SINCO - a greedy coordinate ascent method for sparse inverse covariance selection problem

K. Scheinberg and I. Rish

July 31, 2009

Abstract

In this paper, we consider the sparse inverse covariance selection problem which is equivalent to structure recovery of a Markov Network over Gaussian variables. We introduce a simple but efficient greedy algorithm, called *SINCO*, for solving the Sparse INverse COvariance problem. Our approach is based on coordinate ascent method which naturally preserves the sparsity of the inverse covariance matrix. We compare our algorithm to the state-of-art method called *glasso* [5], evaluating both computational efficiency and structure-reconstruction accuracy of both methods. We show that the two methods are often comparable in speed and accuracy, however, in some regimes, our method can significantly outperform *glasso* in terms of both computational time and structure reconstruction error (particularly, false positive error). Our method has an additional advantage of being easily parallelizable. We also show that the greedy nature of the method is such that one can reproduce the regularization path behavior by applying the method to one instance of the regularization parameter only. Numerical experiments demonstrate advantages of our approach on simulated networks, both random and "structured" (scale-free) ones, where the ground-truth structure is available. We also report promising empirical results on real-life problems with unknown ground-truth structure, such as classification of mental states from fMRI data.

1 Introduction

In many practical applications of statistical learning the objective is not simply to construct an accurate predictive model but rather to discover meaningful interactions among the variables. For example, in applications such as reverse-engineering of gene networks, discovery of functional brain connectivity patterns from brain-imaging data, or analysis of social interactions, the main focus is on reconstructing the network structure representing dependencies among multiple variables, such as genes, brain areas, or individuals. Probabilistic graphical models, such as Markov networks (or Markov Random Fields), provide a statistical tool for multivariate data analysis that allows to capture variable interactions explicitly, as conditional (in)dependence relationships. Herein, we focus on learning the structure of Markov Network over Gaussian random variables, which is equivalent to learning zero-pattern of the inverse covariance matrix. A standard approach to model selection is to choose the simplest model, i.e.

the sparsest network, that adequately explains the data. Formally, this leads to regularized maximum-likelihood problem with the penalty on the number of parameters, or l_0 norm, a generally intractable problem that is often solved approximately by greedy search [7]. Recently, however, novel tractable approximations were suggested that exploit sparsity-enforcing property of l_1-norm regularization and yield convex optimization problems [9, 15, 16, 12, 5].

Herein, we propose a very simple greedy algorithm (SINCO) for solving the l_1-regularized inverse-covariance problem, and compare it with the state-of-art. SINCO solves the primal problem (unlike its predecessors such as COVSEL [12] and *glasso* [5]), using coordinate descent, in a greedy manner, thus naturally preserving the sparsity of the solution. As demonstrated by our empirical results, SINCO has better capability in reducing the false positives error than *glasso* [5], since it is less prone to introducing unnecessary nonzero elements.

The structure reconstruction accuracy is known to be quite sensitive to the choice of regularization parameter, which we denote as λ, and the problem of selecting the "best" (i.e. giving the most accurate structure) value of this parameter in practical settings remains open, despite theoretical advances that analyze asymptotic behavior [1]. We investigate SINCO vs *glasso* behavior in several regimes of λ. What we observe is that SINCO's greedy approach introduces nonzero elements in the same manner as is achieved by reducing the value of λ. Hence, SINCO can reproduce regularization path behavior without actually varying the value of the regularization parameter. Note also that the path produced by SINCO is computed in its entirety, while the regularization path can only be computed for a selected set of values of λ.

Moreover, while *glasso* [5] is comparable to, or faster than SINCO for relatively small number of variables p, SINCO scales better when p increases (e.g., gets closer to 1000 variables), and can noticably outperform *glasso*. Finally, experiments on real-life brain imaging (fMRI) data demonstrate that SINCO reconstructs Markov Networks that achieve same or better classification accuracy than its competitors while using much smaller fraction of edges (non-zero entries of the inverse-covariance matrix). Further advantages of our approach include simplicity, efficiency, a relatively straightforward massive parallelization.

The paper is organized as follows. In Section 2 we state the problem formulation. In Section 3 we describe the SINCO algorithm and the details of the coordinate descent step computation. The regularization path computation and the path generated by SINCO are discussed in Section 4. In Section 5 we discuss the empirical complexity of SINCO with respect to the stopping tolerance and the size of the problem. We also show that SINCO has a particular strength in reducing the false positive error. Finally, we present some results on real-life neuroimaging (fMRI) data in Section 6, and discuss other applications in Section 7.

2 Problem Formulation

Let $X = \{X_1, ..., X_p\}$ be a set of p random variables, and let $G = (V, E)$ be a Markov network (a Markov Random Field, or MRF) representing the conditional independence

[1] As mentioned in [10], "the general issue of selecting a proper amount of regularization for getting a right-sized structure or model has largely remained a problem with unsatisfactory solutions".

structure of the joint distribution $P(X)$. The set of vertices $V = \{1, ..., p\}$ is in a one-to-one correspondence with the set of variables in X. The edge set E contains an edge (i, j) if and only if X_i is conditionally dependent on X_j given all remaining variables, i.e., the lack of edge between X_i and X_j denotes their conditional independence [8].

We will assume a multivariate Gaussian probability density function over $X = \{X_1, ..., X_p\}$:

$$p(\mathbf{x}) = (2\pi)^{-p/2} \det(\Sigma)^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(x-\mu)} \tag{1}$$

where μ is the mean and Σ is the covariance matrix of the distribution, respectively. Without loss of generality we assume that the data is scaled so that $\mu = 0$, hence the purpose is to estimate Σ. Since $det(\Sigma)^{-1} = det(\Sigma^{-1})$, we can now rewrite (1), assuming $C = \Sigma^{-1}$ and $\mu = 0$:

$$p(\mathbf{x}) = (2\pi)^{-p/2} \det(C)^{\frac{1}{2}} e^{-\frac{1}{2}\mathbf{x}^T C \mathbf{x}}. \tag{2}$$

Missing edges in the above graphical model correspond to zero entries in the inverse covariance matrix $C = \Sigma^{-1}$, and vice versa [8], and thus the problem of structure learning for the above probabilistic graphical model is equivalent to the problem of learning the zero-pattern of the inverse-covariance matrix. Note that the inverse of the maximum-likelihood estimate of the covariance matrix Σ (i.e. the empirical covariance matrix $A = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{x}_i$ where $\mathbf{x}_i$ is the i-th sample, $i = 1, ..., n$), even if it exists, does not typically contain any elements that are exactly zero. Therefore an explicit sparsity-enforcing constraint needs to be added to the estimation process.

A common approach is to include as penalty the (vector) l_1-norm of C, which is equivalent to imposing a Laplace prior on C in maximum-likelihood framework [5, 12, 16]. Formally, the entries C_{ij} of the inverse covariance matrix C are assumed to be independent random variables, each following a Laplace distribution

$$p(C_{ij}) = \frac{\lambda_{ij}}{2} e^{-\lambda_{ij}|C_{ij}-\alpha_{ij}|} \tag{3}$$

with zero location parameter (mean) α_{ij} and common scale parameter $\lambda_{ij} = \lambda$, yielding

$$p(C) = \prod_{i=1}^{p} \prod_{j=1}^{p} p(C_{ij}) = (\lambda/2)^{p^2} e^{-\lambda||C||_1}, \tag{4}$$

where $||C||_1 = \sum_{ij} |C_{ij}|$ is the (vector) l_1-norm of C. Then the objective is to find the maximum log-likelihood solution $\arg\max_{C \succ 0} \log p(C|\mathbf{X})$, where $C \succ 0$ denotes that the matrix C is positive definite, $\mathbf{X}$ is the $n \times p$ data matrix, or equivalently, since $p(C|\mathbf{X}) = P(\mathbf{X}|C)P(C)/p(\mathbf{X})$ and $p(\mathbf{X})$ does not include C, to find

$$\arg\max_{C \succ 0} \log P(\mathbf{X}|C)P(C) = \arg\max_{C \succ 0} \log \prod_{i=1}^{n} [\frac{\det(C)^{\frac{1}{2}}}{(2\pi)^{p/2}} e^{-\frac{1}{2}\mathbf{x}_i^T C \mathbf{x}_i}] + \log[(\lambda/2)^{p^2} e^{-\lambda||C||_1}]. \tag{5}$$

We write $\sum_{i=1}^{n} \mathbf{x}_i^T C \mathbf{x}_i = n\mathrm{tr}(AC)$ where tr denotes the trace of a matrix, and $A = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{x}_i$ is the empirical covariance matrix. This yields the following optimization problem considered in [5, 12, 16] (see [16] for the derivation details):

$$\max_{C \succ 0} \ln \det(C) - \mathrm{tr}(AC) - \lambda||C||_1. \tag{6}$$

Herein, we make a more general assumption about $p(C)$, allowing different rows of C (or even different elements) to have different parameters λ_i (or λ_{ij}), i.e., $p(C_{ij}) = \frac{\lambda_i}{2} e^{-\lambda_i |C_{ij}|}$. This reflects our desire to model structured networks with potentially very different node degrees (i.e., row densities in C), and yields

$$p(C) = \prod_{i=1}^{p} \prod_{j=1}^{p} \frac{\lambda_i}{2} e^{-\lambda_i |C_{ij}|} = \prod_{i=1}^{p} \frac{\lambda_i^p}{2^p} e^{-\lambda_i \sum_{j=1}^{p} |C_{ij}|}. \quad (7)$$

Substituting the above into the maximum-loglikelihood problem in the equation 5 yields a general formulation

$$\max_{C \succ 0} \frac{n}{2} [\ln \det(C) - \mathrm{tr}(AC)] - \|C\|_S \quad (8)$$

Here by $\|C\|_S$ we denote the sum of absolute values of the elements of the matrix $S \cdot C$, where $\cdot$ denotes the element-wise product. For example, if S is a product of $\rho = \frac{n}{2}\lambda$ and the matrix of all ones, then the problem reduces to the problem in the equation 6. Note that by allowing the matrix S to have arbitrary nonnegative entries, we automatically include in the formulation the case when the diagonal elements of C are not penalized or when the absolute values of the entries of C are scaled by their estimated value, as it was considered in [16].

The dual of this problem can be written similarly to the dual in [12]

$$\max_{W \succ 0} \{ \frac{n}{2} \ln \det(W) - np/2 : \text{ s.t. } -S \leq \frac{n}{2}(W - A) \leq S \}, \quad (9)$$

where the inequalities involving matrices W, A and S are element-wise. The optimality conditions for this pair of primal and dual problems imply that $W = C^{-1}$ and that $(n/2)W_{ij} - A_{ij} = S_{ij}$ if $C_{ij} > 0$ and $(n/2)W_{ij} - A_{ij} = -S_{ij}$ if $C_{ij} < 0$.

3 The SINCO method

Problem (8) is a special case of a semidefinite programming problem (SDP) [6], which can be solved in polynomial time by interior point methods (IPM). However, as it is well-known, each iteration of an interior point method applied to a semidefinite programming problem of size p requires $O(p^6)$ operations and $O(p^4)$ memory space, which is very costly. Another reason that using an IPM is undesirable for our problem is that an IPM does not produce the sparsity pattern of the solution. The sparsity is recovered in the limit, hence numerical inaccuracy can interfere with the structure recovery.

As an alternative to IPMs, more efficient approaches were developed for problem (8) in [12] and [5]. Both methods are similar in that they are based on applying a block coordinate descent method to the dual of (8). At each iteration only one row (and the corresponding symmetric column) of the dual matrix is optimized, while the rest of that matrix remains fixed. The resulting subproblem is a convex quadratic problem. The difference between the COVSEL method, described in [12], and the *glasso* method in [5] is that COVSEL solves the subproblems via an interior point approach (as second order cone quadratic problems (SOCP)), while *glasso* poses the subproblem as a dual

of the Lasso problem [14] and utilizes LARS [2], an efficient active-set algorithm for solving Lasso. As a result of using an active set approach, the sparsity of the primal matrix is recovered more accurately than with the interior point approach, and the *glasso* method [5] is faster than COVSEL because it takes advantage of that sparsity.[2] A recent row by row (RBR) method for general SDP [17] is based on the same idea of updating one row and column at a time, as *glasso* and COVSEL, but is applied directly to the primal matrix. The resulting subproblem is in general a SOCP, but is likely to be similar to the dual of the subproblem derived for *glasso* in [5]. The application of the RBR method to the problem (8) is a subject of future research and is beyond the scope of this paper. Other gradient-based approaches for problem (8) were suggested (e.g. in [12]), but their performance so far does not exceed those of COVSEL and *glasso*.

Herein, we propose a novel algorithm which can be viewed as a simplified version of the RBR method. We refer to our method as "SINCO" for Sparse INverse COvariance problem. SINCO solves the primal problem directly and also uses coordinate ascent, which naturally preserves the sparsity of the solution. Unlike COVSEL, *glasso* and the general RBR, SINCO only optimizes one diagonal or two (symmetric) off-diagonal entries of the matrix C at each step. The advantages of this approach are that only one or two nonzero entries can be introduced at each step, and also that the solution to each subproblem is available in closed form as a root of a quadratic equation. Computation at each step requires a constant number of arithmetic operation, independent of p. Hence, in $O(p^2)$ operations a potential step can be computed for *all pairs* of symmetric elements (i.e., for all pairs (i, j)). Then the step which provides the *best* function value improvement can be chosen, which is the essence of the greedy nature of our approach. Once the step is taken, the update of the gradient information requites $O(p^2)$ operations. Hence, overall, each iteration takes $O(p^2)$ operations. Note that each step is also suitable for massive parallelization.

In comparison, *glasso* and COVSEL (and RBR) require solution of a quadratic programming problem, whose theoretical and empirical complexity varies depending on the method used, but always exceeds $O(p^2)$. The algorithms apply optimization to each row/column consecutively, hence the greedy nature is lacking. On the other hand, a whole row and column is optimized at each step, thus reducing the overall number of steps. As we will show in our numerical experiments, SINCO, in a serial mode, is comparable to *glasso*, who is orders of magnitude faster than COVSEL [5]. Also, as it is demonstrated by our computational experiments, SINCO leads to a lower false-positive error than *glasso* since it introduces nonzero elements greedily. On the other hand, it may suffer from introducing too few nonzeros and thus missing some of the true positives, especially on denser networks.

Perhaps the most interesting consequence of SINCO's greedy nature is that it reproduces the regularization path behavior while using only one value of the regularization parameter λ. We will discuss this property further in Section 4.

Another advantage of SINCO (as well as of *glasso*) is the ability to efficiently utilize warm starts in various modes. For instance, it is easy to compute a range of solutions for various values of λ, which defines matrix S. One may also use warm starts in

[2] On the other hand, COVSEL is implemented in C with Matlab interface, while *glasso* is a Fortran code with R interface, but not available from Matlab. Hence, for some applications based primarily on Matlab and C, COVSEL has remained the only alternative so far.

computing the leave-one-out error in structure recovery.

3.1 Algorithm description

The main idea of the method is the following: at each iteration, the matrix C is updated by changing one element on the diagonal or two symmetric off-diagonal elements. This implies the change in C that can be written at $C + \theta(e_i e_j^T + e_j e_i^T)$, where i and j are the indices corresponding to the elements that are being changed. The key observation is that, given the matrix $W = C^{-1}$, the exact line search that optimizes the objective function of problem (8) along the direction $e_i e_j^T + e_j e_i^T$ reduces to a solution of a quadratic equation, as we will show below. Hence each such line search takes a constant number of operations. Moreover, given the starting objective value, the new function value on each step can be computed in a constant number of steps. This means that we can perform such line search for all (i, j) pairs in $O(p^2)$ time, which is linear in the number of unknown variables C_{ij}. We then can choose the step that gives the best improvement in the value of the objective function. After the step is chosen, the dual matrix $W = C^{-1}$ and, hence, the objective function gradient, are updated in $O(p^2)$ operations.

Note that the algorithm does not require to maintain the factorization of W or its $(p-1)$-dimensional submatrices, as in *glasso*.

We now describe the method. First, let us consider an equivalent reformulation of the problem (8):

$$\begin{aligned} \max_{C', C''} \quad & \frac{n}{2}[\ln\det(C' - C'') - \operatorname{tr}(A(C' - C''))] - \operatorname{tr}(S(C' + C'')), \\ \text{s. t.} \quad & C' \geq 0,\ C'' \geq 0,\ C' - C'' \succ 0 \end{aligned}$$

For a fixed pair (i, j) consider now the update of C' of the form $C'(\theta) = C' + \theta(e_i e_j^T + e_j e_i^T)$, such that $C' \geq 0$. Let us consider now the objective function as the function of θ.

$$f'(\theta) = \frac{n}{2}(\ln\det(C + \theta e_i e_j^T + \theta e_j e_i^T) + \operatorname{tr}(A(C + \theta e_i e_j^T + \theta e_j e_i^T)) + ||C + \theta e_i e_j^T + \theta e_j e_i^T||_S$$

Similarly, if we consider the update of the form $C''(\theta) = C'' + \theta(e_i e_j^T + e_j e_i^T)$ such that $C'' > 0$, the objective function becomes

$$f''(\theta) = \frac{n}{2}(\ln\det(C - \theta e_i e_j^T - \theta e_j e_i^T) + \operatorname{tr}(A(C - \theta e_i e_j^T + \theta e_j e_i^T)) + ||C + \theta e_i e_j^T + \theta e_j e_i^T||_S$$

The method we propose works as follows:

Algorithm 1

0. Initialize $C' = I,\ C'' = 0,\ W = I$
1. Form the gradient $G' = \frac{n}{2}(W - A) - S$ and $G'' = -S - \frac{n}{2}(W + A)$
2. For each pair (i, j) such that

(i) $G'_{ij} > 0, C''_{ij} = 0,$ compute the maximum of $f'(\theta)$ for $\theta > 0$.

(ii) $G'_{ij} < 0, C'_{ij} > 0,$ compute the maximum of $f'(\theta)$ for $\theta < 0$ subject to $C' \geq 0$

(iii) $G''_{ij} > 0, C'_{ij} = 0,$ compute the maximum of $f''(\theta)$ for $\theta > 0$.

(iv) $G''_{ij} < 0, C''_{ij} > 0,$ compute the maximum of $f''(\theta)$ for $\theta < 0$ subject to $C'' \geq 0$.

3. Choose the step which provides the maximum function improvement. If relative function improvement is below tolerance, then Exit.
4. Update W^{-1} and the function value and repeat.

As mentioned above, the maximum of the one-dimensional function in Step 3 is available in closed form. Indeed, consider the step $\bar{C}' = C' + \theta(e_i e_j^T + e_j e_i^T)$. Let us assume that $\theta > 0$ and that $C''_{ij} = 0$, which implies that we can write the step as $\bar{C} = C + \theta(e_i e_j^T + e_j e_i^T)$, since the (i, j) and (j, i) elements do not become zero for any such step.

The inverse W, then, is updated, according to the Sherman-Morrison-Woodbury formula $(X + ab^T)^{-1} = X^{-1} - X^{-1}a(1 + b^T X^{-1}a)^{-1}b^T X^{-1}$, as follows

$$\begin{aligned}
\bar{W} &= W - \theta(\kappa_1 W_i W_j^T + \kappa_2 W_i W_i^T + \kappa_3 W_j W_j^T + \kappa_1 W_j W_i^T) \\
\kappa_1 &= -(1 + \theta W_{ij})/\kappa \\
\kappa_2 &= \theta W_{jj}/\kappa \\
\kappa_3 &= \theta W_{ii}/\kappa. \\
\kappa &= \theta^2(W_{ii} * W_{jj} - W_{ij}^2) - 1 - 2\theta W_{ij}
\end{aligned}$$

Let us now compute the objective function as the function of θ.

$$f(\theta) = \frac{n}{2}(\ln\det(C + \theta e_i e_j^T + \theta e_j e_i^T) + \operatorname{tr}(A(C + \theta e_i e_j^T + \theta e_j e_i^T)) + ||C + \theta e_i e_j^T + \theta e_j e_i^T||_S$$

We use the following property of the determinant: $\det(X + ab^T) = det(X)(1 + b^T X^{-1}a)$ and the Scherman-Morisson-Woodbury formula. We have

$$\begin{aligned}
& \det(C + \theta e_i e_j^T + \theta e_j e_i^T) = \det(C + \theta e_j e_i^T)(1 + \theta e_j^T (C + \theta e_j e_i^T)^{-1} e_i) \\
= \ & \det(C)(1 + \theta e_i^T C^{-1} e_j)(1 + \theta e_j^T C^{-1} e_i - \theta^2 e_j^T C^{-1} e_j (1 + \theta e_i^T C^{-1} e_j)^{-1} e_i^T C^{-1} e_i) \\
= \ & \det(C)(1 + 2\theta e_i^T C^{-1} e_j + (\theta e_j^T C^{-1} e_i)^2 - \theta^2 e_i^T C^{-1} e_i e_j^T C^{-1} e_j).
\end{aligned}$$

Given the dual solution $W = C^{-1}$, and recalling that W and A are symmetric, but S is not necessarily so, we can write the above as

$$\det(C + \theta e_i e_j^T + \theta e_j e_i^T) = \det(C)(1 + 2\theta W_{ij} + \theta^2(W_{ij}^2 - W_{ii}W_{jj}).$$

Then the change in the objective function is

$$f(\theta) - f = \frac{n}{2}(\ln(1 + 2\theta W_{ij} + \theta^2(W_{ij}^2 - W_{ii}W_{jj})) - 2A_{ij}\theta - S_{ij}\theta - S_{ji}\theta,$$

the last term being derived from the fact that $C_{ij} + \theta$ and $C_{ji} + \theta$ remain positive.

Let us now consider the derivative of the objective function with respect to θ

$$f'(\theta) = \frac{nW_{ij} + n\theta(W_{ij}^2 - W_{ii}W_{jj})}{\theta^2(W_{ij}^2 - W_{ii}W_{jj}) + 1 + 2\theta W_{ij}} - nA_{ij} - S_{ij} - S_{ji}.$$

To find the maximum of $f(\theta)$ we need to find $\theta > 0$ for which $f'(\theta) = 0$. Letting a denote $W_{ii}W_{jj} - W_{ij}^2$, this condition can be written as:

$$nW_{ij} - nA_{ij} - S_{ij} - S_{ji} - (na + 2W_{ij}(nA_{ij} + S_{ij} + S_{ji})\theta + a(nA_{ij} + S_{ij} + S_{ji})\theta^2 = 0.$$

To find the value of θ for which the derivative of the objective function equals zero we need to solve the above quadratic equation

$$ab\theta^2 - (na + 2W_{ij}b)\theta + nW_{ij} - b = 0, \tag{10}$$

where $a = W_{ii}W_{jj} - W_{ij}^2$ and $b = nA_{ij} + S_{ij} + S_{ji}$. Notice that a is always nonnegative, because matrix W is positive definite, and it equals zero only when $i = j$. We know that at $\theta = 0$ $f'(0) > 0$. Let us investigate what happens when θ grows. The discriminant of the quadratic equation is

$$\begin{aligned} D &= (na + 2W_{ij}b)^2 - 4ab(nW_{ij} - b) = (na)^2 + 4nW_{ij}ab + 4W_{ij}^2b^2 - 4abnW_{ij} + 4ab^2 \\ &= (na)^2 + 4b^2W_{ii}W_{jj} > 0, \end{aligned}$$

hence the quadratic equation always has a solution. At $\theta = 0$ the quadratic function equals

$$nW_{ij} - nA_{ij} - S_{ij} - S_{ji} = G'_{ij} + G'_{ji} > 0,$$

Now let us again consider the derivative $f'(\theta)$. At $\theta = 0$ we know that the derivative is positive. We also know that the denominator

$$\theta^2(W_{ij}^2 - W_{ii}W_{jj}) + 1 + 2\theta W_{ij} = (1 + \theta W_{ij})^2 - \theta^2 W_{ii}W_{jj}$$

is positive when $\theta = 0$ and is equal to zero when $\theta = \theta_{max} = 1/(\sqrt{W_{ii}W_{jj}} - W_{ij}) > 0$. The function $f(\theta)$ approaches negative infinity when $\theta \to \theta_{max}$, hence so does $f'(\theta)$. This implies that $f'(\theta)$ has to reach the value zero for some $\theta \in (0, \theta_{max})$. Hence the quadratic equation (10) has one positive solution in this interval. This solution gives us the maximum of $f(\theta)$ and hence the length of the step along the direction $e_i e_j^T + e_j e_i^T$.

The objective function value is easy to update using the formula

$$det(C' - C'' + \theta(e_i e_j^T + e_j e_i^T)) = det(C' - C'')(1 + 2\theta W_{ij} - \theta^2 a));$$

Let us consider the negative step along the direction $e_i e_j^T + e_j e_i^T$ when $C'_{ij} > 0$. The derivations are exactly as above, except for we are now looking for solution $\theta < 0$. As discussed above, the term under the logarithm

$$\theta^2(W_{ij}^2 - W_{ii}W_{jj}) + 1 + 2\theta W_{ij} = (1 + \theta W_{ij})^2 - \theta^2 W_{ii}W_{jj}$$

is positive when $\theta = 0$ and is also equal to zero when $\theta = \theta_{min} = -1/(\sqrt{W_{ii}W_{jj}} + W_{ij}) < 0$. The derivative of $f(\theta)$ at $\theta = 0$ is negative, this time (which is why we are considering a negative step, in the first place), which means that there exists a $\theta \in (\theta_{min}, 0)$ for which this derivative is zero, hence the quadratic equation (10) has a negative solution. This negative solution $\theta_- < 0$ determines the length of the step in the direction $-e_i e_j^T - e_j e_i^T$. It is important to note that the length of the step cannot exceed the value C'_{ij}, hence the final step length is computed as $\max(\theta_-, -C'_{ij})$.

The other two possible steps listed in Step 3 can be analyzed analogously, the main difference being the sign before the terms nA_{ij}, S_{ij} and S_{ji} in the case of the step that updates C''.

Each step can be computed by a constant number of arithmetic operations, hence to find the step that provides the largest function value improvement it takes $O(p^2)$ operations - the same amount of work (up to a constant) that it takes to update W and the gradient after one iteration. Hence the overall per-iteration complexity is $O(p^2)$. Moreover, this algorithms lends itself readily to massive parallelization. Indeed, at each iteration of the algorithm the step computation for each (i, j) pair can be parallelized and the procedure that updates W involves simply adding to each element of W a function that involves only two rows of W. Hence the updates can be also done in parallel and in very large scale cases the matrix W can also be stored in a distributed manner. The same is true of the storage of matrices A and S (assuming that S needs to be stored, that is not all elements of S are the same), while the best way to store C' and C'' matrices may be in sparse form.

The convergence of the method follows from the convergence of a block-coordinate descent method on a strictly convex objective function. The only constraints are box constraints (nonnegativity) and they do not hinder the convergence. In fact we can view our method as a special case of the row by row (RBR) method for SDP described in [17]. In the case of SINCO we extensively use the fact that each coordinate descent step is cheap and, unlike the RBR algorithm we select the next step based on the best function value improvement. On the other hand, we maintain the inverse matrix W, which RBR method does not. However, none of these differences prevent the convergence result for RBR in [17] to apply to our method. Hence the convergence to the optimal solution holds for SINCO.

3.2 Numerical experiments setting

In order to test structure-reconstruction accuracy, we performed experiments on several types of synthetic problems. (Note that, unlike prediction of an observed variable,

structure reconstruction accuracy is harder to test on "real" data since (1) the "true" structure my not be available and (2) known links in "real" networks (e.g., known gene networks) may not necessarily correspond to links in the underlying Markov net.) We generated uniform-random, as well as semi-realistic, structured "scale-free" networks (following power-law degree distribution), that are known to better model various biological, ecological, social, and other real-life networks [3]. The scale-free (SF) networks were generated using the preferential attachment (Barabasi-Albert) model [3] [3].

We generated networks with various density, measured by the % of non-zero off-diagonal entries. For each density level, we generated the networks over p variables, that defined the structure of the "ground-truth" inverse covariance matrix, and for each of them, we generate matrices with random covariances corresponding to the non-diagonal non-zero entries (while maintaining positive definiteness of the resulting covariance matrix). We then sampled n instances, with the value of n depending on the experiment, from the corresponding multivariate Gaussian distribution over p variables.

4 Regularization path

One of the main challenges in sparse inverse covariance selection is the proper choice of the weight matrix S in (8). Typically the matrix S is chosen to be a multiple of the matrix of all ones. The multiplying coefficient is denoted by λ and is called the "regularization parameter". Hence the norm $\|C\|_S$ in (8) reduces to $\lambda\|C\|_1$ (in the vector-norm sense) as in (6). Clearly, for large values of λ as $\lambda \to \infty$ the solution to (8) is likely to be very sparse and eventually diagonal, which means that no structure recovery is achieved. On the other hand, if λ is small as $\lambda \to 0$, the solution C is likely to be dense and eventually approach A^{-1}, and, again, no structure recovery occurs. Hence exploration of a regularization path is an integral part of the sparse inverse covariance selection.

Typically, problem (8) is solved for several values of λ in a predefined range and the best value, according to some criteria, is selected. The reason only a scalar parameter λ is usually considered, is because it is expensive to explore solutions along a multi-dimensional grid.

Since the ground truth for the structure is unknown, except for artificial cases, the selection of lambda based on the recovery of true structure is not possible. There are several methods of selecting a good value of the regularization parameter. Some are based on cross-validation or the prediction error of the resulting Gaussian model. The cross-validation approach to λ selection often leads to overfitting with the chosen value of λ being small and the resulting C being too dense. Other methods, such as in [12] choose one specific value of λ based on theoretical derivations aimed at asymptotic consistency, but in practical settings, this value tends to be too large (too conservative, aiming at lowering the false-positive rate), and thus the resulting matrix C is too sparse. More recently, a Bayesian approach to λ selection was proposed in [1, 13]; the idea is to treat the regularization parameter as a random variable (or a

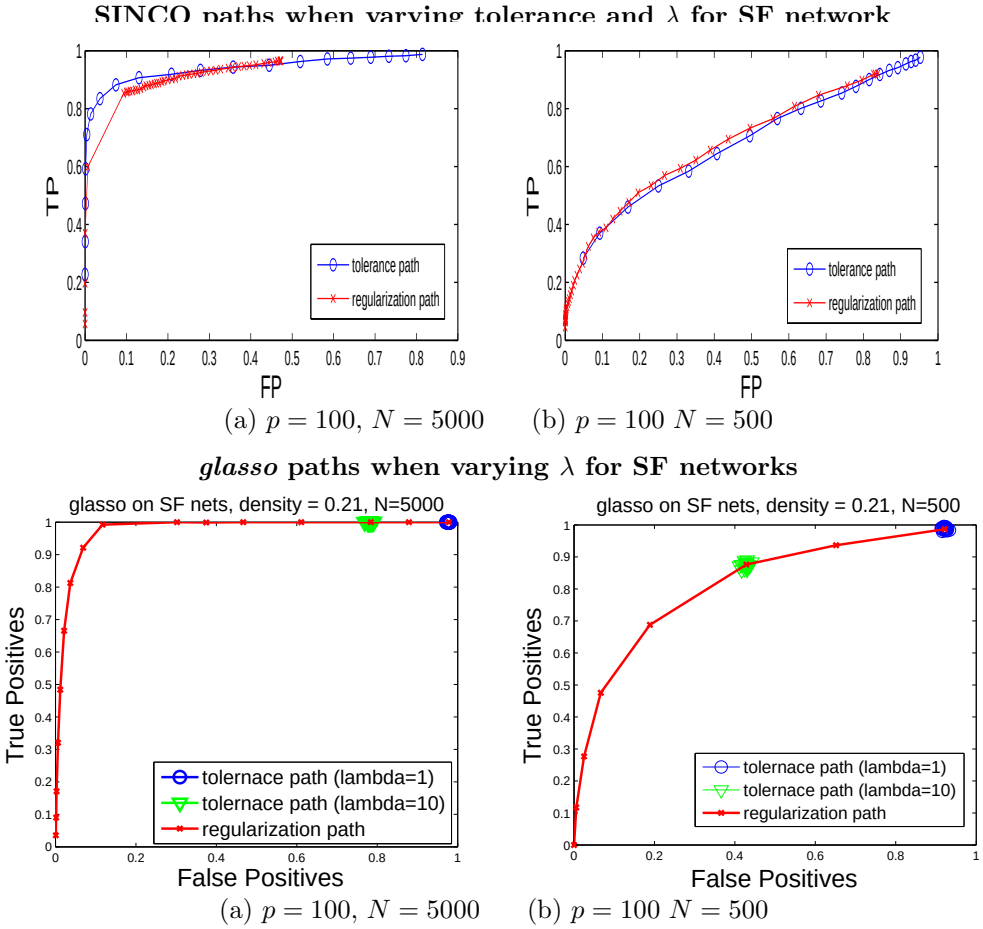
[3] We used the open-source Matlab code available at http://www.mathworks.com/matlabcentral/fileexchange/11947.

vector of random variables in the general case of vector-λ formulation) and search for a maximum a posteriori probability (MAP) solution for both C and λ using an alternating-minimization approach.

In this section, we concentrate on computing the entire regularization path (or some parts of it) rather than on specific λ selection methods. SINCO method is very well-suited for the efficient regularization path computation, since it directly exploits warm starts. When λ is relatively large, a very sparse solution can be obtained quickly. This solution can be used as a warm start to the problem with a smaller value of λ and, if the new value of λ is not much smaller than the previous value, then the new solution is typically obtained in just a few iterations, because the new solution has only a few extra nonzero elements. Warm starts can also be used to initiate different subproblems for leave-one-out validation approach, where the stability of the structure is measured over a collection of data subsets, each of which is composed of all data samples but one. Since each leave-one-out subproblem differers from another one by a rank-two update of matrix A, and since the resulting nonzero pattern is expected to be not very different, the solution to one subproblem can be an efficient warm start for another subproblem.

Typically the output of the regularization path is evaluated via the ROC curves showing the trade-off between the number of true positive (TP) element recovered and the number of false positive (FP) elements. Producing better curves (where the number of TPs rises fast relative to FPs) is usually an objective of any method that does not focus on specific λ selection. An interesting property of SINCO is that it introduces nonzero entries to the matrix C as it progresses. Hence, if we use looser tolerance and stop the algorithm early, then we will observe fewer nonzero entries, hence a sparse solution for any specific value of λ. What we observe, as seen in Figures 1 and 2, is that if we apply SINCO to problem (8) with ever tighter tolerance then the ROC curves obtained from the tolerance solution path match the ROC curves obtained from the regularization path. Here we show several examples of the matching ROC curves for various random networks. We use a randomly generated structured (scale-free) network that is 21% dense and a randomly generated unstructured network, 3% dense. We use $p = 100$ and two instances: $N = 500$ and $N = 5000$. We applied SINCO to one instance of problem (8) with $\lambda = 0.01$ (almost no regularization) with a range of stopping tolerances from 10^{-4} to 10^{-7}. The ROC curve of that path is presented by a line with "o"s. We also applied SINCO with fixed tolerance of 10^{-6} to a range of λ values from 300 to 0.01. The corresponding ROC curves are denoted by lines with "x"s. We can see that the ROC curve of the regularization path for the given range of values of λ is somewhat less steep than that of the tolerance path, but the curves are still very close in the area where they overlap. For baseline we also present the ROC curve of the regularization path computed by *glasso*, which is very similar to the SINCO's ROC curves. Note that changing tolerance does not have the same affect on *glasso* as it does on SINCO. The number of TP and FP does not change noticeably with increasing tolerance. This is due to the fact that the algorithm in *glasso* updates a whole row and a column of C at each iteration while it cycles through the rows and columns, rather than selecting the updates in a greedy manner.

Our observation imply that SINCO can be used to greedily select the elements of graphical model until the desired trade-off between FPs and TPs is achieved. In the limit SINCO solves the same problem as *glasso* and hence the limit number of the



(a) $p = 100$, $N = 5000$ (b) $p = 100$ $N = 500$

(a) $p = 100$, $N = 5000$ (b) $p = 100$ $N = 500$

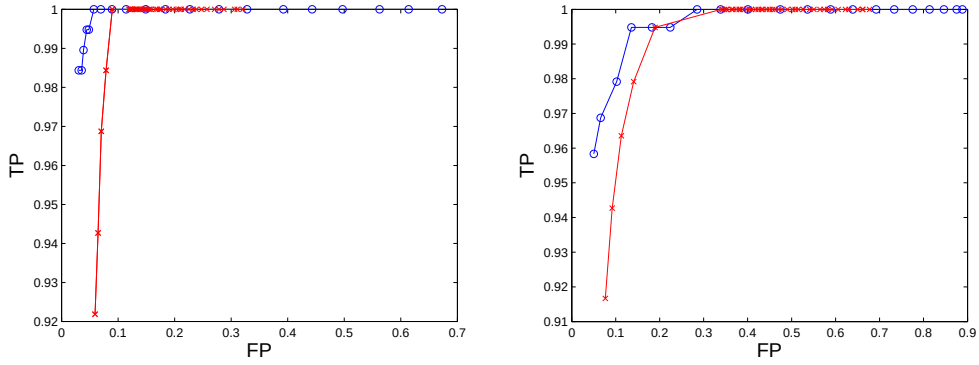
Figure 1: Scale-free networks: SINCO and *glasso* paths when varying tolerance and λ.

true and false positives is dictated by the choice of λ. But since the real goal is to recover true nonzero structure of the covariance matrix, it is not necessary to solve problem (8) accurately. For the purpose of recovering a good TP/FP ratio one can apply SINCO method, without the adjustments to λ.

We should note that computing the regularization path presented in our experiments is typically more efficient in terms of CPU time than computing the tolerance path; the largest computational cost lies in computing the tail of the path for smaller tolerances. On the other hand, the tolerance path appears to be more precise and exhaustive, in terms of possible FP/TP tradeoffs. It is also important to note that the entire tolerance path is automatically produced as a sequence of iterates produced by SINCO, while the regularization path can only be computed as a sequence of solutions for a given set of values of λ.

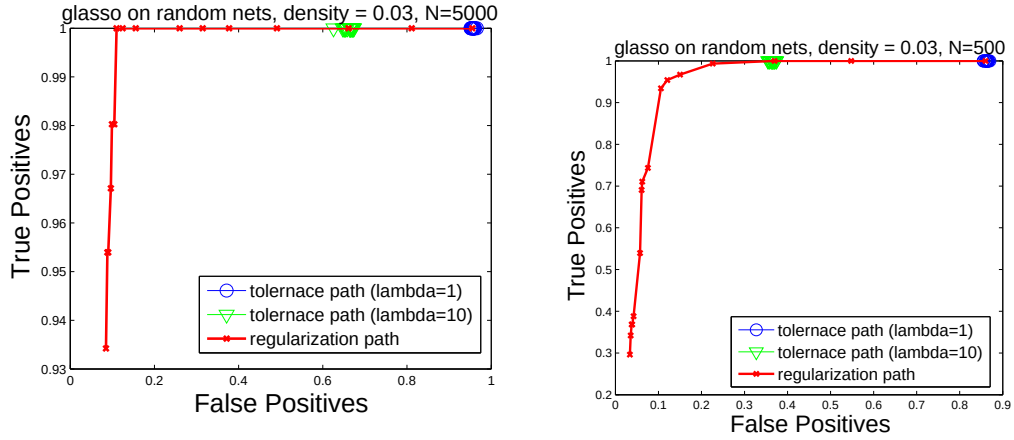
The main conclusion we can draw here is that SINCO has the ability to select the nonzeros in the inverse covariance matrix (or the links in the graph) according to their

SINCO paths when varying tolerance and λ for a random network



(c) $p = 100$, $N = 5000$ (d) $p = 100$ $N = 500$

***glasso* path when varying λ for random networks**



(c) $p = 100$, $N = 5000$ (d) $p = 100$ $N = 500$

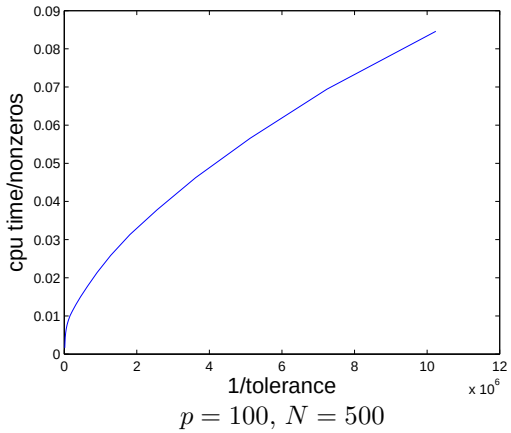
Figure 2: Random networks: SINCO and *glasso* paths when varying tolerance and λ.

"importance".

5 Empirical complexity

Here we will discuss the empirical dependence of the runtime of the SINCO algorithm on the choice of stopping tolerance and the problem size p. We also investigate the effect increasing n has on the results produced by SINCO and *glasso*. Both methods were executed on Intel Core 2Duo T7700 processor (2.40GHz); note, however, that *glasso* is based on well-tuned LARS software which is Fortran code, while SINCO a straight-forward C++ implementation of the algorithm in Section 3 with Matlab interface.

First, we apply our algorithm to the 21%-dense scale-free network that we used in



$p = 100$, $N = 500$

Figure 3: SINCO's CPU time per nonzero with increasing tolerance.

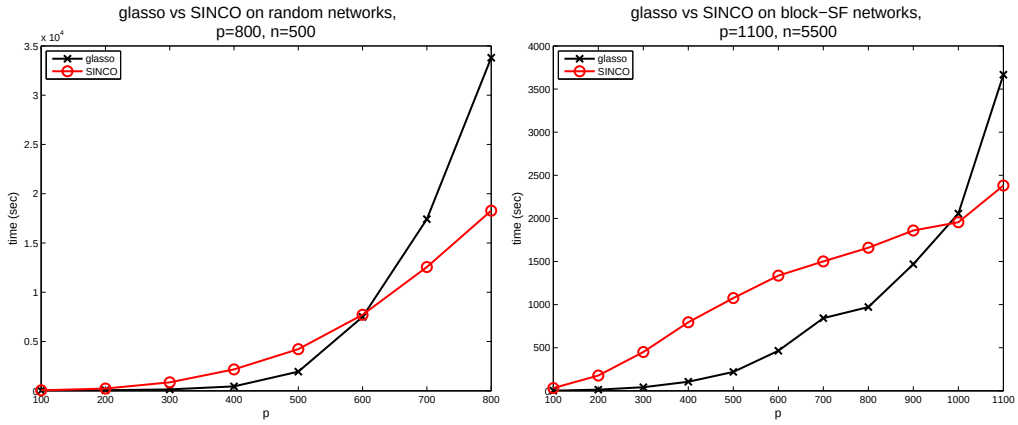
the previous section. When computing the tolerance path, we can measure the CPU time for various tolerance levels. Note that the number of nonzeros in the solution grows as the tolerance decreases. Hence, we compare the CPU time per nonzero in the solution instead of the pure CPU time. In Figure 3 we show how the ratio of CPU time and the number of nonzeros depends on the inverse of the tolerance. As we see, the dependence is almost linear.

Now let us consider the situation when p increases. If together with p the number of nonzeros in the true inverse covariance also increases, then to obtain a comparable problem we need to increase n accordingly. Increasing n, in turn, affects the contribution of λ, since the problem scaling changes. Here we chose to consider the following two simple settings, where we can account for these effects. In the first setting, we increase p while keeping the number of the off-diagonal nonzero elements in the randomly generated unstructured network constant (around 300). We do not, therefore, increase n or λ. The CPU time necessary to compute the entire path for λ ranging from 300 to 0.01 is plotted for $p = 100$, 200, 300, 500, 800 and 1000 in the second plot of Figure 4.

In the second case, we generated block-diagonal matrices of sizes $p = 100$, 200, 300, 500, 800, 1000, with 100×100 diagonal blocks, each of which equals the inverse covariance matrix of a 21%-dense structured (scale-free) network from the previous section. Since the number of nonzero elements grows linearly with p, we increased n and the appropriate range of λ linearly as well. The CPU time for this case is shown in the last plot of Figure 4.

Figure 4 shows that the CPU time (in seconds) for SINCO scales up slower than that of *glasso*, with increasing number of variables, p. The reason for the difference in scaling rates is evident in the ROC curves in Figures 5 and 6, which demonstrate that, for similarly high true-positive rate, *glasso* tends to have much higher false-positive rate than SINCO, thus producing a less sparse solution, overall.

As we can see, the CPU time of SINCO grows moderately with increasing p. We know that each iteration takes $O(p^2)$ operations. Overall, we observed that the number



(a) Random (b) Scale-free

Figure 4: CPU time comparison: SINCO vs *glasso* on (a) random networks ($N = 500$, fixed range of λ) and (b) scale-free networks (density 21%, N and λ scaled by the same factor with p, $N = 500$ for $p = 100$).

of iterations depends roughly linearly on the number of nonzeros in the solution. The coefficient of this linear dependence is determined by the tolerance level. Note that with parallelization the per-iteration cost has a potential of being significantly reduced.

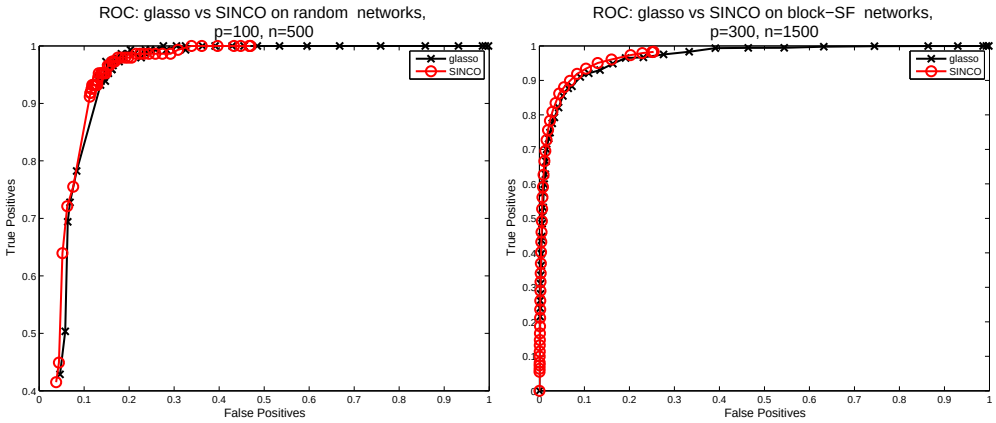
Finally, we investigate the behavior of SINCO for a fixed value of p as n grows. In this setting, we expect to obtain larger TP values and smaller FP error with increasing n. The consistency result in [16] suggests that for our formulation, to obtain an accurate asymptotic structure recovery, we should pick λ that grows with n, but so that its growth is slower than $\sqrt{n}$.

Here we use $\lambda = \log_{10}(n)$. We again apply our algorithm and *glasso* to the 21%-dense scale-free networks with $p = 100$. In Figures 7 and 8 we show the how the value of TP and FP returned by the two algorithms changes with growing n (note that λ is kept fixed for each value of n). We observe that SINCO achieves in the limit nearly 0% false-positive error and nearly 100% true-positive rate, while *glasso*'s FP error grows with increasing n. This result is, again, a consequence of the greedy approach utilized by SINCO. The CPU time taken by *glasso* is, again, less than the SINCO's CPU time, but the latter is still reasonable and also reduces as n grows.

We note that selecting $\lambda = \log_{10}(n)$ is not a crucial choice. Using constant value of λ produces similar results, although the consistency result in [16] does not apply in this case.

6 Results on fMRI data

Here we describe the results of applying SINCO to a real-life data, where the "ground truth" network structure was not available; thus, we could not measure the structure reconstruction accuracy, and instead evaluated the prediction accuracy of the resulting Markov networks. We used fMRI data for mind-state prediction problem described



(a) Random ($p = 100$) (b) scale-free ROC ($p = 300$)

Figure 5: ROC curves for SINCO vs *glasso* on (a) random networks ($N = 500$, fixed range of λ) and (b) scale-free networks (density 21%, N and λ scaled by the same factor with p, $N = 500$ for $p = 100$).

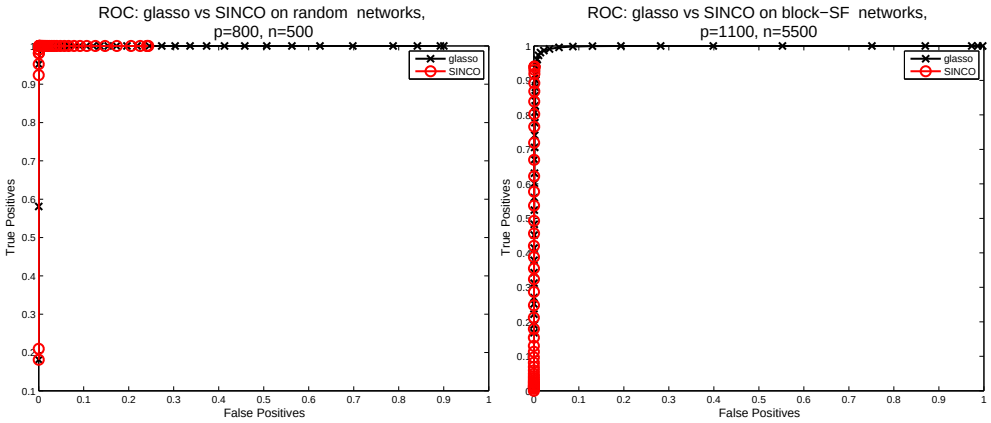
in [11][4]. The data consists of a series of trials in which the subject is being shown either a picture (+1) or a sentence (−1). Our dataset consists from 1700 to 2200 features, dependent on a particular subject, and 40 samples, where half of the samples correspond to the picture stimulus (+1) and the remaining half correspond to sentence stimulus (-1). (One sample corresponds to the averaged fMRI image over 6 scans sequentially taken while a subject is presented with a particular stimulus).[5]. We used leave-one-out cross-validation, and report average results over 40 cross-validation folds, each corresponding to one sample left out for testing, and the remaining 39 used for training.

For each class $Y = \{-1, 1\}$, we learn a sparse Markov Net model that provides us with an estimate the Gaussian conditional density $p(\mathbf{x}|y)$, where $\mathbf{x}$ is the feature (voxel) vector; on the test data, we choose the most-likely class label $\arg\max_y p(\mathbf{x}|y)P(y)$ for each unlabeled test sample $\mathbf{x}$.

Figures 9-11 show the results of comparing SINCO versus COVSEL for 3 subjects in the above study. We compare with COVSEL in this section because it is the other available Matlab implementation, and although COVSEL was shown to be slower than *glasso*, the objective here was rather to compare the prediction accuracy of the two approaches. We observe that SINCO produces much sparser solution that *glasso*, as shown in the previous sections, and than COVSEL, as shown here. What we want to demonstrate now is that the solutions obtained by SINCO are just as accurate as those obtained by COVSEL (and hence *glasso* since they are different implementations of the same approach). We consistently observe that (1) SINCO produces classifiers that are equally (or more) accurate that those produced by COVSEL (Figures 9c-11c), but

[4]For more details, see the StarPlus website http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-81/www/.

[5]As suggested by the provided documentation in the StarPlus web-site we have used the following regions of interest: 'CALC', 'LIPL', 'LT', 'LTRIA', 'LOPER', 'LIPS', 'LDLPFC'. The subjects that are considered herein are those numbered 04847, 04820 and 05680.



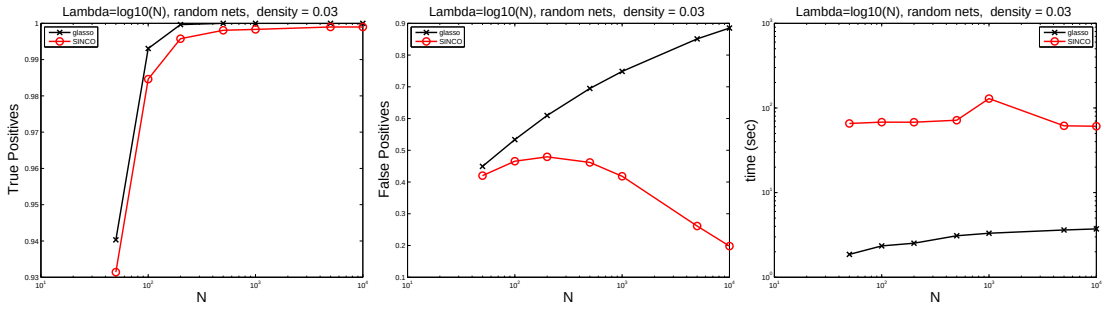
(a) Random ($p = 800$) (b) scale-free ROC ($p = 1100$)

Figure 6: ROC curves for SINCO vs *glasso* on (a) random networks ($N = 500$, fixed range of λ) and (b) scale-free networks (density 21%, N and λ scaled by the same factor with p, $N = 5500$ for $p = 1100$).

much faster (Figures 9b-11b) and using much sparser Markov Net models (Figures 9a-11a), which suggests that COVSEL learns many links that are not essential for discriminative ability of the classifier. It is interesting to note that both Markov Net classifiers are competitive with, and often more accurate than the state-of-art SVM classifier (Figures 9c-11c).

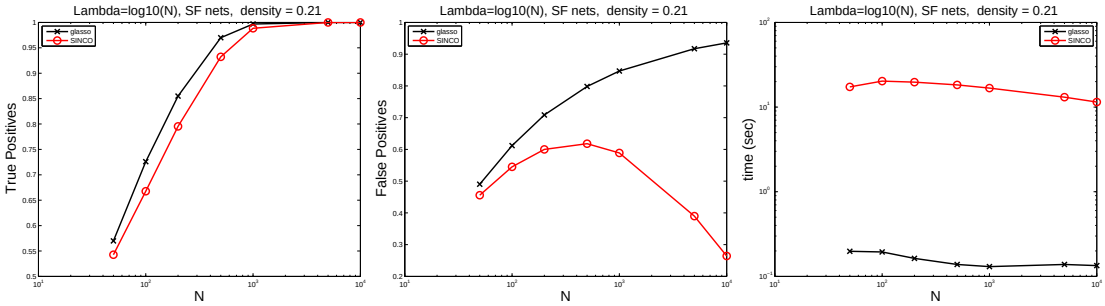
7 Other applications

Similarly to the row-by-row algorithm in [17], our algorithm can be used to solve other semidefinite programming problems. However, to take advantage of the special properties of SINCO algorithm, the positive semidefinite solution matrix must be



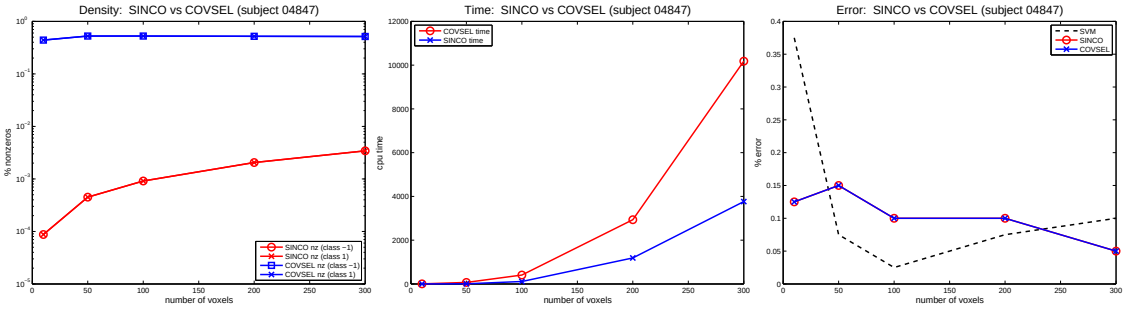
(a) TP (b) FP (c) CPU time

Figure 7: SINCO and *glasso* accuracy with growing n on 20 random networks ($p = 100$, density 3%).



(a) TP (b) FP (c) CPU time

Figure 8: SINCO and *glasso* accuracy with growing n on 25 scale-free networks ($p = 100$, density 21%).



(a) density (b) time (c) classification error

Figure 9: SINCO vs. COVSEL on fMRI data.

sparse. In many specific examples of semidefinite programming problems this is often the case for the dual formulation.

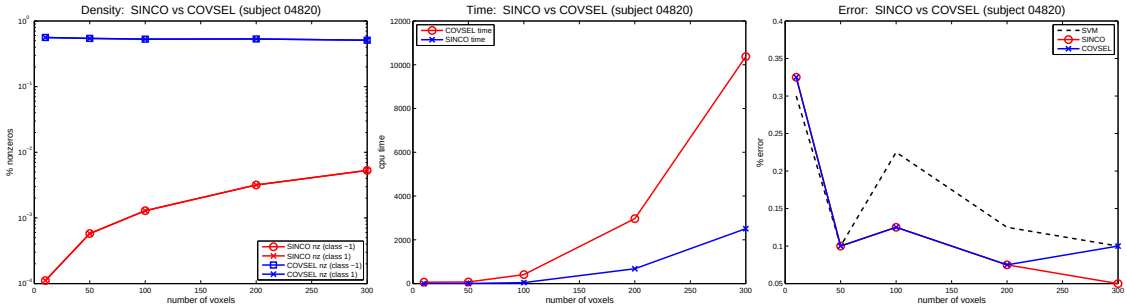
Consider for instance, the matrix completion SDP relaxation [4]. Given a sparse $n \times m$ matrix M, the primal problems is

$$\begin{aligned} \min \quad & \mathrm{tr}(X) \\ \text{s.t.} \quad & X = \begin{bmatrix} W_1 & M^T \\ M & W_2 \end{bmatrix} \succeq 0. \end{aligned}$$

The dual problem is then

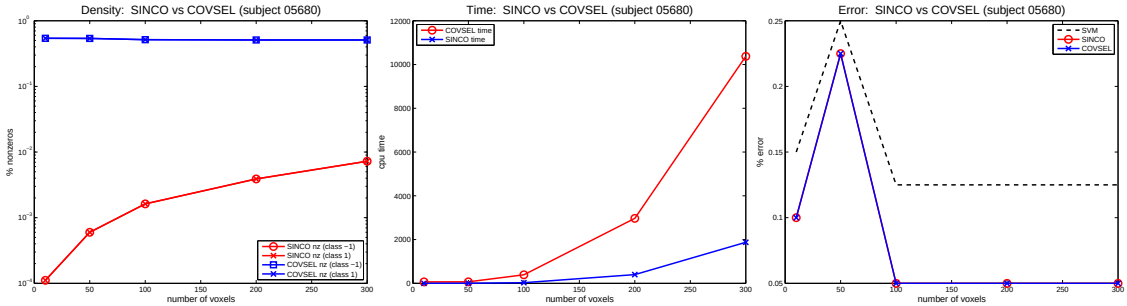
$$\begin{aligned} \min \quad & \sum_{i=1,j=1}^{n,m} M_{ij} Y_{ij} \\ \text{s.t.} \quad & S = \begin{bmatrix} I_{n\times n} & Y^T \\ Y & I_{m\times m} \end{bmatrix} \succeq 0 \\ & Y_{ij} = 0, \ \text{if} M_{ij} = 0. \end{aligned} \tag{11}$$

To be able to apply SINCO we include the positive semidefiniteness constraint into



(a) density (b) time (c) classification error

Figure 10: SINCO vs. COVSEL on fMRI data.



(a) density (b) time (c) classification error

Figure 11: SINCO vs. COVSEL on fMRI data.

the objective function via the $\ln\det$ barrier term with a barrier parameter $\mu > 0$.

$$\begin{aligned} \min \quad & -\mu \ln\det S + \sum_{i=1,j=1}^{n,m} M_{ij} Y_{ij} \\ \text{s.t.} \quad & S = \begin{bmatrix} I_{n\times n} & Y^T \\ Y & I_{m\times m} \end{bmatrix} \\ & Y_{ij} = 0, \ \text{if} M_{ij} = 0. \end{aligned} \tag{12}$$

The smaller the value of μ is, the closer the optimal solution gets to the optimum of (11). We can see that if M is sparse then so is the dual solution S. Unlike the case of the inverse covariance problem, the sparsity pattern of S is given here, hence it is satisfied by all solutions for any μ. This puts additional constraints on the problem and the progress of the coordinate descent algorithm in this case can slow down when μ is small. The same issue arises when any other block-coordinate descent approach is used for positive semidefinite problems; however, when the algorithm is applied to the primal problem, the algorithm simply stalls for small values of μ and no further improvement to the solution can be obtained. In that case the best obtained solution is often acceptable. When applied to the dual problem, however, the algorithm does not improve the dual solution, but the primal solution infeasibility can actually deteriorate

with $\mu \to 0$ if the dual optimality is not achieved. Hence, one may want to stop the optimization before μ gets too small. In the matrix completion case this may compromise obtaining the original goal of obtaining the low-rank primal solution.

Although from the theoretical perspective SINCO will converge to the optimal solution of (12) for any $\mu > 0$, by the convergence theorem for the row-by-row method in [17], the time it takes to converge may be unacceptably large.

In the case of inverse covariance, the low-rank solution is not desirable and the $\ln \det$ term appears in the objective without a barrier parameter. The algorithm is applied directly to the primal problem, hence, while the solution may fail to improve in certain cases, it will not deteriorate.

We have applied SINCO to randomly generated matrix completion instances and obtained primal solutions within $10.e-3$ accuracy. However, it is unclear if the method can be competitive for this case of semidefinite problems and other sparse dual SDPs. This is the subject of future research.

References

[1] N. Bani Asadi, I. Rish, K. Scheinberg, D. Kanevsky, and B. Ramabhadran. A MAP Approach to Learning Sparse Gaussian Markov Networks. In *Proc. of ICASSP 2009*. April 2009.

[2] I. Johnstone B. Efron, T. Hastie and R. Tibshirani. Least angle regression. *Annals of Satistics*, 32(2):407–499, 2004.

[3] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[4] E. J. Candes and B. Recht. Exact matrix completion via convex optimization. *Foundation of Comp. Math*, (to appear), 2009.

[5] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 2007.

[6] R. Saigal H. Wolkowicz and eds. L. Vanenberghe. *Handbook of Semidefinite Programming*. Kluwer Academic Publoshers, 2000.

[7] D. Heckerman. A tutorial on learning Bayesian networks, Tech. Report MSR-TR-95-06. *Microsoft Research*, 1995.

[8] S. Lauritzen. *Graphical Models*. Oxford University Press, 1996.

[9] N. Meinshausen and P. Buhlmann. High dimensional graphs and variable selection with the Lasso. *Annals of Statistics*, 34(3):1436–1462, 2006.

[10] Nicolai Meinshausen and Peter Buehlmann. Stability selection, 2008.

[11] T.M. Mitchell, R. Hutchinson, R.S. Niculescu, F. Pereira, X. Wang, M. Just, and S. Newman. Learning to decode cognitive states from brain images. *Machine Learning*, 57:145–175, 2004.

[12] O.Banerjee, L. El Ghaoui, and A. d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *Journal of Machine Learning Research*, 9:485–516, March 2008.

[13] K. Scheinberg, N. Bani Asadi, and Irina Rish. Sparse MRF Learning with Priors on Regularization Parameters. Technical Report RC24812, IBM T.J. Watson Research Center, 2009.

[14] R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.

[15] M. Wainwright, P. Ravikumar, and J. Lafferty. High-Dimensional Graphical Model Selection Using ℓ_1-Regularized Logistic Regression. In *NIPS 19*, pages 1465–1472. 2007.

[16] M. Yuan and Y. Lin. Model Selection and Estimation in the Gaussian Graphical Model. *Biometrika*, 94(1):19–35, 2007.

[17] S. Ma Z. Wen, D. Goldfarb and K. Scheinberg. Row by row methods for semidefinite programming. Technical Report submitted, Department of IEOR, Columbia University, 2009.