Efficient ΔQ

Geet Duggal

Spring 2007ish

One of our concerns is being able to calculate the modularity efficiently on each Monte Carlo move[1]. Modularity can be conveniently expressed as [1]:

$$Q = \sum_{i \in C} (e_{ii} - a_i^2) \tag{1}$$

where C is the set of communities This essentially amounts to being able to efficiently keep track changes in these two variables:

$$e_{ii} = \frac{1}{2m} \sum_{vw} A_{vw} \delta(c_v, i) \delta(c_w, i) \tag{2}$$

$$a_i = \frac{1}{2m} \sum_{v} k_v \delta(c_v, i) \tag{3}$$

reminding ourselves that m is the number of edges in the undirected graph, c_v indicates vertex v belongs to community c_v, and A_{vw} is the adjacency matrix. Stated in simpler terms, modularity is capturing the total fraction of within-community edges subtracted from what we would expect from a random graph for that community *for all* communities.

Given that changes in modularity (ΔQ) in the Metropolis Monte Carlo scheme described in [2] involve changing the community assigned to a random node, the change in modularity is only caused by that node and the two communities it affects. Aside from the case where the node does not change its community, we can explore how to efficiently recalculate the modularity from the previous value.

Suppose the two communities in the comparison are i and j, where we move v' from i to j:

1. the change in equation (2) involves subtracting the number of edges from i associated with v' ($E_{iv'}$) and adding the number of edges to j that are now newly associated with v', ($E_{jv'}$).

[1] This first revision also includes a C implementation that calculates the modularity given an edge and partition file: `./Q graph.edg nodes.par`. The partition file simply associates a node with a community ID in the same format of the edge file.

2. the change in equation (3) involves properly subtracting $k_{v'}$ from community i's square term and adding it to community j's square term.

Since item 1 has no ugly squares:

$$\Delta e = e_{ii}(t+1) - e_{ii}(t) = \frac{1}{m}(E_{jv'} - E_{iv'})$$

But for item 2, we have:

$$\begin{aligned}
a_i(t) &= \frac{1}{2m}\sum_v k_v \delta(c_v, i) \\
a_j(t) &= \frac{1}{2m}\sum_v k_v \delta(c_v, j) \\
a_i(t+1) &= a_i(t) - \frac{1}{2m}k_{v'} \\
a_j(t+1) &= a_j(t) + \frac{1}{2m}k_{v'} \\
\Delta a^2 &= \left[[a_j(t+1)]^2 + [a_i(t+1)]^2\right] - \left[[a_j(t)]^2 + [a_i(t)]^2\right] \\
&= \left[\left(a_j(t) + \frac{1}{2m}k_{v'}\right)^2 + \left(a_i(t) - \frac{1}{2m}k_{v'}\right)^2\right] - \left[[(a_j(t)]^2 + [a_i(t)]^2\right] \\
&= \left[\frac{1}{4m^2}k_{v'}^2 + \frac{1}{m}a_j(t)k_{v'} + \frac{1}{4m^2}k_{v'}^2 - \frac{1}{m}a_i(t)k_{v'}\right] \\
&= \frac{k_{v'}}{m}\left(\frac{k_{v'}}{2m} + a_j(t) - a_i(t)\right)
\end{aligned}$$

Thus, after some basic algebra, we can express our change in modularity as a simple operation by only storing a few running sums in memory:

$$\Delta Q = \frac{1}{m}\left[(E_{jv'} - E_{iv'}) - k_{v'}\left(\frac{k_{v'}}{2m} + a_j(t) - a_i(t)\right)\right] \tag{4}$$

recalling that we are transferring v' from community i to j. $\diamond$

References

[1] Aaron Clauset, M.E.J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70:066111, 2004.

[2] C.P. Massen and J.P.K. Doye. Thermodynamics of community structure. *Preprint arXiv:cond-mat/0610077v1*, 2007.