Discovering soft maximal cliques on weighted graphs with denoised co-occurence consistency measures, MapReduce and GrowShrink

Advait

February 2024

We will first write a baseline algorithm using MapReduce principles to find soft maximal cliques as defined for a weighted graph using lattice structures. We can relate this algorithmic thinking to the GrowShrink algorithm and seed considerations.

1 Soft maximum cliques discovery, Algorithm 1

D is dictionary which stores sub-graph coherence scores for a given clique. D will have partitions based on sub-graph size for reference efficiency. Procedure C $find-subgraph-coherence$ references D for a given sub-graph and returns the coherence value, else computes the coherence score and returns the coherence value. We avoid duplicate computations. We may need D for thresholding and analysis.

We will traverse a lattice structure L corresponding to nodes in a weighted graph G. We elaborate an iterative algorithm starting with data for cliques of size k.

E.g. Consider (a), (b),(c),(d),(e) as nodes in G. Consider k=2 and set S_2 having cliques (a,b), (a,c),(a,e), (b,c) etc. Consider D to be populated for these cliques. Consider a mapping CS_2 which maps an element in S_2 to its candidate list. Since we operate with key-value pairs in map, S_2 can be of the form (x,C(x)) for each x which is a clique of size 2 where C(x) is the computed coherence score referenced in D. Initalized globally, $max-clique-list[k]$ will store maximal cliques of size k.

Function $map-block(S_k,\ CS_k)$:

returns $S_{k+1}, CS_{k+1}, P_{k,k+1}, P_{k+1,k}$

elaborating what happens to each key element x in S_k, we will call $map-func(S_k, CS_k, f)$ in practise.

each node n in $CS_2(x)$ to is added to x to create some y(x + n) of size k+1.

$CS_{k+1}(y)$ will contain $CS_k(x) - n$. Adjust for thresholding.

$(x)->(y)$ will be added in pairing $P_{k,k+1}[x]$.

$(y)->(x)$ will be added in paring $P_{k+1,k}[y]$.
Since in our algorithm, we can remove the node of least lnc to find the maximally coherent down neighbor, we do not need to maintain this pairing in practice.

$(y, C(y))$ will be added to S_{k+1}.

Avoids redundant computations. Map block code is vectorized. The procedure to obtain to $S_{k+1}, P_{k,k+1}$ can itself consist of map and reduce functions.

Function $reduce - block(P_{k,k+1}, S_{k+1})$:

returns $Grow_k$

elaborating what happens to each key element x in $P_{k,k+1}$, reduce will aggregate according to the keys of $P_{k,k+1}$, which denote clusters of size k and their corresponding one node additional cluster list, will collapse this list to a single value corresponding to the up-neighbour of maximum coherence.

Shuffle according to keys x of $P_{k,k+1}$.

$Grow_k[x]$ is assigned j such that $S_{k+1}[i] \leq S_{k+1}[j]$ for all i in $P_{k,k+1}[x]$

Function $filter - block(S_k, Grow_k, Shrink_k)$:

returns $max - clique - list$ of sub-graphs of size k.

In view of a map reduce paradigm, the filter function can be implemented in terms of map function 2, reduce function 2. For each key x (keys are common to $S_k, Grow_k, Shrink_k$, we will compare $S_k(x)$ with $C(Grow_k(x))$ and $C(Shrink_k((x))$ and return true for $S_k(x)$ if it is greater than both. We can aggregate the keys corresponding to the true values (reverse dictionary or streamlined functions) and return the maximal clique list of size k.

After intialization inside the main alogrithm, adding convergence criterion.

Algorithm $find - maximal - cliques(k, S_k, CS_k)$

$S_{k+1}, CS_{k+1}, P_{k,k+1}, P_{k+1,k} = map - block(S_k,\ CS_k)$

$Grow_k = reduce - block(P_{k,k+1}, S_{k+1})$

$Shrink_k = shrink - func(S_k)$

$max - clique - list[k] = filter - block(S_k, Grow_k, Shrink_k)$

$find - maximal - cliques(k + 1, S_{k+1}, CS_{k+1})$