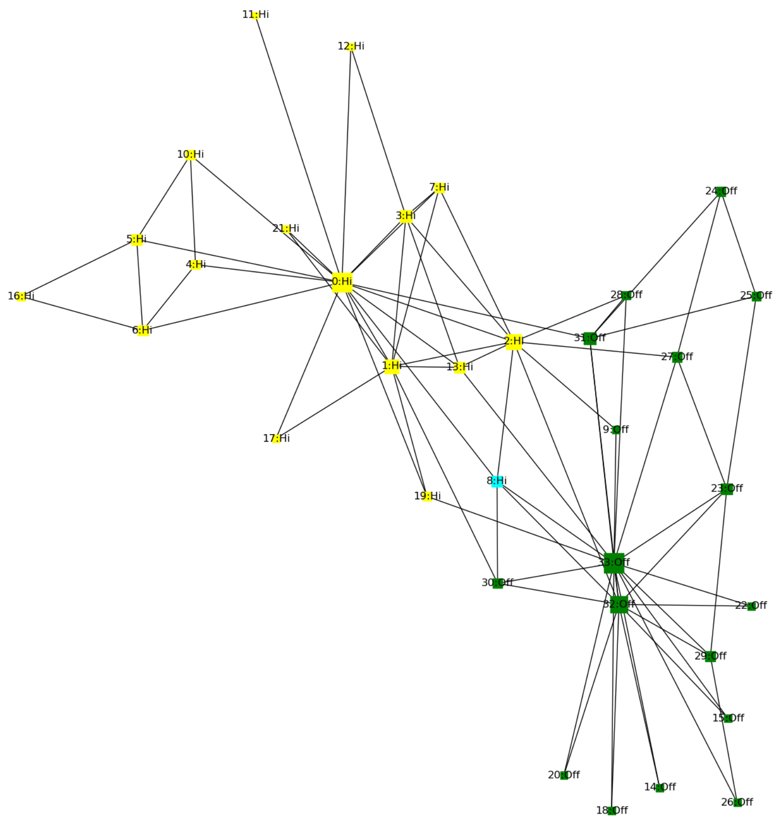
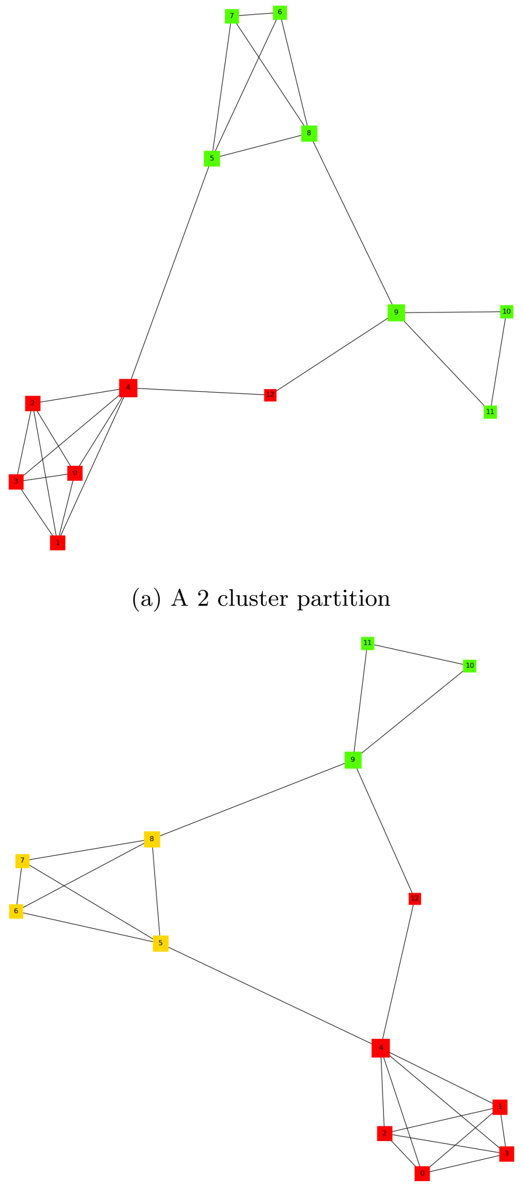


Figure 6: The Famous 'Zachary's Karate Club' Graph. The labels show the actual division between the 'Mr Hi' and 'Officer' groups. Vertex size indicates its PageRank. The colors encode the result of the coolWarmClustering compared to the actual partition. It made 1 mistake: It associates wrongly vertex 8 to 'Officer'. This vertex is hard to resolve because it is connected to hubs in both clusters. In fact a further check confirms that the total sum of 'heat' it sends to members of 'Officer' is 0.45 vs 0.36 to members of 'Mr Hi' (excluding itself in both cases). So in some sense the algorithm is more correct then the actual partition in real life. Perhaps this is due to human irrationality? On the other hand, 8 receives more heat units from 17 members of 'Mr. Hi': 0.485 units, compared to 0.345 units from the 16 members of 'Officer'.



(a) A 2 cluster partition

(b) A 3 cluster partition

Figure 7: The result of running the algorithm set for finding a 2-partition and respectively a 3-partition. The interesting node is 12. In both runs 12 chooses to associate with the biggest clique because node 4 of that clique is the one which receives the most heat from 12

# 5 Spectral clustering and other methods

**Spectral Partitioning** [18, 8, 4, 14], is a common technique for partitioning graphs. The idea is to construct a Laplacian type matrix [5] for the graph $G$, analyze its eigenvalues and eigenvectors and construct a 2-partition of $G$. It is possible to create a hierarchical partitioning of $G$ by repeating the process on each subdivision.

The problem of finding the best graph partition, also called the minimal cut problem is NP-hard. Spectral partitioning methods are heuristics that approximate the solution in 'real life' classes of networks.

**Definition 5.1.** Let $G$ be a bidirectional graph with adjacency matrix $A$ and diagonal degree matrix $D$. Then the following matrices are its **graph Laplacian**, and its **symmetric—, row–normalized—and column–normalized—Laplacians**:

$$\begin{aligned} L &= D - A \\ L_s &= D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2} \\ L_r &= D^{-1} L = I - D^{-1} A \\ L_c &= L D^{-1} = I - A D^{-1} \end{aligned} \tag{8}$$

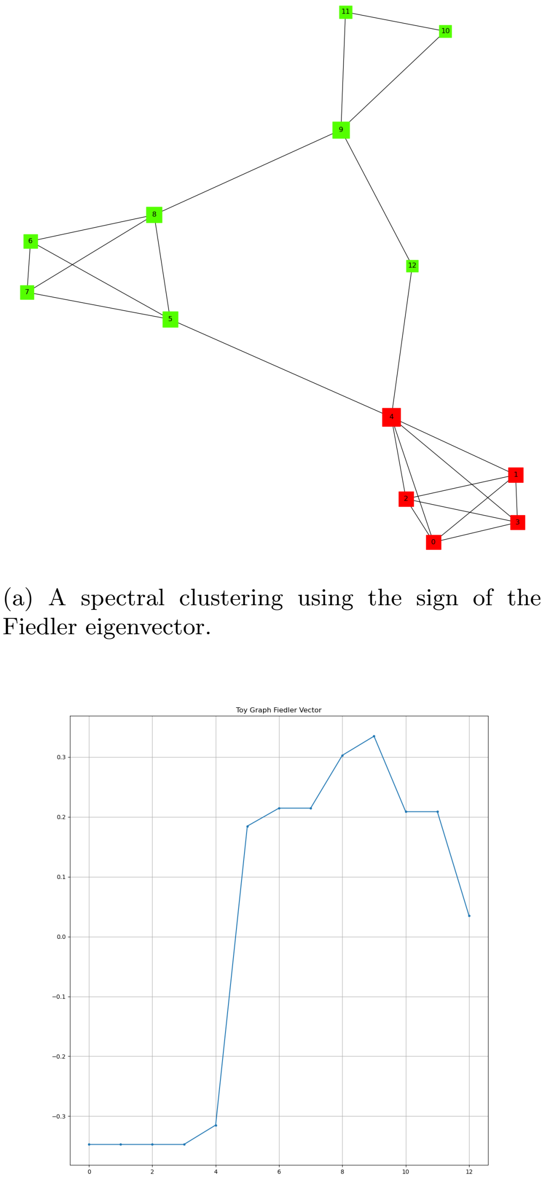
Consider the connected graph $G$ and its adjacency matrix $A$. We create the transition matrix by normalizing it: $T = AD^-1$, and finally we choose a restart parameter $\alpha$ and create the diffusion kernel $K = \alpha[I - (1-\alpha)T]^{-1}$. Theorem 3.1 assures us that $K$ and $T$ have the same eigenvectors and the orders of their corresponding eigenvalues are the same. The eigenvalues are all real and therefore the eigenvectors are real as well. And the eigenvalues of $K$ are all non-negative.

The matrix $K^{-1} = \alpha^{-1}[I - (1-\alpha)T]$ is closely related to the symmetric Laplacian $L_s$ and thecolumn-normalized Laplacian $L_c$. The added parameter $0 < \alpha < 1$ neither changes the eigenvectors, nor the order of the corresponding eigenvalues.

The smallest eigenvalue ,0 of $L$ and $L_s$ corresponds to the largest eigenvalue 1 of and $T$ and its eigenvector is (can be chosen as) all positive. Any other eigenvector of $L$ (which corresponds to an eigenvector of $T$) must contain both positive and negative components due to 2.5.

It's the eigenvector of the second smallest eigenvalue $L$ or $L_s$ which is commonly used in They are commonly referred to as the **Fiedler** eigen-pair in the literature. for spectral partitioning [8]. The simplest method is to use the sign of the components of eigenvector of the second smallest eigenvalue of $L$ or $L_s$. This eigen-pair corresponds to the second largest eigenvalue of $T$

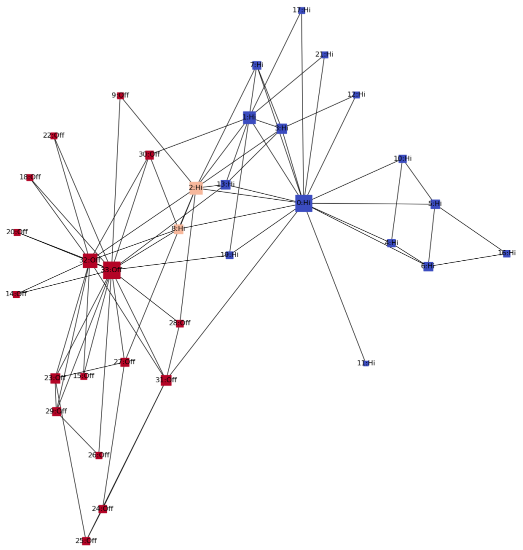
In Newman [10] there is a particular type of spectral partitioning which uses a matrix which the author called the modularity matrix. It has the advantage over the Laplacian in that its greatest eigenvalue may be positive or negative which is used to indicate or suggest the existence of community structure in the graph. However the normalized Laplacians $L_r$ and $L_c$ are more natural in the context of the random walk or propagation method and recommended for example in [18].



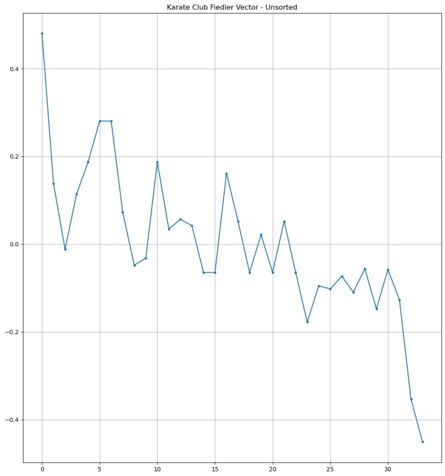
(a) A spectral clustering using the sign of the Fiedler eigenvector.

(b) Plot of the Fiedler vector's components, showing a clear subdivision between the positive and the negative components.

Figure 8



(a) Spectral clustering of the Karate Club network using the Fiedler eigenvector signs. There are 2 deviations from the actual partition, both are borderline nodes which are harder to resolve. The 8 node mistake is in common with the naive algorithm of the previous section.



(b) Plot of the Fiedler Vector of the Karate Club. The split between negative and positive values is much less dramatic but nonetheless still visible.

# 6 (Trying) function prediction method

In this section we are going to try to apply the accumulated knowledge of the previous sections and devise some simple function prediction algorithms for PPI networks. We are going to try several methods which use propagation as well as methods which just use direct neighbors for the predictions.

The network is going to be composed of around 500 vertices (proteins) which are divided into 4 different function groups. We are going to randomly choose half the proteins and mark them as unknown and try to predict their function based on the remaining known proteins.

## 6.a Tools and Source Data

We sourced our data from *Cytoscape* [2], an open source java software for network analysis and visualisation. The software package contains as one of its built in examples a fully annotated yeast interactome. This is a PPI network of thousands of proteins and it contains among many other types of edge and node annotations information about protein function and location in the 'MCs name' field.

We preferred to work within the familiar Python 3 programming language and its rich world of extension packages. Therefore we exported the yeast interactome into a graphml format which was later imported by our network analysis tool of choice, *Networkx* [9]. It is a python based open sourced network analysis package which integrates extremely well with other essential Python packages. Networkx is not as versatile or powerful as Cytoscape and especially rudimentary in its visualisation functionality, but it runs smoothly and easy to work with, unlike Cytoscape. We used the *Numpy* [11] package for vector, matrix and other types mathematical calculations, and *Pandas* [12] for some database operations.

## 6.b Methodology

As mentioned, we used the annotated yeast interactome which is a pure PPI network that comes prepackaged with the software tool 'Cytoscape'. The nodes represent yeast proteins and the edges represent protein-protein interactions.

This network comes with functional annotations in the 'MCs name' field. We selected nodes in the network whose annotations contain **exactly** one of the following keywords: 'DNA Replication' (blue group), 'Golgi' (red), 'Meiosis' (yellow), and 'Stress response' (green). All other nodes are removed from the network. The result is shown on figure 10.

The resulting subnetwork is not a connected graph, which is the type of input data that we wanted to run our test algorithm on. Therefore we selected the biggest connected component of that subnetwork, and that is the graph on which we tested and compared various prediction algorithms.

For the first set of comparative tests of different prediction methods we set a random seed (a process which we repeat 2 times for 2 different trials). Then we selected randomly 50% of the nodes and marked them as 'unknown'. The other 50% are 'known'. We would then try to determine the group affiliation of the unknown nodes based on the group affiliations of the known nodes.

We tried 5 different prediction methods:

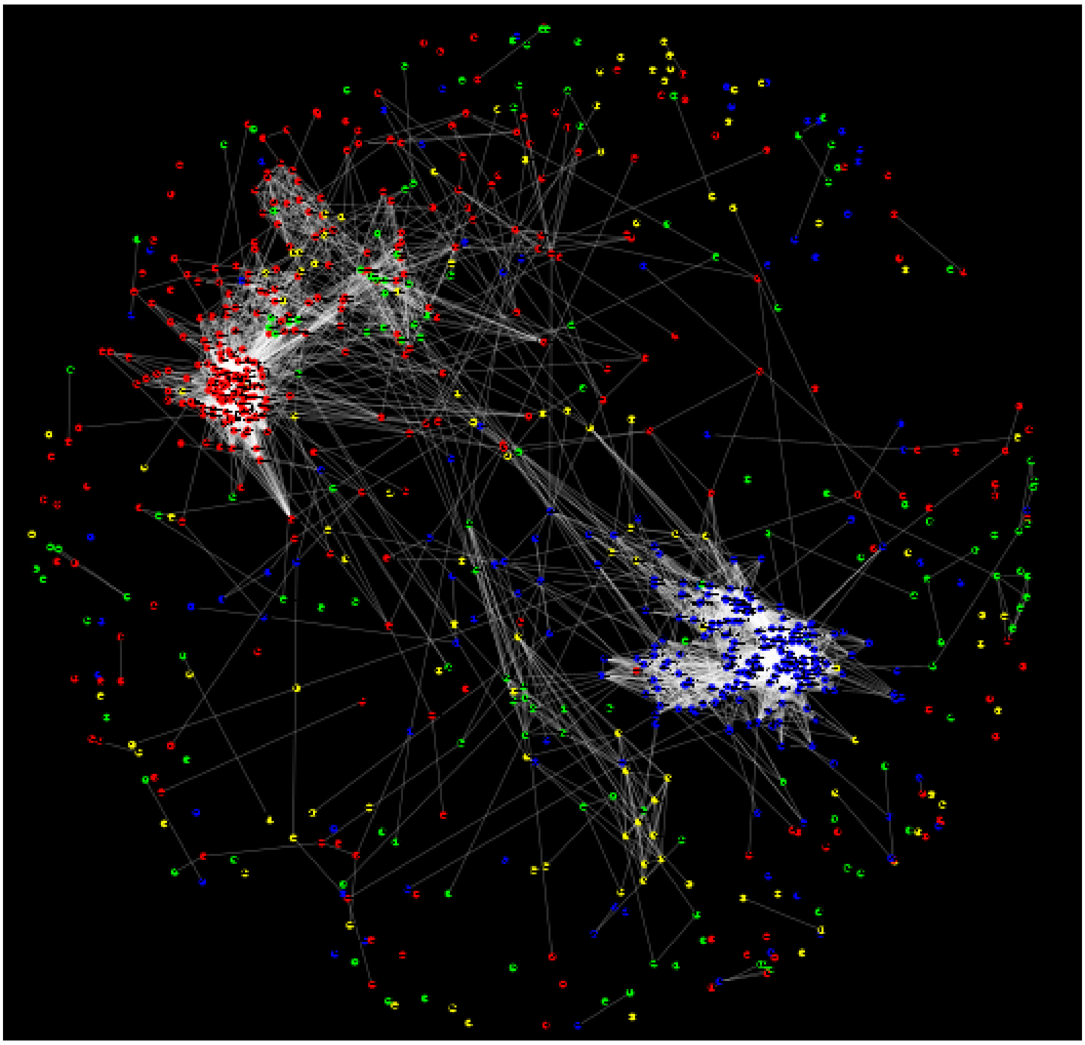


Figure 10: Subgraph of the yeast interactome showing nodes of 4 groups: 'DNA replication' (blue), 'Golgi' (red), 'Meiosis' (yellow), 'Stress response' (green)

- Method 1 Determines the group affiliation of an unknown node by the group that has the maximal volume based on the propagation distribution from the unknown node. This meas: we calculate the stationary distribution with restart to the unknown node. Among the known nodes, we calculate the weighted sums of each of the 4 groups according to that distribution. The group with largest volume is the affiliation we assign to the unknown node.

- Method 2 This method works like method 1, except that we first order the unknown nodes (decreasing order) according to their pageRank. We go over the nodes in this order and assign group affiliation according to method 1, but then we also update the list of the known nodes to include that node. So this node is going to be included in the function prediction of the lower ranked nodes.

  The rational here is that the higher ranked nodes are probably going to be predicted with high accuracy and therefore be helpful in prediction of lower ranked nodes.

  It is probably better to device some simple rule to decide whether it is worth to include a node in the prediction of the rest of the nodes. For example, based on how many unknown neighbors in has vs known neighbors. But we tried to keep things simple at this stage.

- Method 3 We assign an unknown node to the group that has plurality among its neighbors with known group affiliation. So this method is fast and the probably the simplest.

- Method 4 Same as method 3, but again we go in the order of pagRanks and we update the list of known nodes on the fly, like method 2.

- Method 5 Here we take each group of the known nodes and propagate from it. So for example we calculate the stationary distribution when we restart in the known nodes that belong to group 'DNA Replication', and then for the next group and so forth. For each unknown node, we assign it to the group that propagates the highest probability to it.

  So while in method 1 and 2, we start random walking from an unknown node and see what is the probability that we land at a certain group. In Method 5 we start walking randomly from one of the members of a group and check what is the probability that we visit a certain unknown node.

  This method is an order of magnitude faster than method 1 and 2 because we only need to calculate the stationary distributions 4 time (pageRank and for each group). However we suspect it will perform with less accuracy. Because they take into account nodes that are far and not well connected to the unknown node whose function we try to predict. The function groups, in particular the yellow (meiosis) and green (stress) are not well clustered and spread all over the network.

## 6.c Some code snippets

```
# The decision function which is used in methods 1 and 2 to predict a single
# node
# This version uses a stored diffusion kernel to calculate the
# propagation, rather than using the power method multiple times.
def decision_function_diffkernel(v, G, K, group_membership_ar, known_unknown_ar):
    """
    This function uses the given diffusion kernel of the graph to
    calculate the propagation, rather than using the power method.
    Input v: an node assumed to be an int and taken from the list of unkown
    nodes. Input G: The graph.
    Input K: the diffusion kernel of the graph.
    Input group_membership_ar: Array of strings which specifies for each node
    to which group belongs (including the 'unkonw' nodes) this is required for
    thesting the correctness of the decision.
    Input known_unknown_ar: boolean 1d array which
    specifies for each node of the graph whether its membership is known or
    unkown.
    output prediction: A string which is the predicted group affiliation.
    output correctness: True or false depending on the correctness of the
    prediction vs the real group_membership_list[v] value.
    """
    all_nodes = np.arange(len(G.nodes))
    known_nodes = all_nodes[known_unknown_ar]
    unknown_nodes = all_nodes[known_unknown_ar == False]
    bias = np.zeros_like(G.nodes)
    bias[v] = 1
    bias
    p = np.dot(K, bias)
    group_names = np.unique(group_membership_ar)
    q = p * known_unknown_ar  # 0 on all unkown nodes
    testscores = np.zeros_like(group_names)
    for g in range(len(group_names)):
        x = q[group_membership_ar == group_names[g]]
        testscores[g] = x.sum()
    decide = group_names[np.argmax(testscores)]
    correctness = decide == group_membership_ar[v]
    return decide, correctness

def predictMethod2_diffkernel(G, tries=1, knownfraction=0.5, seed=42, alpha=0.2):
    """
    Like predictMethod2 but uses diffusion kernel rather than power
    method.
    Input G: a graph.
    Input tries: how many repetitions to perform, each witha
    different randomly selected 'known' group.
    Input knownfraction: portion of the known nodes out all the
    nodes.
    Input seed: random seed.
    Input alpha: restart parameter.
    """
    groups = [G.nodes[x]["Group"] for x in G.nodes()]
    groups = np.array(groups)  # array is better than a simple list ...
    group_labeling = np.unique(groups)
    df = pd.DataFrame()
    df["Group (ground truth)"] = groups
    scores = np.zeros(tries)
    # determine the pageRanks
    K = diffKernelG(G, alpha=alpha)
    pageRank = biasedPropagateGA(G, bias=np.ones_like(G.nodes), alpha=0.2)
    orderedNodeList = np.argsort(pageRank)
    # set the random seed
    np.random.seed(seed=seed)
    for t in range(tries):
        groups_predict = groups.copy()
        known_unknown = np.random.random(len(G.nodes)) < knownfraction # known=1
        all_nodes = np.arange(len(G.nodes))
        known_nodes = all_nodes[known_unknown]
        unknown_nodes = all_nodes[known_unknown == False]
        orderedUnkownNodeList = orderedNodeList[known_unknown == False]
        known_unknown2 = known_unknown.copy()
        score2 = 0
        df["Rep " + str(t)] = "known"
        df["Rep " + str(t) + "_predict"] = groups_predict
        for v in orderedUnkownNodeList:
            # predict, test = decision_function(v, G, groups, known_unknown2, alpha=alpha)
            #predict, test = decision_function_diffkernel(
            #    v, G, K, groups, known_unknown2
            #)
            predict, test = decision_function_diffkernel(
                v, G, K, groups_predict, known_unknown2
            )
            groups_predict[v] = predict
            score2 += test
            known_unknown2[v] = True # mark v as 'known'
            df["Rep " + str(t)][v] = "correct"
            df["Rep " + str(t) + "_predict"][v] = predict
            if not test: # mark as incorrect and colormark if mistake
                df["Rep " + str(t)][v] = "mistake"
        score2 = score2 / len(unknown_nodes) # got 0.85
        scores[t] = score2
    return scores, df
```

## 6.d Results

Table 1: Prediction Accuracy of the 4 Methods in 2 different random trials. Each trial with a different seed, which result in different randomly selected known/unknown nodes.

| Seed / Method | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 42 | 0.795 | 0.854 | 0.765 | 0.791 | 0.701 |
| 6382020 | 0.774 | 0.832 | 0.751 | 0.751 | 0.755 |

We have additionally performed 50 repeats for different parameters with method 2. The best mean result, of 0.8926 mean accuracy was obtained with a restart probability of 0.31. Restart parameters ($\alpha$) of $0.2 - 0.35$, lets call it mid-lower range produce the best results, while with higher alphas accuracy falls towards 0.5 which is of course the accuracy of just randomly guessing.

Accuracy was about 0.8 when we start with only 10% labeled and it gradually increases with increased labeled proportions (0.85 accuracy at 0.3 label coverage).

We have conducted another test of method 2, with parameters $\alpha = 0.31$, and 0.35 label coverage. This time we calculated the sensitivity, specificity, accuracy and precision for each label as well as the total. The results are the following:

```
     Label       P      TP     FP         N        TN     FN  Sens  Spec   Acc   PPV
0    golgi     165     149     25       209       184     16  0.90  0.88  0.89  0.86
1 DNA_repl     133     129     22       241       219      4  0.97  0.91  0.93  0.85
2  meiosis      30      17      4       344       340     13  0.57  0.99  0.95  0.81
3   stress      46      25      3       328       325     21  0.54  0.99  0.94  0.89
4    Total  374.00  320.00  54.00  1,122.00  1,068.00  54.00  0.86  0.95  0.93  0.86
```

We can see that the problem lies with the two small and spread out groups, 'meiosis' and 'stress'. It is hard to predict that an unlabeled vertex belong to these groups

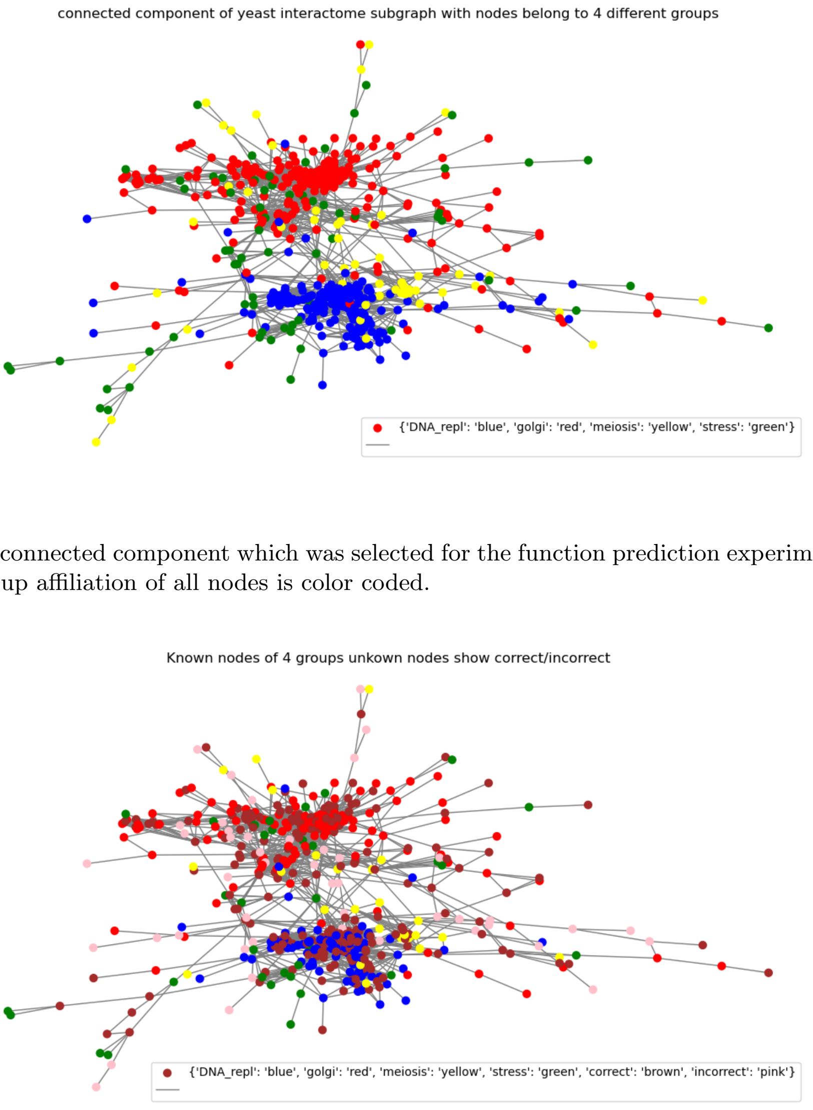
## 6.e Discussion

If we look at figure 11a which shows the connected component that we worked with, the red and blue are pretty nicely clustered. The The greens and specially the yellows are sort of spread all over the network and not nicely clustered. I don't know if this is a good example of a real world scenario.

When we look at figure 11b it seems that most of the incorrect predictions are on nodes that seem 'peripheral' and are not well connected to known nodes. Another source of mistakes seems to bee green nodes that are too close to the red cluster.

Method 2 shows somewhat higher accuracy over the other methods but on the flip side it is an $O(n^2)$ method when optimized methods are used whereas methods 3,4,5 are $O(n)$.

It's probably woth while to repeat the experiment a couple of hundred of times to estimate the mean accuracies of the methods.



connected component of yeast interactome subgraph with nodes belong to 4 different groups

{'DNA_repl': 'blue', 'golgi': 'red', 'meiosis': 'yellow', 'stress': 'green'}

(a) The largest connected component which was selected for the function prediction experiment. Here the Correct group affiliation of all nodes is color coded.

Known nodes of 4 groups unkown nodes show correct/incorrect

{'DNA_repl': 'blue', 'golgi': 'red', 'meiosis': 'yellow', 'stress': 'green', 'correct': 'brown', 'incorrect': 'pink'}

(b) The result of predictiob method 2, which was the best perfomer. The known groups are color coded by RGBY. The unkown nodes are color coded: brown:correct prediction, pink:incorrect.

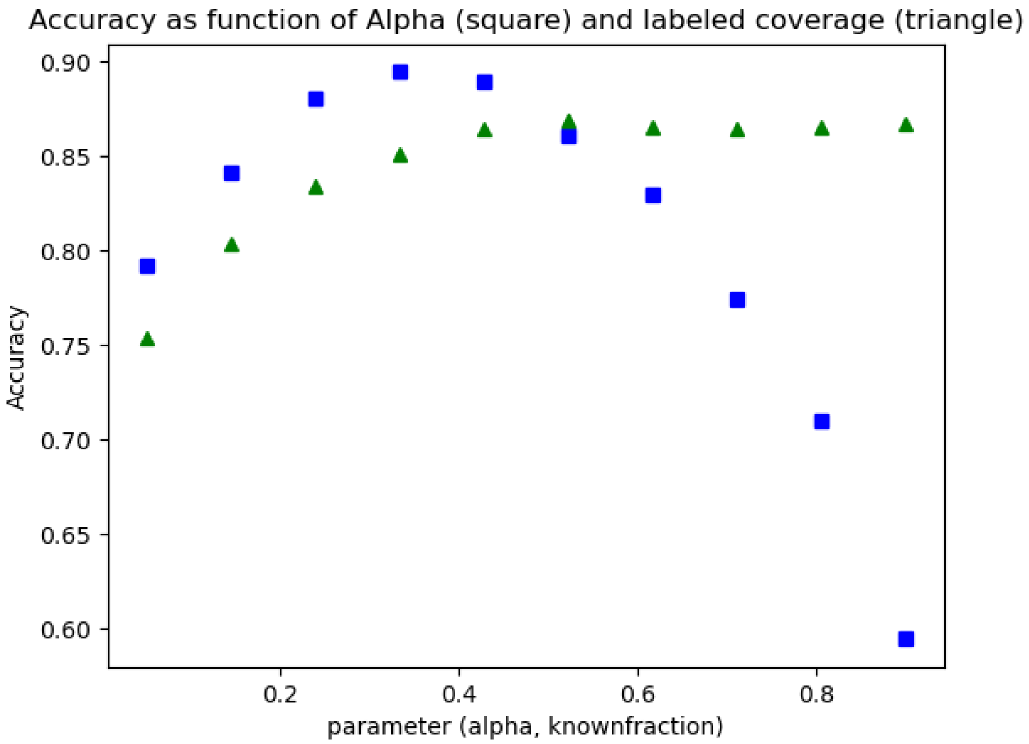


Figure 12: The accuracy as function of the resart parameter alpha, with fixed label coverage of 0.5, and of the label coverage with fixed alpha=0.2

# 7 Further Discussion and Remarks

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

# 8 Reference

## References

[1] Lenore Cowen et al. “Network propagation: a universal amplifier of genetic associations”. In: *Nature Reviews Genetics* 18.9 (2017), p. 551.

[2] *Cytoscape*. Cytoscape Consortium. 2020. URL: https://cytoscape.org (visited on 09/11/2020).

[3] Minghua Deng et al. “Prediction of protein function using protein-protein interaction data”. In: *Proceedings. IEEE Computer Society Bioinformatics Conference*. IEEE. 2002, pp. 197–206.

[4] *Graph Partition*. Wikipedia. 2020. URL: https://en.wikipedia.org/wiki/Graph_partition (visited on 09/11/2020).

[5] *Laplacian Matrix*. Wikipedia. 2020. URL: https://en.wikipedia.org/wiki/Laplacian_matrix (visited on 09/12/2020).

[6] Mark DM Leiserson et al. “Pan-cancer network analysis identifies combinations of rare somatic mutations across pathways and protein complexes”. In: *Nature genetics* 47.2 (2015), pp. 106–114.

[7] Carl D Meyer. *Matrix analysis and applied linear algebra*. Vol. 71. Siam, 2000.

[8] Maxim Naumov and Timothy Moon. *Parallel spectral graph partitioning*. Tech. rep. NVIDIA Technical Report, NVR-2016-001, 2016.

[9] *Networkx*. NetworkX developers. 2020. URL: https://networkx.org (visited on 09/11/2020).

[10] Mark EJ Newman. “Modularity and community structure in networks”. In: *Proceedings of the national academy of sciences* 103.23 (2006), pp. 8577–8582.

[11] *Numpy*. The NumPy project. 2020. URL: https://numpy.org (visited on 09/11/2020).

[12] *Pandas*. Numfocus. 2020. URL: https://pandas.pydata.org (visited on 09/11/2020).

[13] D Serre. “Matrices theory and applications second edition”. In: *Graduate Texts in Mathematics* 1.216 (2010), ALL–ALL.

[14] *Spectral Clustering*. Wikipedia. 2020. URL: https://en.wikipedia.org/wiki/Spectral_clustering (visited on 09/11/2020).

[15] Martin Szummer and Tommi Jaakkola. "Partially labeled classification with Markov random walks". In: *Advances in neural information processing systems*. 2002, pp. 945–952.

[16] Fabio Vandin et al. "Discovery of mutated subnetworks associated with clinical data in cancer". In: *Biocomputing 2012*. World Scientific, 2012, pp. 55–66.

[17] Martin Vingron. "Propagation methods (Lecture Slides)".

[18] Ulrike Von Luxburg. "A tutorial on spectral clustering". In: *Statistics and computing* 17.4 (2007), pp. 395–416.

## List of Figures

## List of Tables