antGLasso: An Efficient Tensor Graphical Lasso Algorithm [Supplementary Material]

BAILEY ANDREW, University of Leeds, UK
DAVID WESTHEAD, University of Leeds, UK
LUISA CUTILLO, University of Leeds, UK

The class of bigraphical lasso algorithms (and, more broadly, 'tensor'-graphical lasso algorithms) has been used to estimate dependency structures within matrix and tensor data. However, all current methods to do so take prohibitively long on modestly sized datasets. We present a novel tensor-graphical lasso algorithm that directly estimates the dependency structure, unlike its iterative predecessors. This provides a speedup of multiple orders of magnitude, allowing this class of algorithms to be used on large, real-world datasets. We consider its applicability to video data and single-cell RNA sequencing datasets.

CCS Concepts: • **Theory of computation** → **Design and analysis of algorithms**; • **Mathematics of computing** → *Probability and statistics*; • **Applied computing** → *Computational genomics*; *Transcriptomics*.

Additional Key Words and Phrases: bigraphical lasso, network inference, scRNA sequencing

ACM Reference Format:
Bailey Andrew, David Westhead, and Luisa Cutillo. 2022. antGLasso: An Efficient Tensor Graphical Lasso Algorithm [Supplementary Material]. In *LeedsAIMed-ConSim2022: Papers, Presentations and Posters on AI in Medical Diagnosis and Care, September 09, 2022, Leeds, UK*. ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

A PROOFS

A.1 Analytic Solution

Theorem A.1. *Let* $\mathbf{V}_\ell \mathbf{\Lambda}_\ell \mathbf{V}_\ell^T$ *be the eigendecomposition of* $\mathbf{\Psi}_\ell$. $\mathbf{V}_\ell$ *are the eigenvectors of* $\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell - 1}$.

Proof. Greenewald et al[?] present a maximum likelihood estimator for the tensor variate case: $-\log \left| \bigoplus_{\ell=1}^K \mathbf{\Psi}_\ell \right| + \sum_{\ell=1}^K m_k \mathbf{S}_\ell^T \mathbf{\Psi}_\ell$. Using the exact same argument as Kalaitzis et al[?], we can derive the fixed point $\mathbf{S}_\ell - \frac{1}{2m_\ell}\mathbf{S}_\ell \circ \mathbf{I} = \frac{1}{m_\ell}\mathrm{tr}_{m_\ell}[(\bigoplus_i \mathbf{\Psi}_i)^{-1}] - \frac{1}{2m_\ell}\mathrm{tr}_{m_\ell}[(\bigoplus_i \mathbf{\Psi}_i)^{-1}] \circ \mathbf{I}$. The order in which we compute $\bigoplus_i \mathbf{\Psi}_i$ matters - we'll assume here that we've transposed the data such that $\bigoplus_i \mathbf{\Psi}_i = \mathbf{\Psi}_\ell \oplus \bigoplus_{i \neq \ell} \mathbf{\Psi}_i$. It is only important that $\mathbf{\Psi}_\ell$ goes first, the rest of the order does not matter. Let $\mathbf{\Psi}_{\setminus \ell} = \bigoplus_{i \neq \ell} \mathbf{\Psi}_i$ so that $\bigoplus_i \mathbf{\Psi}_i = \mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\setminus \ell}$

$$\mathbf{S}_\ell - \frac{1}{2m_\ell}\mathbf{S}_\ell \circ \mathbf{I} = \frac{1}{m_\ell}\mathrm{tr}_{m_\ell}[(\mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\backslash\ell})^{-1}] - \frac{1}{m_\ell}\mathrm{tr}_{m_\ell}[(\mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\backslash\ell})^{-1}] \circ \mathbf{I} \tag{1}$$

$$\mathbf{S}_\ell \circ \mathbf{K}_1^{\frac{2m_\ell-1}{2m_\ell}} = \mathrm{tr}_{m_\ell}[(\mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\backslash\ell})^{-1}] \circ \mathbf{K}^{\frac{1}{2m_\ell}}_{\frac{1}{m_\ell}} \tag{2}$$

$$\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell-1} = \mathrm{tr}_{m_\ell}[(\mathbf{\Psi}_\ell \oplus \mathbf{\Psi}_{\backslash\ell})^{-1}] \tag{3}$$

$$\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell-1} = \mathrm{tr}_{m_\ell}[(\mathbf{V}_\ell \otimes \mathbf{V}_{\backslash\ell})(\mathbf{\Lambda}_\ell \oplus \mathbf{\Lambda}_{\backslash\ell})^{-1}(\mathbf{V}_\ell^T \otimes \mathbf{V}_{\backslash\ell}^T)] \tag{4}$$

$$\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell-1} = \mathrm{tr}_{m_\ell}[(\mathbf{V}_\ell \otimes \mathbf{I})(\mathbf{\Lambda}_\ell \oplus \mathbf{\Lambda}_{\backslash\ell})^{-1}(\mathbf{V}_\ell^T \otimes \mathbf{I})] \tag{5}$$

$$\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell-1} = \mathbf{V}_\ell \mathrm{tr}_{m_\ell}[(\mathbf{\Lambda}_\ell \oplus \mathbf{\Lambda}_{\backslash\ell})^{-1}]\mathbf{V}_\ell^T \tag{6}$$

The penultimate step uses Proposition 3.1 of Li et al[**?**], and the last step requires Lemma 2 of Dahl et al[**?**]. As $\mathrm{tr}_{m_\ell}[(\mathbf{\Lambda}_\ell \oplus \mathbf{\Lambda}_{\backslash\ell})^{-1}]$ is diagonal, we can observe that $\mathbf{V}_\ell$ must be the eigenvectors of $\mathbf{S}_\ell \circ \mathbf{K}_{m_\ell}^{2m_\ell-1}$.

□

Lemma A.2. *Suppose $\mathcal{Y} \sim \mathcal{N}(0, \zeta\{\mathbf{\Psi}_i\}^{-1})$. Then we can diagonalize the precision matrix as follows: $\mathcal{X} = \mathcal{Y} \times_1 \mathbf{V}_1^T \times_2 \ldots \times_K \mathbf{V}_K^T \sim \mathcal{N}(\mathbf{0}, \zeta\{\mathbf{\Lambda}_i\}^{-1})$.*

Proof. We will show that the probability density function of $\mathcal{X}$ is that of a Kronecker sum distribution with the desired parameters. It will rely on the following useful property of ℓ-mode matrix multiplication: $(\mathcal{Y} \times_1 \mathbf{V}_1 \times_2 \ldots \times_K \mathbf{V}_K)_{(\ell)} = \mathbf{V}_\ell \mathcal{Y}_{(\ell)}(\mathbf{V}_K \otimes \ldots \otimes \mathbf{V}_{\ell+1} \otimes \mathbf{V}_{\ell-1} \otimes \ldots \otimes \mathbf{V}_1)^T$.

$$\mathrm{pdf}(\mathcal{X}) = \mathrm{pdf}(\mathcal{Y}) \tag{7}$$

$$= (2\pi)^{\frac{-\prod_i d_i}{2}} \sqrt{\left|\bigoplus_i \mathbf{\Psi}_i\right|} e^{\frac{-1}{2}\sum_\ell^K \mathrm{tr}[\mathbf{\Psi}_\ell \mathcal{Y}_{(\ell)} \mathcal{Y}_{(\ell)}^T]} \tag{8}$$

$$\left|\bigoplus_i \mathbf{\Psi}_i\right| = \left|\bigotimes_i \mathbf{V}_i\right| \left|\bigoplus_i \mathbf{\Lambda}_i\right| \left|\bigotimes_i \mathbf{V}_i^T\right| \tag{9}$$

$$= \left|\bigoplus_i \mathbf{\Lambda}_i\right| \tag{10}$$

$$\mathrm{tr}[\mathbf{\Psi}_\ell \mathcal{Y}_{(\ell)} \mathcal{Y}_{(\ell)}^T] = \mathrm{tr}[\mathbf{\Psi}_\ell (\mathcal{X} \times_1 \mathbf{V}_1 \times_2 \ldots \times_K \mathbf{V}_K)_{(\ell)} (\mathcal{X} \times_1 \mathbf{V}_1 \times_2 \ldots \times_K \mathbf{V}_K)_{(\ell)}^T] \tag{11}$$

$$= \mathrm{tr}[\mathbf{\Psi}_\ell \mathbf{V}_\ell \mathcal{X}_{(\ell)} (\bigotimes_{i \neq \ell} \mathbf{V}_i)^T (\bigotimes_{i \neq \ell} \mathbf{V}_i) \mathcal{X}_{(\ell)}^T \mathbf{V}_\ell^T] \tag{12}$$

$$= \mathrm{tr}[\mathbf{V}_\ell^T \mathbf{\Psi}_\ell \mathbf{V}_\ell \mathcal{X}_{(\ell)} \mathcal{X}_{(\ell)}^T] \tag{13}$$

$$= \mathrm{tr}[\mathbf{\Lambda}_\ell \mathcal{X}_{(\ell)} \mathcal{X}_{(\ell)}^T] \tag{14}$$

□

A.1.1 Corollary A.3. This provides a convenient way to sample from tensor-variate Kronecker sum distributions without high dimensionality or expensive matrix inverses/decompositions.

Corollary A.3. *Suppose you have $\prod_i d_i$ samples of $\mathcal{N}(0,1)$ $\mathbf{z}$, then we have that* $\mathrm{vec}^{-1}[(\bigoplus_i \mathbf{\Lambda}_i)^{-1/2}\mathbf{z}] \times_1 \mathbf{V}_1 \times_2 \ldots \times_K \mathbf{V}_K \sim \mathcal{N}_{KS}(\mathbf{0}, \{\mathbf{V}_i \mathbf{\Lambda}_i \mathbf{V}_i^T\})$

Proof.

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{15}$$

$$(\bigoplus_i \mathbf{\Lambda}_i)^{-1/2}\mathbf{z} \sim \mathcal{N}(\mathbf{0}, (\bigoplus_i \mathbf{\Lambda}_i)^{-1}) \tag{16}$$

$$\text{vec}^{-1}[(\bigoplus_i \mathbf{\Lambda}_i)^{-1/2}\mathbf{z}] \sim \mathcal{N}_{KS}(\mathbf{0}, \{\mathbf{\Lambda}_i\}) \tag{17}$$

$$\text{vec}^{-1}\left[(\bigoplus_i \mathbf{\Lambda}_i)^{-1/2}\mathbf{z}\right] \times_1 \mathbf{V}_1 \times_2 \dots \times_K \mathbf{V}_K \sim \mathcal{N}_{KS}(\mathbf{0}, \{\mathbf{V}_i\mathbf{\Lambda}_i\mathbf{V}_i^T\}) \tag{18}$$

□

Lemma A.4. $\frac{M}{\sum_n^m (x_{i_1 \dots i_K}^{(n)})^2} \approx \sum_\ell^K \lambda_{i_\ell}^\ell$.

Proof. Let $d_{1:n} = \prod_{\ell=1}^n d_\ell$. Defining $d_{1:0}$ as 1, we can observe that:

$$x_{ij} = \text{vec}[\mathcal{X}]_{\sum_\ell i_\ell d_{1:(\ell-1)}} \tag{19}$$

$$\frac{1}{M}\sum_n^m (x_{i_1 \dots i_K}^{(n)})^2 \approx \text{var}[x_{i_1 \dots i_K}] \tag{20}$$

$$= ((\bigoplus_\ell \mathbf{\Lambda}_\ell)^{-1})_{\sum_\ell i_\ell d_{1:(\ell-1)}, \sum_\ell i_\ell d_{1:(\ell-1)}} \tag{21}$$

$$= \frac{1}{\sum_\ell^K \lambda_{i_\ell}^\ell} \tag{22}$$

$$\frac{M}{\sum_n^m (x_{i_1 \dots i_K}^{(n)})^2} \approx \sum_\ell^K \lambda_{i_\ell}^\ell \tag{23}$$

□

Theorem A.5. *We can obtain the eigenvalues from the variances via a linear system.*

Proof. It is easy to see that the approximation of Lemma A.4 defines a linear system of the form $\mathbf{a} = \mathbf{B}\mathbf{\Lambda}$, where $\mathbf{a}$ is a vector whose elements are $\frac{M}{\sum_n^m (x_{i_1 \dots i_K}^{(n)})^2}$ and $\mathbf{\Lambda}$ is a vector made from stacking the eigenvalues of each axis. Since the RHS of Lemma A.4 is a linear combination of eigenvalues, it is easy to construct a matrix $\mathbf{B}$ relating the two. It is not invertible: the eigenvalues are underdetermined[1]. However, we can select the least squares solution by multiplyling both sides by the pseudo-inverse of $\mathbf{B}$: $\mathbf{B}^\dagger\mathbf{a} \approx \mathbf{\Lambda}$.

The matrix $\mathbf{B}$ will be much larger than necessary - in Section A.3 we will see how to reduce its size. □

A.1.2 Regularization. The original BiGLasso performs lasso independently on each row of the precision matrices at each iteration. The natural analogy in this case would be to perform row-wise lasso at the end of antGLasso. The function to minimize is, for a row $\hat{r}$, $f(r) = (r - \hat{r})^T(r - \hat{r}) + \beta\,||r||_1$. If r and $\hat{r}$ were restricted to be nonnegative, then this would be differentiable. Suppose for now that that is the case.

[1]This system is both overdetermined and undertermined, in the sense that it is a rectangular matrix (overdetermined) whose rank is not maximal (undetermined). Its rank is always $K - 1$ less than maximal, where K is the number of tensor dimensions of the input.

$$\frac{\partial}{\partial r_i} f(r) = \frac{\partial}{\partial r_i}(r - \hat{r})^T (r - \hat{r}) + \beta ||r||_1 \tag{24}$$

$$= \frac{\partial}{\partial r_i} \sum_i (r_i - \hat{r}_i)^2 + \beta r_i \tag{25}$$

$$= 2(r_i - \hat{r}_i) + \beta \tag{26}$$

$$-\frac{\beta}{2} = r_i - \hat{r}_i \tag{27}$$

$$\hat{r}_i - \frac{\beta}{2} = r_i \tag{28}$$

Since the domain of $\hat{r}_i$ is nonnegative, and the function is monotonic, if $\hat{r}_i - \frac{\beta}{2} < 0$ then the minimum on the domain occurs at $r_i = 0$. We can easily enforce a nonnegative domain by performing our regularization after taking the absolute value of the row. This gives us a regularizer $\text{shrink}(\hat{r}) = \text{sign}(\hat{r}) \circ \min(0, |\hat{r}| - \frac{\beta}{2})$ that is equivalent to performing row-wise Lasso on the output of our algorithm. Note that this allows us to reframe the regularization as a thresholding. In fact, the threshold does not depend on the row, but is rather a global constraint. Thus, instead of using β as an argument, we could instead give the algorithm a percent of edge connections to keep. When framed this way, our hyperparameters become easily interpretable! We found that when setting the threshold percent to be the true percent of connections in simulated data, the result had roughly equal precision and recall.

A.2 Heuristic

Suppose we want to find the variances of the elements of the $\mathcal{X}$ tensor using only the empirical covariance matrices produced by the Nonparanormal Skeptic method[**?**]. This would allow us to generalize to non-Gaussian data *à la* Li et al[**?**]. Suppose there exists some function f that maps $\mathcal{Y}$ to a normal distribution. The Nonparanormal Skeptic directly computes $f(\mathcal{Y})f(\mathcal{Y})^T$.

$$\text{var}\left[\mathcal{X}_{i_1,...,i_K}\right] = \text{var}\left[f(\mathcal{Y})\bar{\times}_1 \mathbf{V}_1 \Delta_{i_1}^T \bar{\times}_2 ... \bar{\times}_K \mathbf{V}_K \Delta_{i_K}^T\right] \tag{29}$$

$$= \mathbb{E}\left[(f(\mathcal{Y})\bar{\times}_1 \mathbf{V}_1 \Delta_{i_1}^T \bar{\times}_2 ... \bar{\times}_K \mathbf{V}_K \Delta_{i_K}^T)(f(\mathcal{Y})^T \bar{\times}_1 \Delta_{i_K} \mathbf{V}_K^T \bar{\times}_2 ... \bar{\times}_K \Delta_{i_1} \mathbf{V}_1^T)\right] \tag{30}$$

$$= \Delta_{i_1} \mathbf{V}_1^T \mathbb{E}\left[(f(\mathcal{Y})\bar{\times}_2 ... \bar{\times}_K \mathbf{V}_K \Delta_{i_K}^T)(f(\mathcal{Y})^T \bar{\times}_1 \Delta_{i_K} \mathbf{V}_K^T \bar{\times}_2 ... \bar{\times}_{K-1} \Delta_{i_2} \mathbf{V}_2^T)\right] \mathbf{V}_1 \Delta_{i_1}^T \tag{31}$$

Let $\tilde{f}(Y) = f(\mathcal{Y})\bar{\times}_2 ... \bar{\times}_K \mathbf{V}_K \Delta_{i_K}^T$. Then:

$$\text{var}\left[\mathcal{X}_{i_1,...,i_K}\right] = \Delta_{i_1} \mathbf{V}_1^T \mathbb{E}\left[\tilde{f}(\mathcal{Y})\tilde{f}(\mathcal{Y})^T\right] \mathbf{V}_1 \Delta_{i_1}^T \tag{32}$$

$$\approx d_1 \Delta_{i_1} \mathbf{V}_1^T \mathbf{S}_1 \mathbf{V}_1 \Delta_{i_1}^T \tag{33}$$

Our heuristic is to assume that the Nonparanormal Skeptic is directly producing $\tilde{f}$ instead of f. This is clearly a faulty assumption, not least of which because $\tilde{f}$ depends on the indices $i_2, ..., i_K$. However, in practice it works very well, and in fact gives comparable accuracy to EiGLasso. The heuristic only depends on the first dimension's eigenvectors, which means that the eigenvalues for the other dimensions won't typically be in the same order as their corresponding eigenvectors. To get around this, we run the calculation multiple times, once for each dimension (the formula for each dimension's variances is exactly analogous).

A.3 Monte Carlo Speedup

As the speedup is technical, it is helpful to see a worked example for the (2, 2, 3, 2) tensor case. We have $\mathbf{B\Lambda} = \mathbf{a}$ and we want to find a smaller matrix $\tilde{\mathbf{B}}$ which preserves the system. The idea is to guess the value of one eigenvalue for each axis, as doing so will fix a unique solution for the rest of them. A series of algebraic manipulations will make finding such a unique solution easy. In doing so we only look at a small subset of $\mathbf{B}$ ($\tilde{\mathbf{B}}$) and hence need to make several guesses and average out the result so that our solution is not overly affected by any individual guess.

We'll choose $(\lambda^1_2 = 1, \lambda^2_1 = 1, \lambda^3_2 = 1, \lambda^4_2 = 1)$ as our initial guesses for this worked example. In fact, we do not need to make a guess for one of the axes (the rank of the matrix is $K - 1$ less than maximal, so we only need $K - 1$ guesses). However it's simpler to describe if we ignore this for now. Note that if we simultaneously swap the i, jth columns of $\mathbf{B}$ and the i, jth rows of $\mathbf{\Lambda}$, the system is preserved. Likewise for the i, jth rows of $\mathbf{B}$ and the i, jth rows of $\mathbf{a}$. In light of that, we will express the system as:

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array}$$

and perform a series of row and column swaps to simplify it. We'll say that two systems of equations are equal if they can be arranged into each other through a series of permutations and removal of redundant[2] rows/columns.

[2]In practice, these equations are not consistent, so there is overdeterminedness that is not actually redundant - hence the need to run multiple iterations of this and average the result.

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array} = \begin{array}{cc:cc:ccc:cc|c}
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_1 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_2 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_3 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_4 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_5 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_6 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & a_7 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & a_8 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_9 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_{10} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_{11} \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_{12} \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{13} \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{14} \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_{15} \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_{16} \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & a_{17} \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & a_{18} \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & a_{19} \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & a_{20} \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & a_{21} \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & a_{22} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & a_{23} \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & a_{24} \\
\hline
\lambda_1^1 & \lambda_2^1 & \lambda_1^2 & \lambda_2^2 & \lambda_1^3 & \lambda_2^3 & \lambda_3^3 & \lambda_1^4 & \lambda_2^4 &
\end{array} \tag{34}$$

We want to re-order this system so that it behaves as if we had chosen $(\lambda_1^1, \lambda_1^2, \lambda_1^3, \lambda_1^4)$ as our initial guesses (i.e. guessing the first eigenvalue of each axis). Note that if we swapped the first and second columns, and then swapped every odd column with every even row, the matrix would look exactly the same but now the λ^1 we guessed is the first column rather than the second. A similar line of reasoning applies to the other guesses we made, although instead of swapping individual rows we need to swap chunks of rows. The size of the chunks would be 2 for the second guess, 4 for the third guess, and 12 for the fourth guess. This is because we want to preserve the structure from the previous guesses - the size of the tensor is (2, 2, 3, 2) and hence the size of the chunks are $(1, 2, 2 \times 2, 2 \times 2 \times 3)$.

After doing this, you'll get a system looking like this:

$$
\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array}
=
\begin{array}{cc|cc|ccc|cc|c}
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_{18} \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_{17} \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_{20} \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_{19} \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_{14} \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_{13} \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & a_{16} \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & a_{15} \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_{22} \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_{21} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_{24} \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_{23} \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{6} \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{5} \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_{8} \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_{7} \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & a_{2} \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & a_{1} \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & a_{4} \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & a_{3} \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & a_{10} \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & a_{9} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & a_{12} \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & a_{11} \\
\hline
\lambda^1_2 & \lambda^1_1 & \lambda^2_1 & \lambda^2_2 & \lambda^3_2 & \lambda^3_1 & \lambda^3_3 & \lambda^4_2 & \lambda^4_1 &
\end{array}
\tag{35}
$$

Note that the **B** component did not change, but the order of **a** and **Λ** did. We can then shrink this matrix by grabbing the first row, and every row that differs from it by the position of exactly one of the 1s.

$$
\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array}
=
\begin{array}{cc|cc|ccc|cc|c}
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_{18} \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & a_{17} \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & a_{20} \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & a_{14} \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_{22} \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_{6} \\
\hline
\lambda^1_2 & \lambda^1_1 & \lambda^2_1 & \lambda^2_2 & \lambda^3_2 & \lambda^3_1 & \lambda^3_3 & \lambda^4_2 & \lambda^4_1 &
\end{array}
\tag{36}
$$

The system of equations is no longer overdetermined, and the set of solutions of this new system will be a subset of the old system. Our goal is now to reshape this into an upper triangular matrix. Move all of our guess columns to the end, and then the top row to the bottom:

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array} = \begin{array}{ccccc:cccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & a_{17} \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & a_{20} \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & a_{14} \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & a_{22} \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & a_{6} \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & a_{18} \\ \hline \lambda_1^1 & \lambda_2^2 & \lambda_1^3 & \lambda_3^3 & \lambda_1^4 & \lambda_2^1 & \lambda_1^2 & \lambda_2^3 & \lambda_2^4 & \end{array} \tag{37}$$

We then add our guesses in to make the matrix square:

$$\begin{array}{c|c} \mathbf{B} & \mathbf{a} \\ \hline \mathbf{\Lambda}^T & \end{array} = \begin{array}{ccccc:cccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & a_{17} \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & a_{20} \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & a_{14} \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & a_{22} \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & a_{6} \\ \hdashline 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & a_{18} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \lambda_1^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \lambda_2^3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \lambda_2^4 \\ \hline \lambda_1^1 & \lambda_2^2 & \lambda_1^3 & \lambda_3^3 & \lambda_1^4 & \lambda_2^1 & \lambda_1^2 & \lambda_2^3 & \lambda_2^4 & \end{array} \tag{38}$$

Thus, we never actually need to create $\mathbf{B}$. We can directly create $\tilde{\mathbf{B}}$, and perform the demonstrated permutations on $\mathbf{a}$ and $\mathbf{\Lambda}$. As mentioned earlier, we don't actually need a guess for the first dimension (λ^1). To solve the system for $\mathbf{\Lambda}$, we need to find the inverse of $\tilde{\mathbf{B}}$. We've partitioned the matrix into four blocks (indicated by the dashed lines) - we can then invert this matrix using the block matrix inversion formula. Because of the simple forms of the submatrices involved, this is a cheap operation.

$$\tilde{\mathbf{B}}^{-1} = \left[\begin{array}{ccccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]^{-1} \tag{39}$$

$$= \left[\begin{array}{c|c} \mathbf{I} & \mathbf{M} \\ \hline \mathbf{0} & \mathbf{N} \end{array}\right]^{-1} \tag{40}$$

$$= \left[\begin{array}{c|c} \mathbf{I} & -\mathbf{M}\mathbf{N}^{-1} \\ \hline \mathbf{0} & \mathbf{N}^{-1} \end{array}\right] \tag{41}$$

The last step follows from the application of the block matrix inverse formula. Note that we have reduced the computation of $\tilde{\mathbf{B}}^{-1}$ to the inverse of a $K \times K$ matrix ($\mathbf{N}$) and one matrix multiplication.

We can see that $\mathbf{N}$ will always have the form it does (zeros except for ones at the diagonal and first row). The ones on the diagonal are because we added the guesses at the end, and the ones on the first row are because we had a 1 at the top of every 'guess' column. It is not hard to see that the inverse of such a matrix will have 1s on the diagonal, -1 on the off-diagonal first row, and 0s elsewhere. Hence, we can just construct its inverse directly.

We are interested in constructing $-\mathbf{M}\mathbf{N}^{-1}$ directly too. For a small number of iterations b, this is not a bottleneck - however, by computing it directly we can leverage sparse matrix multiplication routines in a manner such that we can easily run this computation a thousand times[3] before the computation time becomes noticeable. We can compute this product easily using block matrix multiplication.

[3]Which in practice is more than enough for our Monte Carlo approximation to be as accurate as the non-approximative version of the algorithm.

$$-\mathbf{M}\mathbf{N}^{-1} = -\left[\begin{array}{c:ccc} 0 & 1 & 1 & 1 \\ \hdashline 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{array}\right]\left[\begin{array}{c:ccc} 1 & -1 & -1 & -1 \\ \hdashline 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \tag{42}$$

$$= -\left[\begin{array}{c:c} 0 + \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} & 0\begin{bmatrix} -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}\mathbf{I} \\ \hdashline \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + 0 & \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}\begin{bmatrix} -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array}\right] \tag{43}$$

$$= -\left[\begin{array}{c:c} 0 & \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \\ \hdashline \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & -\mathbf{J} + \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array}\right] \tag{44}$$

$$= \left[\begin{array}{c:c} 0 & \begin{bmatrix} -1 & -1 & -1 \end{bmatrix} \\ \hdashline \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array}\right] \tag{45}$$

In terms of generalizing this beyond the example, the upper blocks should have as many rows as $d_1 - 1$ (but it will be the same row, repeated), and the lower right-hand block is always the same as in $\mathbf{M}$, but with the 1s and 0s swapped.

We have now reduced a very large matrix inversion and multiplication problem into a series of small, sparse matrix multiplications with a pre-computed matrix. This pushes the runtime bottleneck of our algorithm solely onto the eigendecomposition used in Theorem A.1.

The most expensive part of this reduction is the permutations we perform on $\mathbf{a}$. It's not noticeable for 2-axis tensor inputs, but can be noticeable for 3-axis inputs. Since after permuting $\mathbf{a}$ we select a subset of it, we can achieve further speedups by directly working out the indices of the desired elements of $\mathbf{a}$, rather than permuting them. For details on this final speedup, we refer the reader to our Python implementation of antGLasso.

A.4 Pseudocode

Here we give the pseudocode for antGLasso in Algorithm 1.

B ONLINE RESOURCES

A Python implementation of antGLasso is available on our GitHub, https://github.com/BaileyAndrew/antGLasso-Implementation. *This repository is currently set to private, although that will change in the near future.*

Algorithm 1 Analytic Tensor Graphical Lasso (antGLasso)

```
Input: $\{\mathcal{Y}^{(n)}_{d_1 \times \ldots \times d_K}\}, \{\beta_\ell\}, b$
Output: $\{\Psi_\ell\}$
  for $1 \leq \ell \leq n$
      $S_\ell \leftarrow \frac{1}{nm_\ell} \sum_i^n \mathbf{Y}^{(i)}_{(\ell)} \mathbf{Y}^{(i)T}_{(\ell)}$
      $\mathbf{V}_\ell \leftarrow$ eigenvectors$[S_\ell \circ \mathbf{K}^{2m_\ell - 1}_{m_\ell}]$
  end for
  for $1 \leq p \leq m$
      $\mathcal{X}^{(p)} \leftarrow \mathcal{Y}^{(p)} \times_1 \mathbf{V}_1 \times_2 \ldots \times_K \mathbf{V}_K$
  end for
  for $\vec{i} \in [1, \ldots, d_1] \times \ldots \times [1, \ldots, d_K]$
      $\mathbf{a}_{\sum_\ell i_\ell d_{1:(\ell-1)}} \leftarrow \frac{n}{\sum_p^n (x^{(p)}_{i_1, \ldots, i_K})^2}$
  end for
  $\Lambda \leftarrow \mathbf{0}$
  Construct $\tilde{\mathbf{B}}^{-1}$ as in Theorem ??
  for $b$ iterations of different tuples $\vec{i}$
      Let $\mathbf{P}^{\vec{i}}_1, \mathbf{P}^{\vec{i}}_2$ be the matrices representing the permutations that capture the process described in Section A.3.
      If first iteration, set each element in $\lambda^{1:(K-1)}_{\vec{i}}$ to 1
      $\Lambda \leftarrow \Lambda + \mathbf{P}^{\vec{i}}_1 \tilde{\mathbf{B}}^{-1} \mathbf{P}^{\vec{i}}_2 \begin{bmatrix} \mathbf{a} \\ \lambda^{1:(K-1)}_{\vec{i}} \end{bmatrix}$
  end for
  $\Lambda \leftarrow \Lambda / b$
  for $1 \leq \ell \leq K$
      $\Psi_\ell \leftarrow \mathbf{V}_\ell \Lambda_\ell \mathbf{U}^T_\ell$
      for $i$th row $\mathbf{r}$ in $\Psi_\ell$
          $\mathbf{r}_{\backslash i} \leftarrow \text{sign}(\mathbf{r}_{\backslash i}) \circ \min(0, |\mathbf{r}_{\backslash i}| - \frac{\beta_\ell}{2})$
      end for
  end for
```