DyPerm: Maximizing Permanence for Dynamic Community Detection (Supplementary Materials)

1 Methodology

Proposition 1. *The order in which the edges (both intra- and inter-community) associated with a node are inserted, is immaterial for permanence maximization.*

Proof. Let us assume that node u belongs to community C_u. There are two types of edges associated with u: (i) intra-community edges whose both end nodes are present inside C_u, and (ii) inter-community edges whose one end node is inside C_u, other is outside C_u. In the case of adding an intra-community edge, $I(u)$ and $d(u)$ increase by 1. Since, the increment is the same, we say that the order in which intra-community edges of a node are added does not matter. In the case of adding an inter-community edge, the value of permanence decreases or remains same depending upon the change in $E_{max}(u)$. But, as shown in Algorithm 5 of the main text, $Perm(u)$ is checked after moving it to v's community and after moving its neighbors, one by one, till the permanence value keeps increasing. The same steps are followed for v, after moving it to u's community. The order in which the edges are inserted does not matter because in any case, the algorithm will displace the nodes and their neighbors so as to maximize the permanence value of each community, separately.

Proposition 2. *The order in which the edges (both intra- and inter-community) associated with a node are deleted, is immaterial for permanence maximization.*

Proof. In the case of deleting intra-community edges, the order is immaterial because with the deletion of every internal edge connected with node u, both $I(u)$ and $d(u)$ decrease by one. In the case of deleting inter-community edges, the order of deletion is immaterial. For every edge to an external neighbor of a node u that is deleted, $d(u)$ will decrease and $E_{max}(u)$ may decrease or remains same. $C_{in}(u)$ remains unchanged. Therefore, $Perm(u)$ increases. So each deletion is favorable for u. Even if the order of deletion is changed, the resultant permanence value will increase.

Proposition 3. *If C is a community in the current snapshot of $\mathcal{G}$, then adding any intra-community edge to C does not split it into smaller communities.*

Proof. We prove this by showing that the contribution of u in C is more than that when C splits. Permanence of u after edge addition is: $Perm^C(u) = [\frac{I(u)+1}{E_{max}(u)} \times \frac{1}{d(u)+1}] - [1 - C_{in}(u)]$. Since permanence is a vertex-centric metric, we have to look at the effect it has on neighboring communities of u as well. Let both u and v belong to community C. If C splits into two smaller communities, one containing u and the other containing v, the new $E_{max}(u)$, i.e.,$E^{'}_{max}(u)$ ranges from $E_{max}(u)$ to $I(u)$ since it can be equal to, at most, the number of internal neighbors of u. Also, the new number of internal neighbors of u, i.e., $I^{'}(u)$ ranges from 0 (where only u gets disconnected and forms a new community) to $I(u) - 1$ (when just one neighbor of u goes away). Therefore, it further generates 4 subcases:

Case C.1.1: $E^{'}_{max}(u) = E_{max}(u)$ **and** $I^{'}(u) = 0$. In this case, the modified value of $Perm^C(u)$, i.e., $Perm^{'C}(u) = 0 - [1 - C_{in}(u)] \leq 0$. As per the definition of Permanence of a vertex, it cannot be negative. Therefore, this case cannot occur.

Case C.1.2: $E^{'}_{max}(u) = E_{max}(u)$ **and** $I^{'}(u) = I(u) - 1$. In this case, $Perm^{'C}(u) = [\frac{I(u)-1}{E_{max}(u)} \times \frac{1}{d(u)+1}] - [1 - C_{in}(u)] < Perm^C(u)$. This case does not occur because it does not result in Permanence getting maximized.

Case C.1.3: $E^{'}_{max}(u) = I(u)$ **and** $I^{'}(u) = 0$. In this case, $Perm^{'C}(u) = 0 - [1 - C_{in}(u)] < 0$. As per the definition of Permanence of a vertex, it cannot be negative. Therefore, this case is not possible.

Case C.1.4: $E^{'}_{max}(u) = I(u)$ **and** $I^{'}(u) = I(u) - 1$. In this case, $Perm^{'C}(u) = \frac{I(u)-1}{I(u)} \frac{1}{d(u)+1} - [1 - C_{in}(u)] < Perm^C(u)$, since $I(u) > E_{max}(u)$. This case does not occur because it does not result in Permanence getting maximized.

Therefore, the current network structure remains intact in this case.

Proposition 4. *Deleting an intra-community edge between nodes u and v decreases the permanence value of the two nodes.*

Proof. Let $P_a(u) = \frac{I(u)}{E_{max}(u)} \frac{1}{d(u)}$ and $P_b(u) = 1 - C_{in}(u)$.
Let's suppose that an edge connecting nodes u and v, which are both members of a community C, is removed.
For u, let $Perm^C(u)$ be the permanence before deleting the edge and let $Perm^{'C}(u)$ be its permanence afterwards. For the proposition to be true, we have to prove, $Perm^C(u) - Perm^{'C}(u) < 0$ i.e., $\Delta Perm^C(u) < 0$.
$\Delta Perm^C(u) = \frac{I(u)}{d(u)} - \frac{I(u)-1}{d(u)-1} + \frac{x+I(u)-1}{I(u)(I(u)-1)} - \frac{I(u)-1}{(I(u)-1)(I(u)-2)}$.
Simplified further, $\frac{xI(u)-2x+2-2I(u)}{I(u)(I(u)-1)(I(u)-2)} - \frac{d(u)-I(u)}{d(u)(d(u)-1)} > 0$. Here, x represents the number of links between u and its other internal neighbors, apart from v.
The denominator of the above expression is greater than 0 as all terms are positive, so we solve for the numerator we get,
$d(u)(d(u)-1)(xI(u) - 2x + 2 - 2I(u)) - I(u)(d(u) - I(u))(I(u)-1)(I(u)-2)$.
On solving further, we get
$I(u)^4 + I(u)^3(3 - d(u)) + I(u)^2(3d(u) - 2) + I(u)(xd(u)(d(u)-1) - 2d(u)^2) + 2d(u)(d(u) - xd(u) + x - 1) > 0$.
Now, $I(u)^4$ is always positive.
$I(u)^3(3 - d(u)) > 0$ if $d(u) < 3$.
$I(u)^2(3d(u) - 2) > 0$ if $d(u) \neq 0$.
$I(u)(xd(u)(d(u)-1) - 2d(u)^2) > 0$, if $d(u) = 0$.

$2d(u)(d(u) - xd(u) + x - 1) > 0$, if $d(u) = 0, 1$.
From the above derivation we see that there is no value of d(u) for which all the expressions are positive. Hence, $Perm^C(u) - Perm'^C(u) > 0$.

2 Time complexity of DyPerm

This algorithm consists of 4 subroutines i.e., $NodeAddition$, $NodeDeletion$, $EdgeAddition$ and $EdgeDeletion$. $NodeAddition$ and $NodeDeletion$ are represented in the form of $EdgeAddition$ and $EdgeDeletion$ respectively. Therefore, the time complexity of **DyPerm** will be the maximum of $EdgeAddition$ and $EdgeDeletion$ subroutines and we compute the time complexity of the same.

Time complexity of EdgeAddition: $EdgeAddition$ is further consisted of 2 subroutines based on whether edge is intra-community ($IntraEdgeAddition$) and inter-community ($InterEdgeAddition$). Therefore, the time complexity of $EdgeAddition$ will be the maximum of time complexity of $IntraEdgeAddition$ and $InterEdgeAddition$. $IntraEdgeAddition$ is constant in time i.e., runs in $\mathcal{O}(1)$ as there is no change in the community structure. $InterEdgeAddition$ is similar to breadth first search using an adjacency list representation for the underlying graph. For calculating the permanence value for a node u, finding the number of internal neighbors and the degree of the node takes constant time, c. Further, if there are k communities in the network then computing $E_{max}(u)$ takes $\mathcal{O}(k)$. Computing the old and new permanence values for the two communities involved takes $\mathcal{O}(n_C(c + k))$, where n_C denotes the number of nodes in a community C . Iterating over the internal neighbors of a node u and calculating permanence twice for each neighbor takes $\mathcal{O}(V + E(c + k))$ which essentially becomes $\mathcal{O}(E)$. The time complexity of $EdgeAddition$ becomes $max(\mathcal{O}(1), \mathcal{O}(E))$ i.e., $\mathcal{O}(E)$.

Time complexity of EdgeDeletion: $EdgeDeletion$ further consists of 2 subroutines based on whether edge is intra-community ($IntraEdgeDeletion$) and inter-community ($InterEdgeDeletion$). Therefore, the time complexity of $EdgeDeletion$ will be the maximum of time complexity of $IntraEdgeDeletion$ and $InterEdgeDeletion$. $InterEdgeDeletion$ is constant in time i.e., runs in $\mathcal{O}(1)$ as there is no change in the community structure. $IntraEdgeAddition$ is similar to breadth first search using an adjacency list representation for the underlying graph. For calculating the permanence value for a node u, finding the number of internal neighbors and the degree of the node takes constant time, c. Further, if there are k communities in the network then computing $E_{max}(u)$ takes $\mathcal{O}(k)$. Computing the old and the new permanence values for the two communities involved takes $\mathcal{O}(n_C(c + k))$, where n_C denotes the number of nodes in a community C. Iterating over the internal neighbors of a node u and calculating permanence twice for each neighbor takes $\mathcal{O}(V + E(c + k))$ which essentially becomes $\mathcal{O}(E)$. The time complexity of $EdgeDeletion$ becomes $max(\mathcal{O}(1), \mathcal{O}(E))$ i.e., $\mathcal{O}(E)$. Therefore, the total time complexity of **DyPerm** becomes $\mathcal{O}(E)$.

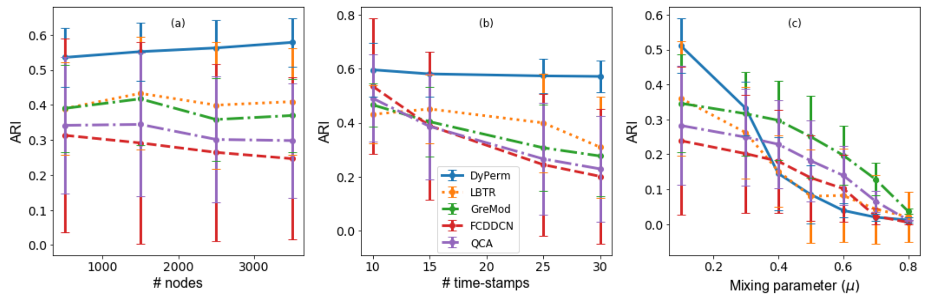

Fig. 1. Accuracy (average ARI and its standard deviation across different time-stamps for each network) of the competing methods with the change of LFR parameters for experimental setup I.

3 Experimental Results

Figure 1 shows the ARI value (and its standard deviation) of the competing methods with the change in different LFR parameters. Once again, we observe similar pattern – DyPerm significantly outperforms all other baseline methods. DyPerm beats LBTR by 46.2%, 46.3%, and QCA by 36.58%, 38.1% with the increase of the number of nodes and time-stamps respectively, averaged over all the time-stamps. These improvements are significant according to the t-test with 95% confidence interval. All algorithms perform quite low with respect to μ. In the case of μ parameter, DyPerm beats LBTR by 28.5% and QCA by 41.3% till $\mu = 0.3$ and beyond that, all algorithms except LBTR performs low. But on an average, DyPerm beats all other algorithms. Figure 2 shows the ARI

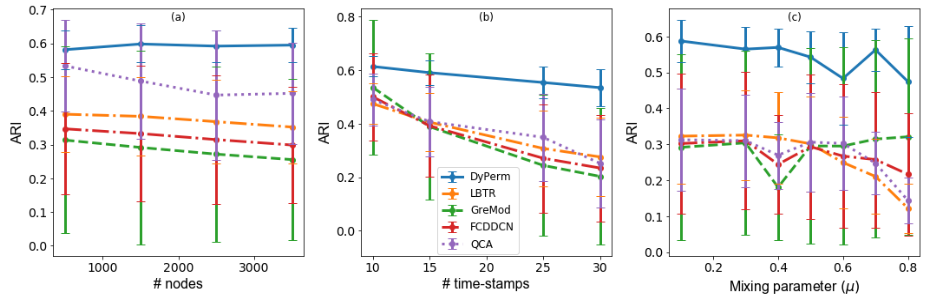

Fig. 2. Accuracy (average ARI and its standard deviation across different time-stamps for each network) of the competing methods with the change of LFR parameters for experimental setup II.

value (and its standard deviation) of the competing methods with the change in different LFR parameters for experimental setup II. Once again, we observe similar pattern – DyPerm significantly outperforms all other baseline methods. DyPerm beats LBTR by 27.95%, 32.3%, 45.55% and QCA by 101.3%, 30.76%, 20.76% with the increase of the number of nodes, time-stamps, and μ respectively, averaged over all the time-stamps. These improvements are significant according to the t-test with 95% confidence interval. Apart from DyPerm, all other algorithms perform quite poor with respect to μ.