Graph Partitioning and Clustering

Vincenzo Bonifaci

April 27, 2016

1 Graph Partitioning

In the graph partitioning problem we want to divide a graph into two parts of given sizes n_1 and n_2. The goal is to minimize the number of edges cut by the resulting partition. This problem is NP-hard, so here we discuss some heuristics.

1.1 Kernighan-Lin Heuristic

A popular approach is the *Kernighan-Lin* heuristic, which is a local-search type algorithm. Let $e(S,T)$ be the size of a partition (S,T) (our objective function). One round of the algorithm works as follows:

1. Start with a random partition of the nodes into two sets S, T of size n_1, n_2 respectively. Initially, all nodes are unmarked.
2. As long as there is an unmarked pair $(u,v) \in S \times T$ such that swapping u with v decreases $e(S,T)$, swap u and v:
$$S \leftarrow S \setminus \{u\} \cup \{v\},$$
$$T \leftarrow T \setminus \{v\} \cup \{u\}.$$
3. When no improving unmarked pair exists, stop and return (S,T) as a candidate solution.

The algorithm repeats several of these rounds (the exact number is left to the user – say, r rounds), starting with a new random partition every time, and at the end returns the best of the candidates found.

Note that step 2. can be implemented by computing these values: $\delta_S(u)$ (the degree of u within S), $\delta_T(u)$, $\delta_S(v)$, $\delta_T(v)$, and A_{uv}. Indeed, the decrease in the cut size is exactly

$$\delta_T(u) - \delta_S(u) + \delta_S(v) - \delta_T(v) - 2A_{uv}.$$

The Kernighan-Lin appears to give good results in practice, but it has no quality guarantee and it may be quite slow. Every round we swap at most n pairs. To identify a pair we require

$$\sum_{u\in S}\sum_{v\in T}(\delta_G(u)+\delta_G(v)) = |T|\sum_u \delta_G(u) + |S|\sum_v \delta_G(v) = n_2 O(m) + n_1 O(m) = O(mn).$$

Therefore the algorithm requires time proportional to rmn^2 in total.

1.2 Spectral Partitioning

Another popular approach to partitioning uses the matrix properties of the graph Laplacian – in particular, the second smallest eigenpair. This is why this approach is called *spectral partitioning.*

Note that, if $side : V \to \{S, T\}$ is the function that assigns every node to either S or T, we can write $e(S,T) = \frac{1}{2}\sum_{i,j:\, side(i)\neq side(j)} A_{ij}$. We introduce variables

$$s_i = \begin{cases} +1 \text{ if } side(i) = S \\ -1 \text{ if } side(i) = T. \end{cases}$$

Observe that

$$\frac{1}{2}(1 - s_i) = \begin{cases} 1 \text{ if } side(i) \neq side(j) \\ 0 \text{ if } side(i) = side(j). \end{cases}$$

So we can write

$$e(S,T) = \frac{1}{4}\sum_{i\in V,\, j\in V} A_{ij}(1 - s_i s_j).$$

We can rewrite

$$\sum_{i,j} A_{ij} = \sum_i \deg(i) = \sum_i \deg(i) s_i^2 = \sum_{i,j} \deg(i)\delta_{ij} s_i s_j,$$

where $\deg(i)$ is the degree of i and δ_{ij} is the Kronecker symbol. Therefore

$$e(S,T) = \frac{1}{4}\sum_{i,j}(\deg(i)\delta_{ij} - A_{ij}) s_i s_j = \frac{1}{4}\sum_{i,j} L_{ij} s_i s_j,$$

where $L_{ij} = \deg(i)\delta_{ij} - A_{ij}$ is the (i,j)-th element of the graph Laplacian matrix. In matrix form,

$$e(S,T) = \frac{1}{4} s^\top L s.$$

In other words, our goal is to minimize $s^\top L s$ subject to $s_i \in \{-1, +1\}$ for all i, and $\sum_i s_i = n_1 - n_2$. In general this is hard! We will take the following approach: instead of ranging in the set $\{-1, +1\}^n$, we first allow the vector s to be *any vector in* $\mathbb{R}^n$ *of total length* n (we will later see how to get back to the original domain). That is, we allow $s \in \mathbb{R}^n$, but we require

$$\sum_i s_i^2 = n.$$

We also still require that $\sum_i s_i = n_1 - n_2$ as before, which can also be expressed as

$$\mathbf{1}^\top s = n_1 - n_2.$$

We can solve the relaxed problem by the method of Lagrangian multipliers. The theory tells use that there exist two multipliers λ, μ' such that, for all i,

$$\frac{\partial}{\partial s_i}\left(\sum_{j,k} L_{jk}s_j s_k + \lambda(n - \sum_j s_j^2) + \mu\left((n_1 - n_2) - \sum_j s_j\right)\right) = 0.$$

This implies (if $\mu = \mu'/2$)

$$\sum_j L_{ij}s_j = \lambda s_i + \mu,$$

which in matrix notation is

$$Ls = \lambda s + \mu\mathbf{1}.$$

Recall that $\mathbf{1}$ is always an eigenvector of L, with eigenvalue 0, so multiplying both sides by $\mathbf{1}^\top$ gives

$$0 = \lambda\mathbf{1}^\top s + \mu\mathbf{1}^\top\mathbf{1} = \lambda(n_1 - n_2) + \mu n,$$

equivalent to

$$\mu = -\frac{n_1 - n_2}{n}\lambda.$$

If we define the vector

$$x = s + \frac{\mu}{\lambda}\mathbf{1} = s - \frac{n_1 - n_2}{n}\mathbf{1},$$

then the Lagrangian condition tells us that

$$Lx = L(s + \frac{\mu}{\lambda}\mathbf{1}) = Ls = \lambda s + \mu\mathbf{1} = \lambda x,$$

so λ must be an eigenvalue of L, and x its associated eigenvector! Notice also that

$$\mathbf{1}^\top x = \mathbf{1}^\top s - \frac{\mu}{\lambda}\mathbf{1}^\top\mathbf{1} = (n_1 - n_2) - \frac{n_1 - n_2}{n}n = 0,$$

so x should be orthogonal to $\mathbf{1}$ (so λ cannot be zero). Other than this, it seems that we can choose any eigenpair (λ, x) of L, but which one gives the smallest cut? Note that our objective function satisfies

$$e(S,T) = \frac{1}{4}s^\top Ls = \frac{1}{4}x^\top Lx = \frac{1}{4}\lambda x^\top x.$$

On the other hand,

$$\begin{aligned} x^\top x &= s^\top s + \frac{\mu}{\lambda}(s^\top\mathbf{1} + \mathbf{1}^\top s) + \frac{\mu^2}{\lambda^2}\mathbf{1}^\top\mathbf{1} \\ &= n - 2\frac{n_1 - n_2}{n}(n_1 - n_2) + \frac{(n_1 - n_2)^2}{n}n \\ &= 4\frac{n_1 n_2}{n}, \end{aligned}$$

and hence

$$e(S,T) = \frac{n_1 n_2}{n}\lambda.$$

Since n_1, n_2 and n are fixed, it means that the smaller the λ, the better. We know that $\lambda_1 = 0$ and $0 \leq \lambda_2 \leq \ldots \leq \lambda_n$. We cannot pick λ_1 because then x would not be orthogonal to $\mathbf{1}$. So the best choice is to pick λ_2 and its associated eigenvector, v_2. That is, we pick x proportional to v_2, but scaled to satisfy $x^\top x = 4n_1n_2/n$. This is indeed the optimal solution to the relaxed problem. In terms of the vector s,

$$s_i = x_i + \frac{n_1 - n_2}{n}.$$

However, remember that what we really wanted is to have $s_i \in \{-1, +1\}$. To do so, we try to make the product

$$s^\top (x + \frac{n_1 - n_2}{n}\mathbf{1}) = \sum_i s_i (x_i + \frac{n_1 - n_2}{n})$$

as large as possible. This is achieved by setting $s_i = +1$ for the n_1 nodes with largest x_i value and $s_i = -1$ for the other n_2 nodes. Actually, there is another solution that we should not ignore: above we assumed that $|S| = n_1$, $|T| = n_2$, but the symmetric situation is equally valid, therefore we can also assign $s_i = +1$ for the n_2 nodes with largest x_i value and $s_i = -1$ for the other n_1 nodes. Clearly, among these solutions we prefer the one with the smallest cut value. In summary, the algorithm is:

1. Compute the second eigenpair (λ_2, v_2) of the graph Laplacian.
2. Sort the elements of v_2 from largest to smallest.
3. One candidate partition puts the n_1 nodes with largest value in S and the others in T.
4. The other candidate partition puts the n_1 nodes with smallest value in S and the others in T.
5. Among the two partitions above, return the one with the lower cut value.

1.3 Partitioning without Knowing the Sizes

Above we assumed that we know the sizes (n_1, n_2) of the partition that we are looking for. However, often one does not know n_1 and n_2 in advance. In this case, it does not make sense to just minimize $e(S, T)$ – most likely we would get a single node in one of the two partitions. Instead, it is more appropriate to minimize a value normalized by the partition sizes, called the *cut ratio* or *sparsity* of the cut:

$$\sigma(S, T) = \frac{\sum_{i \in S, j \in T} A_{ij}}{|S||T|}.$$

A related quantity is the *conductance* of S:

$$\phi(S, T) = \frac{\sum_{i \in S, j \in T} A_{ij}}{\min(e(S), e(T))},$$

where $e(S)$ is the sum of the degrees of the nodes in S. The conductance of the whole graph is defined to be the lowest conductance across all partitions:

$$\phi_G = \min_{(S,T)} \phi(S, T).$$

Proposition 1.1. *The sparsity of any cut of G is at least λ_2/n, where λ_2 is the second smallest eigenvalue of the Laplacian of G.*

Proof. Consider any cut (S,T). Let $|S| = pn$, $|T| = qn$ with $p+q=1$, $p,q \geq 0$. Consider the vector x defined by $x_i = q$ if $i \in S$, $x_i = -p$ if $i \in T$. Then $\mathbf{1}^\top \cdot x = 0$. Also, if (i,j) crosses the cut, $x_i - x_j = q - (-p) = 1$, while if (i,j) does not cross the cut, $x_i - x_j = 0$. Therefore

$$\begin{aligned} x^\top L x &= x^\top D x - x^\top A x \\ &= \sum_i \deg(i) x_i^2 - 2 \sum_{(i,j)\in E} x_i x_j \\ &= \sum_{(i,j)\in E} (x_i - x_j)^2 \\ &= e(S,T). \end{aligned}$$

Also, $x^\top x = q^2 pn + p^2 qn = pqn(p+q) = pqn = |S||T|/n$. But by variational characterization of the eigenvalues, λ_2 is the minimum value of the ratio $x^\top Lx/(x^\top x)$ across all x orthogonal to $\mathbf{1}$. Therefore,

$$\frac{e(S,T)}{|S||T|/n} \geq \lambda_2.$$

□

On the other hand, there is a spectral partitioning algorithm (similar to the one we discussed before) that gives a relation between λ_2 and the conductance of the graph:

1. Compute the second eigenpair (λ_2, v_2) of the graph Laplacian.
2. Sort the elements of v_2 from largest to smallest.
3. Return the minimum conductance cut among the $n-1$ cuts given by this ordering.

We omit the proof of the following result.

Theorem 1.2 (Cheeger Inequality)**.** *The partition found by the above algorithm satisfies*

$$\phi(S,T) \leq \sqrt{\lambda_2 m/n}.$$

2 Clustering and Community Detection

2.1 Modularity

Let c_i be the class (category) or type of node i. The total number of edges running between nodes of the same type is

$$\sum_{(i,j)\in E} \delta(c_i, c_j) = \frac{1}{2} \sum_{i,j\in V} A_{ij}\delta(c_i, c_j),$$

where $\delta(c_i, c_j) = 1$ if $c_i = c_j$ and 0 otherwise. We compare this quantity with the expected number of edges between nodes if edges are placed at random. This is

$$\frac{1}{2}\sum_{i,j}\frac{k_i k_j}{2m}\delta(c_i, c_j),$$

where k_i is the degree of i. The normalized difference between the above two quantities is a measure of the assortative mixing of the network, called the *modularity* of the network:

$$Q = \frac{1}{2m}\sum_{i,j}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(c_i, c_j).$$

The modularity is a real number between -1 and $+1$ (why?). A positive modularity implies associative mixing (nodes of the same kind tend to connect more with each other); a negative modularity implies dissociative mixing (nodes of the same kind tend to connect less with each other).

2.2 Girvan-Newman

The Girvan-Newman method is a divisive hierarchical clustering algorithm. It produces a sequence of partitions of the nodes of the graph, where each partition is a refinement of the previous partition. At the end of the sequence it returns the best partition seen so far.

1. Initially, all nodes have the same type.
2. Compute the modularity. If it is higher than the best modularity computed so far, store this value (and the corresponding partition).
3. Find the edge(s) of highest betweenness.
4. Remove it (or them) from the graph. If the graph splits into multiple components, this is a level of regions in the partitioning of the graph: label the components with different types (nodes in the same component get the same type, nodes in different components get different types).
5. If edges are still present, go back to point 2.
6. At the end, return the partition with highest modularity.

2.3 Greedy Modularity Maximization

Since the Girvan-Newman method is based on recomputing the betweennesses of all edges many times, it can be quite slow. Newman proposed a faster algorithm based on agglomerative hierarchical clustering.

1. Initially, each node has a distinct type.
2. Compute the modularity. If it is higher than the best modularity computed so far, store this value (and the corresponding partition).

3. Join the pair of communities that results in the largest increase (or smallest decrease) of modularity. Update the type labels accordingly.

4. If there is still more than one type, go back to point 2.

5. At the end, return the partition with highest modularity.

The algorithm can be implemented faster than Girvan-Newman (see Newman's 2004 paper for details; implementing this algorithm and testing it on real network could make for an interesting project).

2.4 Spectral Modularity Maximization

We can apply a spectral approach to modularity maximization assuming two communities. If $s_i \in \{-1, +1\}$ is the variable indicating where node i belongs to community 1 or community 2, notice that $(s_i s_j + 1)/2$ equals 1 when i and j are in the same group and 0 otherwise, therefore $\delta(c_i, c_j) = (s_i s_j + 1)/2$. Define

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m},$$

so the modularity can be written as

$$Q = \frac{1}{4m} \sum_{i,j} B_{ij}(s_i s_j + 1) = \frac{1}{4m} \sum_{i,j} B_{ij} s_i s_j + 0.$$

In matrix terms,

$$Q = \frac{1}{4m} s^\top B s,$$

where the $n \times n$ matrix B has elements B_{ij}. B is called the *modularity matrix*.

The problem is again hard given the binary nature of the s vector. So we relax the binary constrain to allow $s \in \mathbb{R}^n$, but we keep the constraint $s^\top s = n$.

Using the technique of Lagrangian multipliers, we find there exists β such that for each i,

$$\frac{\partial}{\partial s_i} \left(\sum_{j,k} B_{jk} s_j s_k + \beta(n - \sum_j s_j^2) \right) = 0.$$

That is, $\sum_j B_{ij} s_j = \beta s_i$, which in vector form is $Bs = \beta s$. So s is an eigenvector of B. The corresponding modularity value is

$$Q = \frac{1}{4m} \beta s^\top s = \frac{n}{4m} \beta,$$

so for maximum modularity we want to have β as large as possible. In other words we want β to be the largest eigenvalue of the matrix B, and the optimal solution to the relaxed problem is its associated eigenvector u_1.

Again, we cannot really set $s = u_1$ in the original problem, because of the binary constraints $s_i \in \{-1, +1\}$. But, heuristically, we can maximize $s^\top u_1 = \sum_i s_i (u_1)_i$, i.e. we pick $s_i = +1$ if $(u_1)_i > 0$ and $s_i = -1$ if $(u_1)_i < 0$.

A potential issue with the efficiency of this method is that the B matrix is *not* sparse (differently from the Laplacian). Indeed, B can easily have $\Theta(n^2)$ entries. So, if we apply the power method blindly, we incur a high cost per iteration. But, the adjacency matrix A *is* sparse, and note that B differs from A just by a rank-1 term:

$$B = A - \frac{\boldsymbol{k}\boldsymbol{k}^\top}{2m},$$

where $\boldsymbol{k}$ is the vector of node degrees. So we can still compute a matrix-vector product Bx in $O(m+n)$ operations, by first computing the number $\boldsymbol{k}^\top x$ (in $O(n)$ time), and then computing

$$Bx = Ax - \frac{\boldsymbol{k}(\boldsymbol{k}^\top x)}{2m}$$

for a cost of $O(m+n)$. Using the power method, we can then compute u_1 with a similar running time as for the usual computation of the largest eigenpair of a sparse matrix.