A Appendix

A.1 Coordinate Descent Updates for Alternating Newton Coordinate Descent Method

In our alternating Newton coordinate descent algorithm, each element of $\Delta_{\mathbf{\Lambda}}$ is updated as follows:

$$(\Delta_{\mathbf{\Lambda}})_{ij} \leftarrow (\Delta_{\mathbf{\Lambda}})_{ij} - c_{\mathbf{\Lambda}} + S_{\lambda_{\mathbf{\Lambda}}/a_{\mathbf{\Lambda}}}(c_{\mathbf{\Lambda}} - \frac{b_{\mathbf{\Lambda}}}{a_{\mathbf{\Lambda}}}),$$

where $S_r(w) = \text{sign}(w)\max(|w| - r, 0)$ is the soft-thresholding operator and

$$\begin{aligned}
a_{\mathbf{\Lambda}} =& \mathbf{\Sigma}_{ij}^2 + \mathbf{\Sigma}_{ii}\mathbf{\Sigma}_{jj} + \mathbf{\Sigma}_{ii}\mathbf{\Psi}_{jj} + 2\mathbf{\Sigma}_{ij}\mathbf{\Psi}_{ij} + \mathbf{\Sigma}_{jj}\mathbf{\Psi}_{ii} \\
b_{\mathbf{\Lambda}} =& (\mathbf{S}_{\mathbf{yy}})_{ij} - \mathbf{\Sigma}_{ij} - \mathbf{\Psi}_{ij} + (\mathbf{\Sigma}\Delta_{\mathbf{\Lambda}}\mathbf{\Sigma})_{ij} + (\mathbf{\Psi}\Delta_{\mathbf{\Lambda}}\mathbf{\Sigma})_{ij} + (\mathbf{\Psi}\Delta_{\mathbf{\Lambda}}\mathbf{\Sigma})_{ji} \\
c_{\mathbf{\Lambda}} =& \mathbf{\Lambda}_{ij} + (\Delta_{\mathbf{\Lambda}})_{ij}.
\end{aligned}$$

For $\mathbf{\Theta}$, we perform coordinate-descent updates directly on $\mathbf{\Theta}$ without forming a second-order approximation of the log-likelihood to find a Newton direction, as follows:

$$\mathbf{\Theta}_{ij} \leftarrow \mathbf{\Theta}_{ij} - c_{\mathbf{\Theta}} + S_{\lambda_{\mathbf{\Theta}}/a_{\mathbf{\Theta}}}(c_{\mathbf{\Theta}} - \frac{b_{\mathbf{\Theta}}}{a_{\mathbf{\Theta}}}),$$

where

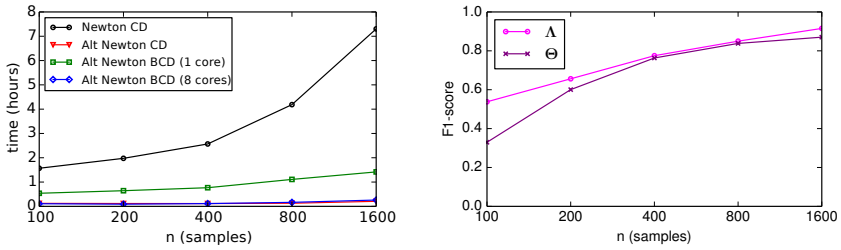
$$\begin{aligned}
a_{\mathbf{\Theta}} =& 2\mathbf{\Sigma}_{jj}(\mathbf{S}_{\mathbf{xx}})_{ii} \\
b_{\mathbf{\Theta}} =& 2(\mathbf{S}_{\mathbf{xy}})_{ij} + 2(\mathbf{S}_{\mathbf{xx}}\mathbf{\Theta}\mathbf{\Sigma})_{ij} \\
c_{\mathbf{\Theta}} =& \mathbf{\Theta}_{ij}.
\end{aligned}$$

A.2 Time Complexity Analysis of Alternating Newton Block Coordinate Descent

In this section we describe the time complexity of the alternating Newton block coordinate descent method. The active set sizes for $\mathbf{\Lambda}$ and $\mathbf{\Theta}$ are $m_{\mathbf{\Lambda}}$ and $m_{\mathbf{\Theta}}$, respectively. Also, because we use the iterative conjugate gradient method to compute columns of $\mathbf{\Sigma}$, we assume that solving $\mathbf{\Lambda}\mathbf{\Sigma}_i = \mathbf{e}_i$ takes at most K iterations.

A.2.1 Time Cost of Updating $\mathbf{\Lambda}$

The time complexity of each $\mathbf{\Lambda}$ update is dominated by the cost of precomputing columns of $\mathbf{\Sigma}$ and $\mathbf{\Psi}$. The cost of these precomputations is $O\Big(\big[1 + \frac{B_{\mathbf{\Lambda}}}{q}\big][m_{\mathbf{\Lambda}}Kq + nq^2]\Big)$, where $B_{\mathbf{\Lambda}} = \sum_{z \neq r} |\{j | i \in C_z, j \in C_r, (i,j) \in \mathcal{S}_{\mathbf{\Lambda}}\}|$ is the number of cache misses and K is the number of conjugate gradient iterations. Although the worst-case of $B_{\mathbf{\Lambda}} = k_{\mathbf{\Lambda}}q$ requires computing $\mathbf{\Sigma}$ and $\mathbf{\Psi}$ a total of $k_{\mathbf{\Lambda}}$ times, in practice, graph clustering dramatically reduces this additional cost of block-wise optimization. In the best case, when graph clustering identifies perfect block-diagonal structure in the active set, the number of cache misses $B_{\mathbf{\Lambda}} = 0$ and we incur no runtime penalty from limited memory.



(a) (b)

Figure 1: Results from varying sample size n on chain graph with $p = q = 10{,}000$. (a) Comparison of computation time of different methods. (b) Comparison of edge recovery accuracy as measured by F_1-score.

A.2.2 Time Cost of Updating $\boldsymbol{\Theta}$

The overall runtime is dominated by the cost of precomputing columns of $\mathbf{S_{xx}}$ and $\boldsymbol{\Sigma}$. The complexity of these operations is $O(m_{\boldsymbol{\Lambda}}Kq+m_{\boldsymbol{\Theta}}q+n\tilde{p}B_{\boldsymbol{\Theta}})$, where $\tilde{p}$ is the number of non-empty rows in $\boldsymbol{\Theta}$ and $B_{\boldsymbol{\Theta}} = \sum_{i,r} |\{i|j \in C_r, (i,j) \in \mathcal{S}_\Theta\}|$ is the number of cache misses. Without any row-wise sparsity we have $\tilde{p} = p$ and $B_{\boldsymbol{\Theta}} = k_{\boldsymbol{\Theta}}p$, so the worst-case is that $\mathbf{S_{xx}}$ is computed a total of $k_{\boldsymbol{\Theta}}$ times. The additional cost of computing $\mathbf{S_{xx}}$ due to $B_{\boldsymbol{\Theta}}$ cache misses is substantially reduced in real datasets where most input variables influence few or none of the outputs. In the best case, if the active set of $\boldsymbol{\Theta}$ has a block structure, where each input influences only one group of outputs, overall $\mathbf{S_{xx}}$ will be computed at most once per iteration and $B_{\boldsymbol{\Theta}} \leq \tilde{p}$.

A.3 Additional Results from Synthetic Data Experiments

We compare the performance of the different algorithms on synthetic datasets with different sample sizes n, using a chain graph structure with $p = q = 10{,}000$. Figure 1(a) shows that our methods run significantly faster than the previous method across all sample sizes. In Figure 1(b) we measure the accuracy in recovering the true chain graph structure in terms of F_1-score for different sample sizes n. At convergence, F_1-score was the same for all methods to three significant digits. As expected, the accuracy improves as the sample size increases.