Some simulations for the algorithm

Chanwoo Lee
April 8, 2021

1 Simulation setting

I perform 4 different simulations as follows

1. **pb_smooth (from current `pbtensor`):** Generate the sorted symmetric probability tensor $\mathcal{W}$ and generate adjacency $\mathcal{A}$ from stochastic block model. Generated $\mathcal{A}$ is symmetric and diagonal entries are 0.
2. **pb_non_smooth (from previous `pbtensor`):** Generate the unsorted symmetric probability tensor $\mathcal{W}$ and generate adjacency $\mathcal{A}$ from stochastic block model. Generated $\mathcal{A}$ is symmetric and diagonal entries are 0.
3. **sn_smooth:** Generate the sorted signal tensor Θ ranging from -10 to 10 and generate observed tensor $\mathcal{Y} = \Theta + \mathcal{E}$ where $\mathcal{E}$ is i.i.d. Gaussian noise with $\sigma^2 = 1$.
4. **sn_non_smooth** Generate the unsorted signal tensor Θ ranging from -10 to 10 and generate observed tensor $\mathcal{Y} = \Theta + \mathcal{E}$ where $\mathcal{E}$ is i.i.d. Gaussian noise with $\sigma^2 = 1$.

Figure 1 shows the signal tensors corresponding to each simulation setting when the number of node is 50. I perform simulations for different $n \in \{50, 100, \ldots, 250\}$ with a fixed signal tensor size $\Theta \in \mathbb{R}^{20\times 20\times 20}$.

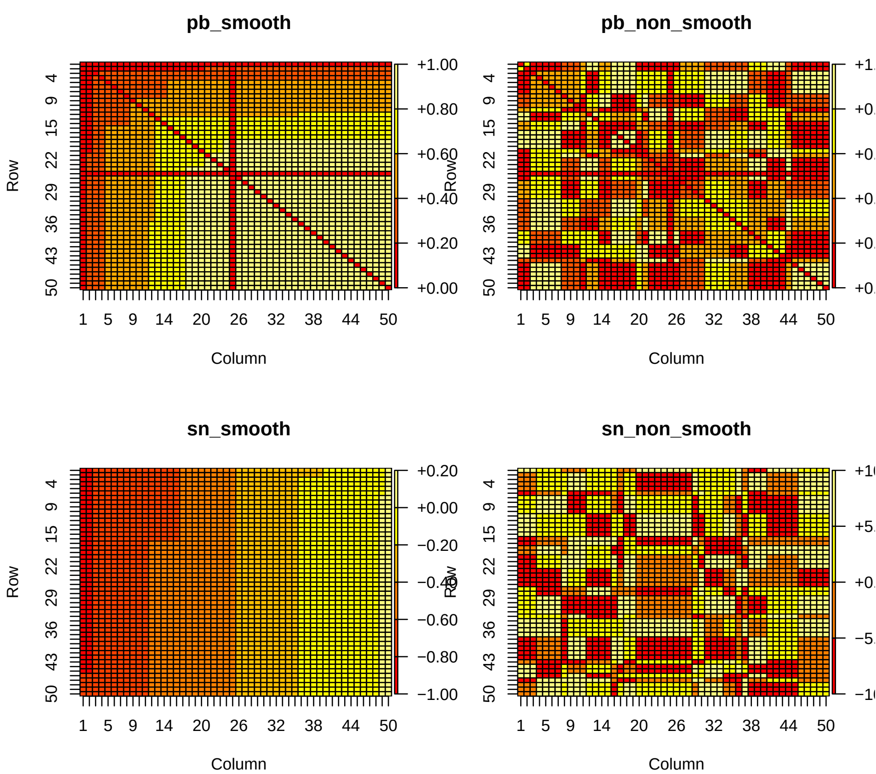


Figure 1: Signal tensors of four different simulations.

4-step:
1. update clustering at M1
2. M2
3. M3
4. core —> estimated Y;

2 Alternative algorithm

Current clustering group updates are based on

$$\hat{\boldsymbol{M}}_k(a) = \arg\min_{r\in[r_k]} \left\| \left(\mathcal{M}_k(\mathcal{Y}_k^{(t)})\right)_{j:} - \left(\mathcal{M}_k(\mathcal{S}^{(t)})\right)_{a:} \right\|_2^2,$$

where $\mathcal{S}^{(t)}$ is an averaged tensor of $\mathcal{Y}$ according to previous clustering group $\boldsymbol{M}_k^{(t)}$ for all $k \in [K]$. For some simulations, I found that this clustering group update makes it easier to trap in local minimums and stop updating. Instead, I tried to update the cluster group $\boldsymbol{M}_k^{(t+1)}$ using k-means on $\mathcal{M}_k(\mathcal{Y}_k^{(t)})$. Though this step does not guarantee monotonic decreasing objective values, it forces the algorithm to update clustering groups so that last objective value has smaller one in the end. I will label this method as `tbmClustering2`.

3 Output

Figure 2 shows the MSE according to different simulation settings. As we expected, **pb_smooth** setting is the easiest. For **non_smooth** settings, the algorithm seems to be trapped easily on local minimum and does not update clustering groups well. Since **sn_non_smooth** has the worst performance, local minimum problem is not from small magnitude of signal but other factors (one can check higher signal case where $\Theta(\omega) \in [-20.20]$ for all $\omega \in [n]^3$). One possible explanation for **sn_smooth** having worse MSE result compared to **pb_smooth** is that **sn_smooth** is locally hard to distinguish memberships as in Figure 3. To be specific, **pb_smooth** has an intrinsic Bernoulli noise proportional to the probability size while **sn_smooth** has uniform noise which is independent of signal magnitude. Therefore, if we sort signal tensor and add uniform noise, relative noise size (which I define as $|\Theta(\omega_1) - \Theta(\omega_1)|/\sigma$ where $|\omega_1 - \omega_2| \le c$) increases.

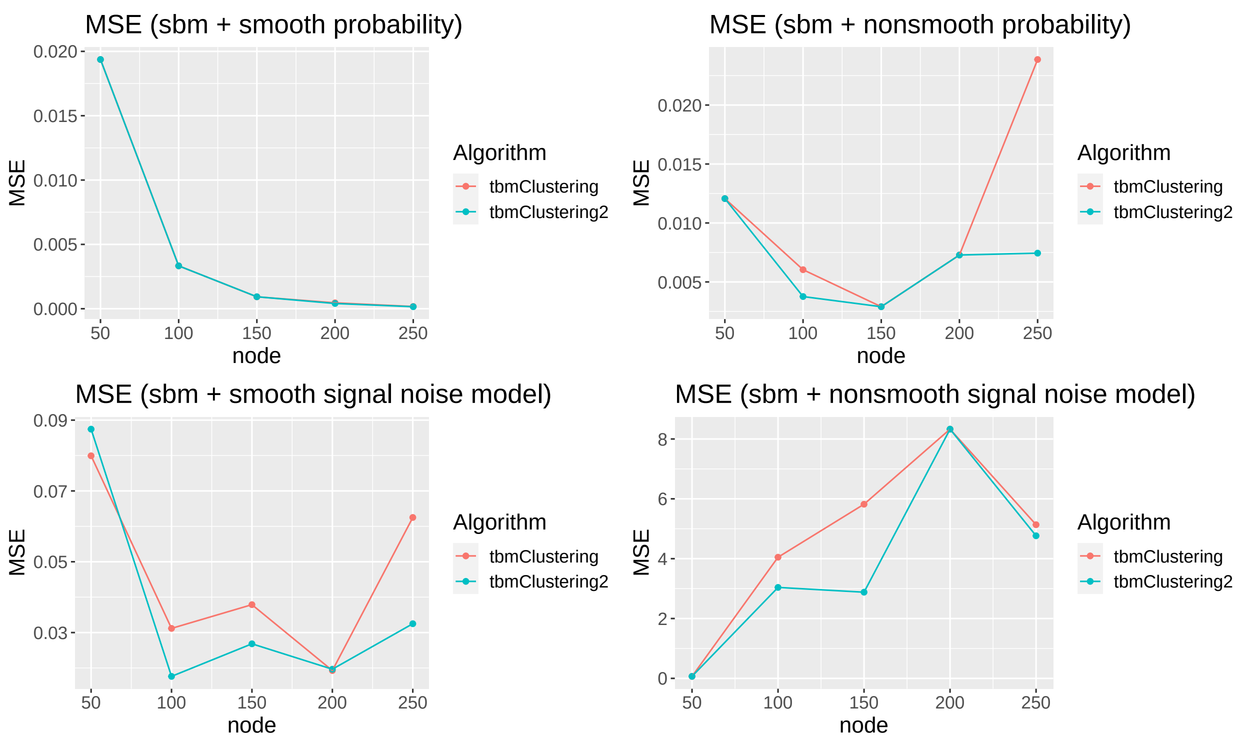


Figure 2: MSE results according to different simulation settings

4 Extra figures

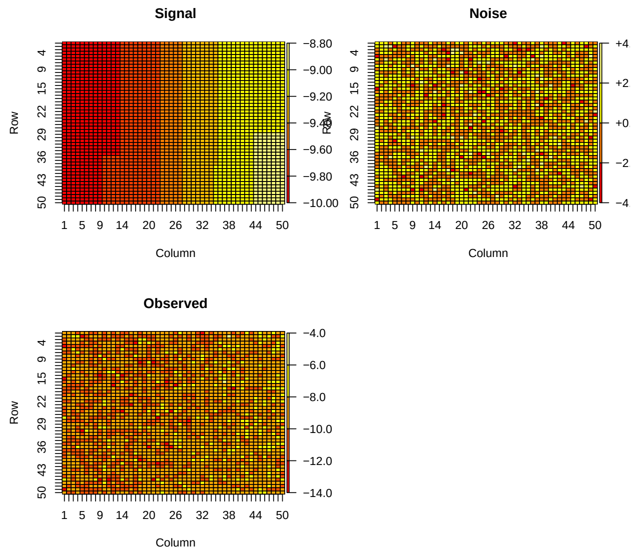


Figure 3: Signal, noise, and observed tensor in **sn_smooth** with n = 50

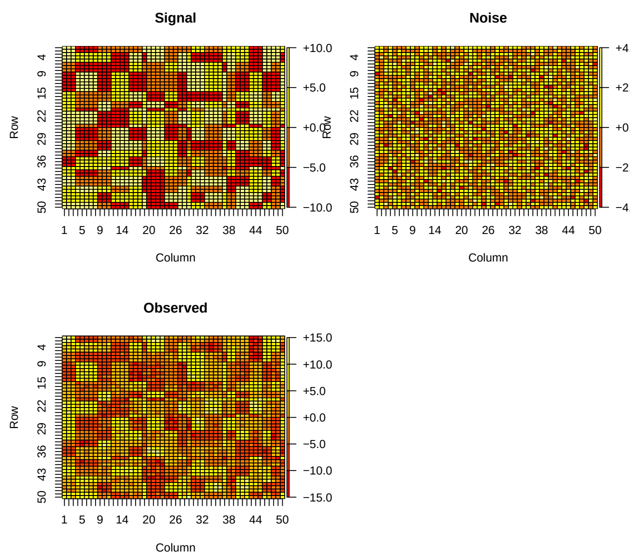
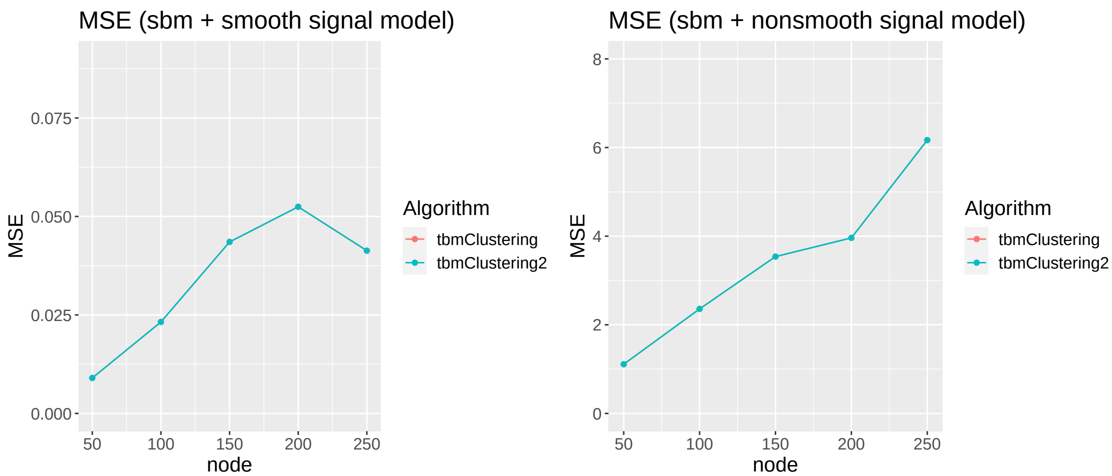


Figure 4: Signal, noise, and observed tensor in **sn_non_smooth** with n = 50

core tensor: U[0,10] sort.
sym = T; pbtensor
sym=F; hgmodel.bloc
diagP=T;

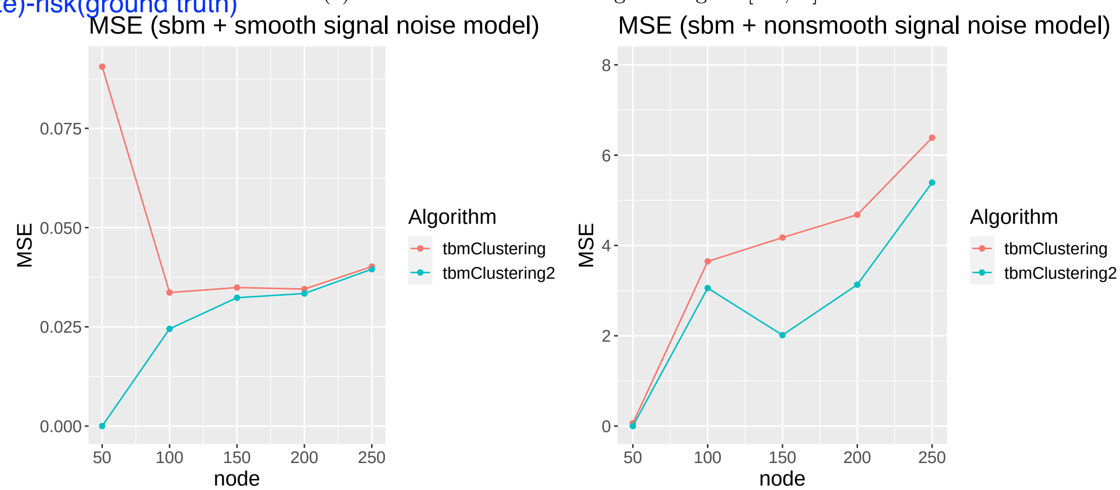
kmeans()
KmeansPP(…)



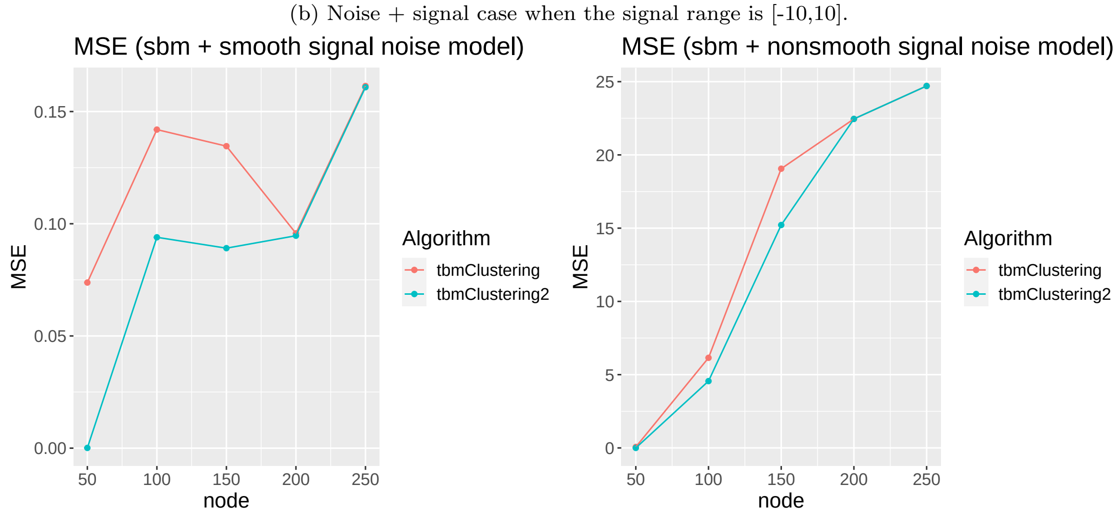
excess risk:

risk(estimate)-risk(ground truth)

(a) Noiseless case when the signal range is [-10,10]



(b) Noise + signal case when the signal range is [-10,10].



(c) Noise + signal case when the signal range is [-20,20].