Summer

Write about more papers

debjit.paria1999

June 2019

Problem

We will try to solve the graph bipartition or biclustering problem. And the graphs are always connected. And a lot of time I might use 'i' in place of 'v_i' in the context of graph theory.

In this whole discussion the measure of a good partition or cluster is its conductance.

The problem was to analyze a spectral clustering algorithm and try to find theoretical point to prove it's usefulness. So first we will describe the general spectral clustering algorithm using the unnormalized laplacian. Then we will see that the spectral algorithm can be made to perform as bad as we want. Then we will show how can we change the algorithm to work perfectly for cockroach graphs.
So in all scenarios we are given the weight adjacency matrix W. D is the degree diagonal matrix. And unormalized Laplacian L.

Algorithm 1: Unnormalized spectral algorithm for the bipartitioning problem

Input : Similarity matrix $W \in \mathbb{R}^{n \times m}$
Compute the unnormalized Laplacian L.
Compute the fielder eigenvector **F** of the unnormalized laplacian L.
Now get the best thresholding cut.

Now we describe the normalzied version.

Algorithm 2: Unnormalized spectral algorithm for the bipartitioning problem

Input : Similarity matrix $W \in \mathbb{R}^{n \times m}$
Compute the normalized Laplacian L_{sym}.
Compute the fielder eigenvector **F** of the unnormalized laplacian L.
Now get the best thresholding cut.

Suppose F is the fielder eigenvector of the unnormalized Laplacian. Consider the graph in figure 1.

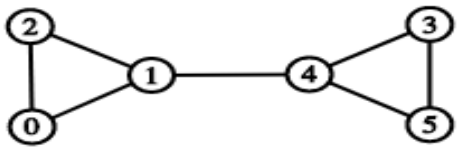

Figure 1: Two K_3 joined by an edge.

F = [0.445 0.322 0.445 -0.445 -0.322 -0.445]. Observe that the biggest edge difference $|F_i - F_j|$ is between vertex 1 and 4.

Define the edge weights between two vertices v_i and v_j for the edges in the given graph G, as $EW_{ij} = ((F_i - F_j)^2) * W_{ij}$. Intuitively the edge weights says how dissimilar the two vertices are.

Therefore we start with removing the edges which have the highest weights until we are left with two clusters.

And this algorithm gives us better results for cockroach graphs. So this spectral algorithm seems to work better for this class of graphs.

Algorithm 3: The unnormalized proposed algorithm

Input : Similarity matrix $W \in \mathbb{R}^{n \times m}$
Compute the unnormalized Laplacian L.
Compute the fielder eigenvector **F** of the unnormalized laplacian L.
Now compute the EW_{ij}.
Now we keep on removing edges one by one by descending order of weights until we are left with two clusters.

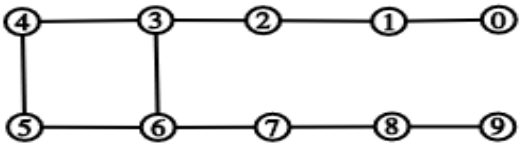

Figure 2: A Cockroach Graph

F = [0.56 0.40 0.12 0.03 0.01 -0.01 -0.03 -0.12 -0.40 -0.56]. One of the edges with highest edge cost is the edge between vertices 2 and 3. If we remove that we get the partition C_1 = ((0, 1, 2), (3, 4, 5, 6, 7, 8, 9)).
Where as the spectral clustering algorithm would give us C_2 = ((0, 1, 2, 3, 4),(5, 6, 7, 8, 9)).
Now therefore we have:-

- $RCut(C_1) < RCut(C_2)$
- $NCut(C_1) < NCut(C_2)$
- $Conductance(C_1) < Conductance(C_2)$

Thus this algorithm gives us better results for cockroach graphs.

Relevant papers read and Idea

1. Spectral Clustering based on the graph p-Laplacian: In this paper the authors defined the idea of p-Laplacian, p-eigenvalue, p-eigenvector, etc.(The normal spectral clustering algorithm uses p = 2.) He will also show how the cut found by thresholding the 2^{nd} p-eignvector will help us to get clusters and will show bounds on how well the conductance of the partition is compared to the h_G, the conductance of the graph. And will also prove that as $p \to 1$, the partition we get by thresholding the 2^{nd} eigenvector converges to the optimal Cheeger's cut.

   The Laplacian, Δ_2 used in getting theoretical results for spectral clustering algorithm is a operator which induces the following quadratic form for a function $f : V \to \mathbb{R}$,

   $\langle f, \Delta_2 f \rangle = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2$

   The generalized p-Laplacian Δ_p is an operator which induces the general form

   $\langle f, \Delta_p f \rangle = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}|f_i - f_j|^p$.

   Define the functional $F_p^{(2)}(f) = \frac{Q_p}{min_{c\in\mathbb{R}}||f-c1||}$.

Define Qp

Text

   Now we will describe few theorems which will help us realize our goal.
   Theorem 4.1: For $p > 1$ and every partition of V into two clusters $C, \overline{C}$ there exists a functional $f_{p,C}$ such that $F_p^{(2)} = cut(C, \overline{C}) \left| \frac{1}{|C|^{\frac{1}{p-1}}} + \frac{1}{|\overline{C}|^{\frac{1}{p-1}}} \right|$

Text

   with special cases $F_p^{(2)} = RCut(C, \overline{C})$

   $\lim_{p\to 1} F_p^{(2)} = RCC(C, \overline{C})$.

   Theorem 4.3**:** Denote by λ_p^2 the second p-eigenvalue of the unnormalized p-Laplacian $\Delta_p^{(u)}$. For $p > 1$

   $\left(\frac{2}{max_i d_i}\right)^p \left(\frac{h_{RCC}}{p}\right)^p \leq \lambda_p^2 \leq 2^{p-1} h_{RCC}$.

   Now consider $p \to 1$ the inequality becomes tight so we can say that in the limit, λ_p^2 approximates the Cheeger's cut arbitarily well.

   Just as we threshold on the entries of the fielder eigenvector to get a partition. We will do the same here with the 2^{nd} p-eigenvector.

   Suppose v_p^2 is the 2^{nd} p-eigenvector.

   $argmin_{(C_t=v_i\in V|v_p^2(i)>t)} RCC(C_t, \overline{C_t})$ gives us the optimal thresholding cut. Now let h_{RCC}^* denote the conductance value we got from the optimal thresholding cut given above.

   Theorem 4.4: For $p > 1$ we have, $h_{RCC} \leq h_{RCC}^* \leq p(max_{v_i\in V} d_i)^{\frac{p-1}{p}} (h_{RCC})^{\frac{1}{p}}$.

Now consider $p \to 1$ the inequality becomes tight so in the limit the thresholding cut actually converges to a optimal Cheeger's cut.

Idea: If we could prove some kind of Cheeger's inequality kind of bound for our algorithm then we can possibly also extend the idea for our algorithm also. We will use the 2^{nd} p-eigenvector v_p^2 and use this eigenvector and get the edge weights for the graph and then maybe we can prove inequalities so that as $p \to 1$ our algorithm gives us a optimal Cheeger's cut. Not only that but the proof in this paper maybe is helpful to prove bounds for our algorithm.

2. A random walk view of spectral segmentation: In this paper we view the normalized spectral clustering algorithm from a Random walk point of view. They interpret the similarity as the edge flow of a Markov random walk and also give a characterization when the NCut algorithm gives us exact results.

   We define the stochastic matrix P = $D^{-1}S$ whose row sum is one. P_{ij} represents the probability of moving from node **i** to **j** in one step, given that we are node **i**.

   Proposition 1: If λ and x are eigenvalues and eigenvectors of P resp., then $1 - \lambda$ and x are eigenvalue and eigenvectors of L_{sym}.

   So now in place of 2^{nd} smallest eigenvector of L_{sym} we can use the 2^{nd} highest eignvector of P.

   So define $\pi_i^{\infty} = \frac{d_i}{Vol(v_i)}$, $P^T\pi^{\infty} = \pi^{\infty}$ therefore π^{∞} is the stationary distribution of the markov chain with probability matrix P.

   Now define $P_{AB} = \frac{\sum_{i\in A, j\in B} \pi_i^{\infty} P_{ij}}{\pi^{\infty}(A)} = \frac{\sum_{i\in A, j\in B} W_{ij}}{Vol(A)}$.

   From this it follows that $NCut(C, \overline{C}) = P_{A,\overline{A}} + P_{\overline{A},A}$.

   If the NCut is small , then $P_{A,\overline{A}}$ and $P_{\overline{A},A}$ are both small, this intuitively says that the random walk tends to stay in its own cluster and seldom travel to its complement.

   Define a vector x to be piecewise constant w.r.t to partition $\Delta = (A_1, A_2, \ldots, A_k)$ if $x_i = x_j$ for $i, j \in A_s$.

   Proposition 2: P has k peicewise constant eigenvectors w.r.t to Δ if and only if the sums $P_{is} = \sum_{i\in A_s} P_{ij}$ are constant for all $i \in A_{s'}$ and all $s, s' = 1 \ldots k$ and the matrix $\hat{P} = [P_{ss'}]_{s,s'=1\ldots k}$ is non singular.

   Basically it wants to say if we take A_s to be one vertex and then make a Markov chain out of P. Intuitively this says that the stochastic matrix has k peicewise constant eigenvectors if and only if the underlying

Marokov chain can be aggregated into a Markov chain with state space $\Delta = (A_1, A_2, \ldots, A_k)$ and the transition probability matrix $\hat{P}$.

Now if x is a eignevector of $\hat{P}$ with eigenvalue λ then x' is a eigenvector of P with eigenvalue λ, given that $\hat{P}$ is non-singular. Where x'_i is equal to x_j if $v_i \in A_j$.

So if the eigenvalues of $\hat{P}$ are all greater than the other $n-k$ eigenvalues of P, then the NCut algorithm will work perfectly.

Idea: Think about k = 2, then we can use Proposition 2 and say that P has 2 peicewise constant eigenvectors w.r.t Δ we just need to choose a eigenvector which is not. But this isn't a very practical algorithm as in practice it might not always happen that our dataset will obey the conditions in this proposition. If the eigenvalues of $\hat{P}$ are all greater than the other $n-k$ eigenvalues of P and the conditions above are satisfied then our algorithm works perfectly.

Which is not constant

3. Linkage : In this paper, we found a new way to improve our clustering. Suppose we use the general spectral algorithm to cluster the given point into two clusters A and B. Now in case of large overlap cases, we expect some nodes could be moved from one side of the cluster to the other while lowering the objective function. So to identify which nodes can be moved we define linkage l as similarity measure between a point and a cluster as $l(u, A) = \frac{W(u,A)}{W(A,A)}$ and similarly we can define $l(u, A) = \frac{W(u,A)}{W(A,A)}$. Now we define $\Delta l(u) = L(u, A) - L(u, B)$ if u is in A and if this function is negative then it makes sense that u should be in cluster B similarly we define $\Delta l(u) = L(u, B) - L(u, A)$ if u is in B. So we calculate Δl, then sort all the entries then start moving the point and we keep doing it as long as it improves the objective function.

   This algorithm can be used to imporve the objective function of any clustering.

4. Kannan: A partition $(C_1, C_2, \ldots, C_k)$ of V is called a (α, ϵ) clustering if every cluster C_i has conductance at least α and the inter-cluster weights are at most ϵ of the total weights of the edges.

   We try to minimize ϵ while improving α and that's how we get things like conductance of partition and Ncut criterion.

   Define $\pi_i^\infty = \frac{d_i}{Vol(v_i)}$, $P^T\pi^\infty = \pi^\infty$ therefore π^∞ is the stationary distribution of the markov chain with probability matrix P.

   Now define $\phi(A) = \frac{\sum_{i\in A, j\in\overline{A}} \pi_i^\infty P_{ij}}{min(\pi^\infty(A), \pi^\infty(\overline{A}))} = \frac{\sum_{i\in A, j\in\overline{A}} W_{ij}}{min(Vol(A), Vol(\overline{A}))}$.

Theorem 3: Suppose P is the stochastic matrix of S. If v is the 2^{nd} largest eigenvector of P with eigenvalue λ_2. W.L.O.G $v_1 \leq v_2 \ldots \leq v_n$, Then

$\min_{A \subseteq \{1,2,\ldots,n\}} \frac{\sum_{i \in A, j \in \overline{A}} \pi_i^{\infty} P_{ij}}{min(\pi^{\infty}(A), \pi^{\infty}(\overline{A}))} \geq 1 - \lambda_2 \geq \frac{1}{2} \left(min_{l, 1 \leq l \leq n} \frac{\sum_{1 \leq i \leq l, l+1 \leq j \leq n} \pi_i^{\infty} P_{ij}}{min(\sum_{1 \leq i \leq l} \pi_i^{\infty}, \sum_{l+1 \leq i \leq n} \pi_i^{\infty})} \right)^2$

Algorithm 4: Stochastic matrix spectral algorithm for the bipartitioning problem

Input : Similarity matrix $W \in \mathbb{R}^{n \times m}$
Compute the stochastic matrix P.
Compute the 2^{nd} largest eigenvector **F** of the stochastic matrix P.
Now sort $\{1, \ldots, n\}$ by the order of $F_i's$. And get the indexing $i_1, i_2, \ldots, i_n$ such that $F_{v_{i_1}} \leq F_{v_{i_2}} \ldots \leq F_{v_{i_n}}$.
Now get the best threshold indexing cut *i.e* to take the cut with best conductance among the cuts $C = \{(v_{i_1}, \ldots v_{i_k}) | 1 \leq k < n\}$.

This is the algorithm described in theorem. And this theorem proves Cheeger's inquality for this algorithm.

Idea : This might help us to improve our own algorithm and I will about more about this later on.

5. On performance of spectral graph partition methods: In this paper they have proved that the general spectral algorithm can be made to perform arbitrarily bad on a class of graphs called cockroach graphs and path cross tree graphs the modified spectral algorithm fails.

   Idea: We can use the theorems here to prove that our algorithm works well for cockroach graphs but that comes after we have proved some theoretical results for the proposed algorithm. And we can also check if our algorithm fails on these cases given in the paper where the spectral algorihtm and modified spectral algorithm fails.

Experiments

Algorithm 5: The unnormalized proposed algorithm (UnP)

Input : Similarity matrix $W \in \mathbb{R}^{n \times m}$
Compute the unnormalized Laplacian L.
Compute the fielder eigenvector **F** of the unnormalized laplacian L.
Now compute the EW_{ij}.
Now we keep on removing edges one by one by descending order of weights until we are left with two clusters.

This is unnormalized laplacain proposed algorithm(UnP).

Now we can also define the normalized laplacian proposed algorithm(NP) by using F as the fielder eigenvector of L_{Sym}.

Another algorithm can be to use all the Fielder eigenvectors of the Laplacian. Let them be $F^1, F^2, \ldots F^k$. Now define $EW_{ij} = (\sum_{m=1}^{k}(F_i^m - F_j^m)^2) * W_{ij}$. This gives us a new algorithm for the problem.

If we are using unnormalized Laplacain's fielder eigenvector then we call the algorithm Unnormalized Laplacain Updated Proposed algorithm(UnUP) and if we are using normalized Laplacian's fielder eigenvector then we call the algorithm Normalized Laplacian Updated Proposed algorithm(NUP).

Figure 3: Conductance of the output partition

	NUP	NP	NS	UnUP	UnP	UnS
Petersen	0.555556	1.000000	0.500000	1.000000	0.666667	0.466667
Enneahedron	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333
Enneahedron_1	0.714286	0.714286	0.375000	0.600000	0.600000	0.375000
Cockroach	0.200000	0.200000	0.230769	0.200000	0.200000	0.230769
Cockroach_1	0.166667	0.166667	0.166667	0.166667	0.166667	0.166667
Path_point	0.428571	0.428571	0.428571	0.428571	0.428571	0.411765
Two_clusters	0.076923	0.076923	0.076923	0.076923	0.076923	0.076923
Discrete_moons	0.071429	0.071429	0.081967	0.071429	0.071429	0.081967
Double_Tree	0.052632	0.052632	0.052632	0.052632	0.052632	0.052632
Double_Tree1	0.333333	0.333333	0.250000	0.333333	0.333333	0.333333

The only graph which has multiple fielder eigenvector is Petersen graph. Now if we ignore it for a while we see that UnP gives us better results than UnUP, NP and NUP. And the only case where NUP performs better than UnP is in the case where the graph is Petersen graph. Thus we decide to try to prove theoretical results for UnP. In Enneahedron_1 and Petersen graph UnS and NS out performs UnP. In Path_point graphs UnS out performs UnP.

Figure 4: Conductance of the output partition

	NUP	NP	NS	UnUP	UnP	UnS
Petersen	0.793651	1.111111	0.833333	1.111111	0.833333	0.933333
Enneahedron	0.666667	0.666667	0.666667	0.666667	0.666667	0.666667
Enneahedron_1	0.914286	0.914286	0.750000	0.872727	0.872727	0.750000
Cockroach	0.247619	0.247619	0.461538	0.247619	0.247619	0.461538
Cockroach_1	0.291667	0.291667	0.291667	0.291667	0.291667	0.291667
Path_point	0.642857	0.642857	0.642857	0.642857	0.642857	0.691765
Two_clusters	0.153846	0.153846	0.153846	0.153846	0.153846	0.153846
Discrete_moons	0.092705	0.092705	0.163934	0.092705	0.092705	0.163934
Double_Tree	0.105263	0.105263	0.105263	0.105263	0.105263	0.105263
Double_Tree1	0.369048	0.369048	0.336957	0.350282	0.350282	0.413333

If we ignore the Petersen graph for a while we see that UnP gives us better results than UnUP, NP and NUP. And the only case where NUP performs better than UnP is in the case where the graph is Petersen graph. In Enneahedron_1 and Petersen graph UnS and NS out performs UnP. In Path_point graphs UnS out performs UnP. This supports our decision to try to prove theoretical results for UnP.

After this we tried to defined edges weights as:-

- $EW_{ij} = (\frac{F_i}{d_i} - \frac{F_j}{d_j})^2 * W_{ij}$
- $EW_{ij} = (\frac{F_i}{\sqrt{d_i}} - \frac{F_j}{\sqrt{d_j}})^2 * W_{ij}$
- $EW_{ij} = (F_i \times d_i - F_j \times d_j)^2 * W_{ij}$
- $EW_{ij} = (F_i \times \sqrt{d_i} - F_j \times \sqrt{d_j})^2 * W_{ij}$

And also tried to define all these different kind of edge weights using all fielder eigenvectors. We ran experiments for F as the fielder eigenvector for normalized laplacian as well for unnormalized laplacian on all these algorithm discussed above. But none of these tweaks above gave us better results than UnP.

Algorithm 6: The Kannan Algorithm

Input : Similarity matrix $W \in \mathbb{R}^{n \times m}$
Compute the stochastic matrix $P = D^{-1}W$.
Compute the 2^{nd} largest eigenvector $\mathbf{F}$ of the stochastic matrix P.
Now sort $\{1, \ldots, n\}$ by the order of $F_i's$. And get the indexing $i_1, i_2, \ldots, i_n$ such that $F_{v_{i_1}} \leq F_{v_{i_2}} \ldots \leq F_{v_{i_n}}$.
Now get the best indexing threshold cut $i.e$ to take the cut with best conductance among the cuts $C = \{(v_{i_1}, \ldots v_{i_k}) | 1 \leq k < n\}$.

Suppose the partition given by this algorithm is C_K and the conductance of the partition is ϕ_K, the eigenvalue of $\mathbf{F}$ to be λ_2 and the conductance of the graph to be ϕ_G. Then

$$\phi_G \geq 1 - \lambda_2 \geq \frac{(\phi_K)^2}{2} \geq \frac{(\phi_G)^2}{2}$$

	Petersen	Enneahedron	Enneahedron_1	Cockroach	Cockroach_1	Path_point	Two_clusters	Discrete_moons	Double_Tree	Double_Tree1
kannan	0.466667	0.333333	0.466667	0.2	0.166667	0.411765	0.076923	0.071429	0.052632	0.266667
UnP	0.666667	0.333333	0.600000	0.2	0.166667	0.428571	0.076923	0.071429	0.052632	0.333333

Figure 5: Conductance of output partition

Observe that Kannan algorithm works better than UnP in every case.

Theorem : Our algorithm in case of complete graphs will always output $\{1, 2, \ldots i\}$ and $\{i+1, i+2, \ldots n\}$ where the condition $f_i \leq f_j$ for all $i < j$ is given.

Proof : Lemma 1 : Any of the edges removed has higher weight than any of the edges left.

Lemma 2: If $F_i < F_j < F_k$ and $i < j < k$. Then if F_i and F_k are in the same cluster then F_j is also in the same cluster as them.

Lemma 3: F_1 and F_n are in two different clusters.

Write down how we cannot remove edges from the same cluster of F_i and we will always get a thresholding partition no matter what

So suppose we know that F_1 is in cluster A and F_n is in cluster B.

Now suppose the highest F_i in cluster A is k_1 and the lowest F_i in cluster B is k_2. Then $k_1 \leq k_2$ and all the $F_i < k_1$ are in cluster A and all the $F_i > k_2$ are in cluster B. So if $k_1 < k_2$ then we can say that we will get a partition as $\{1, 2, \dots i\}$ and $\{i+1, i+2, \dots n\}$. If $k_1 = k_2$, then due to implementation detail it happens that we will always get a partition as $\{1, 2, \dots i\}$ and $\{i+1, i+2, \dots n\}$.

And so our algorithm cannot perform better than Kannan algorithm as Kannan algorithm in case of **complete graphs** as Kannan algorithm goes over all the partition of the form ($\{1, 2, \dots i\}$, $\{i+1, i+2, \dots n\}$) where as our algorithm outputs one of the cluster of this form.

So now the idea is to find some sort of ordering for the vertices such that index thresholding the ordering and taking the partition which gives us the best possible conductance.

And one such ordering is order the vertices as they get cut of from the cluster they are not in.

Suppose $v_1, v_2, \dots, v_k$ is in cluster A and $v_{k+1}, v_{k+2}, \dots, v_n$ is in cluster B by using the UnP algorithm. As we said earlier $|F_i - F_j|$ intuitively is measure of how dissimilar v_i and v_j are. Our algorithm removes the most dissimilar edge. So eventually it will disconnect the most dissimilar vertex from the complement cluster of its own cluster. So now we create two list cluster_order_A and cluster_order_B.

cluster_order_A is a order list of vertices which are in cluster A and cluster_order_B is a order list of vertices which are in cluster B. Now suppose v_i and v_j are both in cluster A and v_i has higher order than v_j in cluster_order_A then that means that our UnP algorithm disconnected v_i from cluster B earlier than v_j.

So now we get two lists cluster_order_A= $v_{i_1}, \dots v_{i_k}$ and cluster_order_B = $v_{i_{k+1}}, \dots v_{i_n}$ and given that v_{i_1} was cut of earlier than v_{i_2} and v_{i_n} was cut of earlier than $v_{i_{n-1}}$. So now we get a order list $v_{i_1}, v_{i_2}, \dots, v_{i_n}$. Where we can do index thresholding and get the best conductance cut out of all of them and output it.

And this works better than the Kannan algorithm in every case even in complete graphs it works better than the Kannan algorithm.