Design Lab- CS59001

Name :: Kanishk Singh   Rollno :: 17CS30018

Introduction

Implementation of Belief-Propagation algorithm for Baseline Evaluation of MultiLayer Louvain Algorithm over LFR Benchmarks.

Algorithm : The algorithm attempts to solve for the marginals of each node, P(ti = q), in the finite temperature regime instead of searching for a global modularity maximum. By looking for a ``consensus of good partitions'''' rather than seeking a single ``best'''' partition, the algorithm converges to nontrivial structures above a certain temperature only if there is a broad underlying structure within the network. The belief propagation works by deriving update conditions for the node beliefs in terms of the message from node ***i*** to ***k*** that helps k determine what node k “believes” its own community to be. After computing marginals for each node, the nodes in the network are assigned to the community according to its greatest marginal with randomly broken ties.

Github repo-link ::

Directory Structure:

- **mixmod_wt :**
    - modularity.py
    - Mixmod_wt_correctingimplementation_without_half.py
- **newnetsForcomparingBaseline**
    - _networks
        - Network_alpha_p_mu_p1_p2
- **modbp**
    - GenerateGraphs.py
    - ModularityBP.py
    - bp.py
    - main.py
    - Q_modularity.ipynb
    - modb.ipynb

Implementation Details

(A) ModularityBP.py

This file contains the code for the implementation of the multilayer belief propagation algorithm. The class and helper methods are described below :

- **class ModularityBP():**
    - input_parameters
        - mlgraph - object for containing the multilayer-graph.
        - layer_vec - layer membership of individual nodes(0-indexed)
        - interlayer_edgelist - takes a list of edges across layers. For eg., - [[1,189],[46,125]] denotes the edge between members of two different layers.
        - intralayer_edgelist - takes a list of edges across layers. For eg., - [[1,18],[146,125]] denotes the edge between members of two same layers.
    - variables
        - marginals
        - Partitions - argmax(marginals), contains the community detected after running the belief-propagation algorithm
        - niters - number of message iterations before it reaches convergence

        - Retrieval_modularities -contains modularity index and metadata after obtaining the communities across members.
    - Methods
        - runmodbp
            - beta - The inverse temperature parameter at which to run the modularity belief propagation algorithm. Must be specified each time BP is run.
            - q - The number of mariginals used for the run of modbp.
            - niter - Maximum number of iterations allowed.
            - resgamma - resolution parameter at which to run the algorithm. [Default - 1.0]
            - omega - The coupling strength used in running the algorithm. This represents how strongly the algorithm tries to assign nodes connected by an interlayer connection to the same community.
            - helper methods for running one pass of belief propagation update
        - calc_modularity
            - calculates the modularity of graph for given partition, resolution, and omega

(B) Q_modularity.ipynb and modbp.ipynb

(a) Methods :

(i) **parser –** this reads the LFR benchmarks and extracts layer membership, intra and inter layer edges to be given to the multilayer belief-propagation algorithm This generates a dictionary for teh LFR Benchmarks and stores them in a pkl.

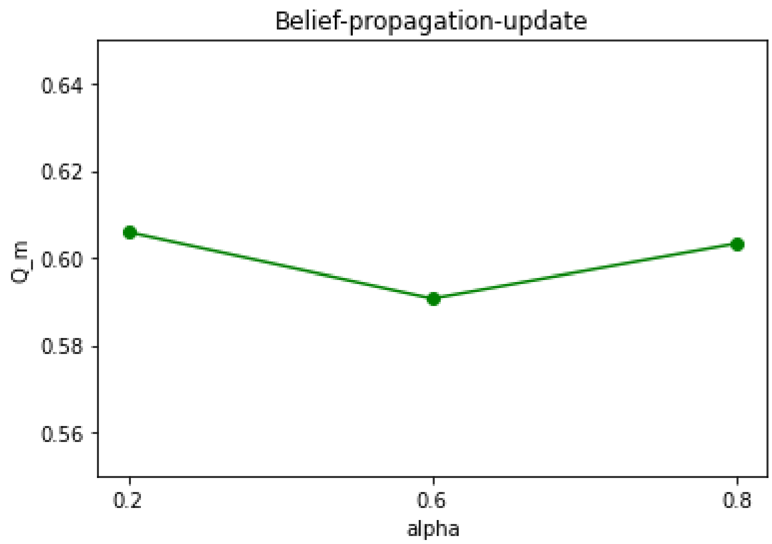
(ii) **community –** this takes in the pkl file and runs the belief-propagation algorithm to generate the community membership for each-node and stores them in a pkl file.

(iii) **result -** this file takes in the generated communities and the original LFR benchmarks to compute the Q_M modularity for the communities detected by the baseline belief-propagation algorithm.

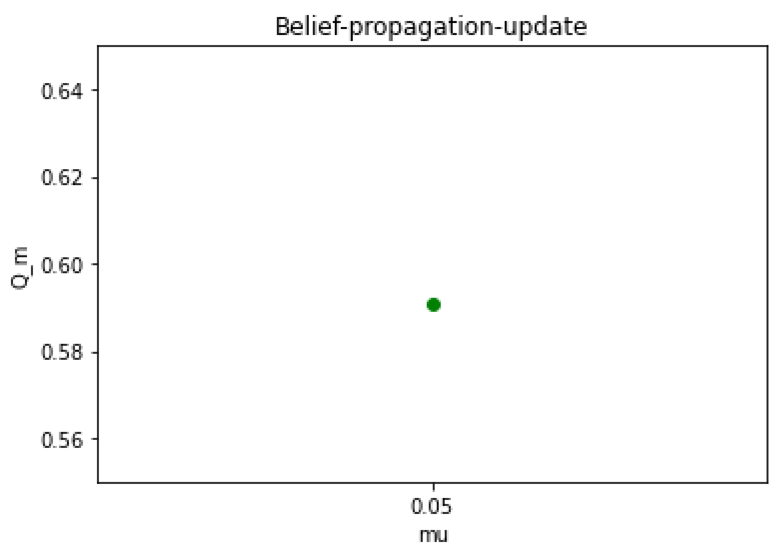
(iv) **draw_plot -** this takes in the generated modularities and groups the plots with varying alpha, mu, p, p1and p2.

Experiments and results :

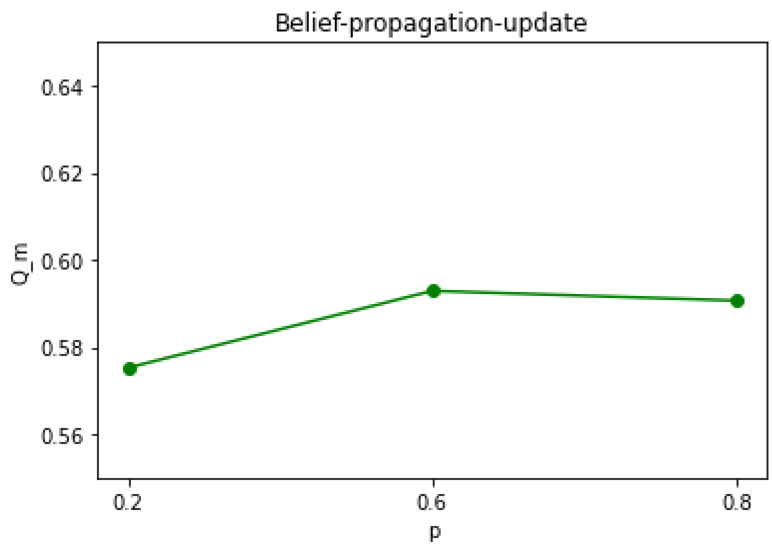
(a) Varying α with μ = 0.05, p = 0.8, p1=0.8 and p2 = 0.1



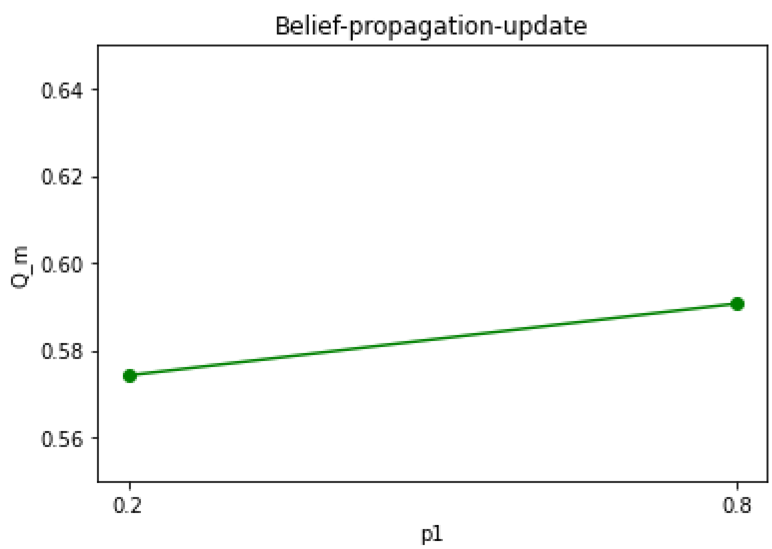
(b) Varying μ with α= 0.6, p = 0.8, p1=0.8 and p2 = 0.1



(c) Varying p with μ= 0.05, α= 0.6, p1=0.8 and p2 = 0.1



(d) Varying p1 with α= 0.6, p = 0.8, μ= 0.05 and p2 = 0.1



(e) Varying p2 with α= 0.6, p = 0.8, p1=0.8 and μ= 0.05