Modularity-Maximizing Graph Communities via Mathematical Programming

Gaurav Agarwal and David Kempe

大綱

介紹兩個Modularity最大化的演算法

1. 轉換成Linear Programming

2. 轉換成Vector Programming

先備知識 Modularity

$$Q(\mathcal{C}) := \frac{1}{2m}\sum_{u,v}\left(a_{u,v} - \frac{d_u d_v}{2m}\right) \cdot \delta(\gamma(u), \gamma(v))$$

演算法 Intro

our approach is to aim for the best division at each level individually, requiring a partition into two clusters at each level. Clusters are recursively subdivided as long as an improvement is possible.

演算法 原理

Quadratic Programming is NP-complete. Hence, we use the standard technique of relaxing the QP to a corresponding Vector Program (VP), which in turn can be solved in polynomial time using semi-definite programming (SDP).

演算法 Step 1. Replacing

$$\frac{1}{2m}\sum_{u,v}\left(a_{u,v}-\frac{d_u d_v}{2m}\right)\cdot\delta(\gamma(u),\gamma(v))$$

替換成 $m_{u,v}$

$$\frac{1}{4m}\sum_{u,v} m_{u,v} \cdot (1 + y_u y_v)$$

For every vertex v, we have a variable y_v which is 1 or -1 depending on whether the vertex is in one or the other partition.

$$\text{Maximize } \frac{1}{4m}\sum_{u,v} m_{u,v}\cdot(1+y_u y_v)$$
$$\text{subject to } y_v^2 = 1 \text{ for all } v.$$

演算法 QP→VP

To turn a quadratic program into a vector program, one replaces each variable y_v with a (n-dimensional) vector-valued variable $\mathbf{y}_v$, and each product $y_u y_v$ with the inner product $\mathbf{y}_u \cdot \mathbf{y}_v$

$$\text{Maximize } \frac{1}{4m}\sum_{u,v} m_{u,v}\cdot(1+y_u y_v)$$
$$\text{subject to } y_v^2 = 1 \text{ for all } v.$$

演算法 QP→VP

$\mathbf{y}_v$: (n-dimensional)

To turn a quadratic program into a vector program, one replaces each variable y_v with a (n-dimensional) vector-valued variable $\mathbf{y}_v$, and each product $y_u y_v$ with the inner product $\mathbf{y}_u \cdot \mathbf{y}_v$

$$\text{Maximize } \frac{1}{4m} \sum_{u,v} m_{u,v} \cdot (1 + \mathbf{y}_u \cdot \mathbf{y}_v)$$
$$\text{subject to } \mathbf{y}_v^2 = 1 \quad \text{for all } v.$$

演算法 solving SDP

We use the standard process

1. for transforming the VP formulation to the SDP formulation
2. for obtaining back the solution to the VP from the solution to SDP.

The result of solution of VP will be vectors y_v for all vertices v

For solving the SDP problems in our experiments, we use a standard off-the-shelf solver CSDP

演算法 Step 2. Rounding

To obtain a partition from the node locations y_v, we use a rounding procedure for the Max-Cut problem.

$$\text{Maximize } \tfrac{1}{2}\sum_{(u,v)\in E}(1 - y_u y_v)$$
$$\text{subject to } y_v^2 = 1 \quad \text{for all } v.$$

演算法 Step 2. Rounding

$$\text{Maximize } \frac{1}{4m}\sum_{u,v} m_{u,v}\cdot(1+y_u\cdot y_v)$$

$$\boxed{\text{subject to } y_v^2 = 1 \quad \text{for all } v.}$$

All y_v on n-dimensional hypersphere

Find a hyperplane pass through origin to cut the hypersphere

演算法 Step 2. Rounding

1. All y_v on n-dimensional hypersphere

The constraint for all v ensures that all nodes are embedded at distance 1 from the origin.

e.g. (1, 0, 0)、(3/5, 4/5, 0)、$\left(\frac{\sqrt{3}}{3}, \frac{\sqrt{3}}{3}, \frac{\sqrt{3}}{3}\right)$

演算法 Step 2. Rounding

$$\text{Maximize } \frac{1}{2}\sum_{(u,v)\in E}(1 - y_u y_v)$$
$$\text{subject to } y_v^2 = 1 \quad \text{for all } v.$$

2. Find a hyperplane pass through origin to cut the hypersphere

Find vector S, which is an n-dimensional vector, each of whose components is an independent N (0, 1) Gaussian.

→ **choose random hyperplanes and retain the best resulting partition.**

$$S := \{v \mid \mathbf{y}_v \cdot \mathbf{s} \geq 0\} \text{ and } \overline{S} := \{v \mid \mathbf{y}_v \cdot \mathbf{s} < 0\}$$

演算法 Step 2. Rounding

After this 2 steps, we obtain two clusters C’ and C” , then we calculate the modularity. The modularity Q increases by:

$$\Delta Q(C) = \frac{1}{m}\left(\frac{(\sum_{v\in C'} d_v)(\sum_{u\in C''} d_u)}{2m} - |e(C', C'')|\right)$$

Algorithm 2 Hierarchical Clustering

Let M be an empty Max-Heap.
Let C be a cluster containing all the vertices.
Use VP rounding to calculate (approximately) the maximum increase in modularity possible, $\Delta Q(C)$, achievable by dividing C into two partitions.
Add $(C, \Delta Q(C))$ to M.
while the head element in M has $\Delta Q(C) > 0$ **do**
    Let C be the head of M.
    Use VP rounding to split C into two partitions C', C'', and calculate $\Delta Q(C'), \Delta Q(C'')$.
    Remove C from M.
    Add $(C', \Delta Q(C')), (C'', \Delta Q(C''))$ to M.
end while
Output as the final partitioning all the partitions remaining in the heap M, as well as the hierarchy produced.