Parallelization of SINCO Algorithm

Aykut Bulut, Murat Mut

December 22, 2011

Abstract

Sparse inverse covariance selection problem is encountered in many practical applications. SINCO is an algorithm for the sparse inverse covariance problem. The literature includes the computational experiments that explores the computational capabilities of the algorithm and its comparison with the *glasso* and *COVSEL*. SINCO is an algorithm that has parallelization potential. This study explores the gains from parallelization of SINCO.

Keywords: Sparse inverse covariance selection, glasso, COVSEL, parallelization

1 Introduction

In many practical applications of statistical learning the objective is to discover meaningful interactions among the variables. Some examples come from

- Gene networks.
- Discovery of functional brain connectivity patterns from brain-imaging data.
- Analysis of social interactions

1.1 Problem formulation and background

$X = \{X_1, \ldots, X_p\}$ is a set of random variables, $G = (V, E)$ is a Markov network representing the conditional independence structure of the joint distribution $P(X)$.

The set of vertices $V = \{1, \ldots, p\}$ corresponds to the set of variables. The edge set contains (i, j) if and only if X_i is dependent on X_j given all the remaining variables.

Assume a multivariate Gaussian pdf over $X = \{X_1, \ldots, X_p\}$,

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

Zero entries in $C = \Sigma^{-1}$ correspond to no edges in the graph. The goal is to find a sparse maximum likelihood estimate of the inverse-covariance matrix $C = \Sigma^{-1}$.

A common approach is to include as penalty the ℓ_1-norm of C yielding a convex regularization of ℓ_0 norm measuring the sparsity. This yields $p(C) = (\lambda/2)^{p^2} e^{-\lambda \|C\|_1}$ where the objective is to find the maximum log-likelihood solution of

$$\arg\max_{C \succ 0} \log p(C|\mathbf{X})$$

where X is an $n \times p$ matrix. After this change, the problem becomes

$$\arg\max_{C \succ 0} \frac{n}{2}[\log\det(C) - tr(AC)] - \|C\|_S \tag{1}$$

where $A = \dfrac{1}{n}\sum_{i=1} x_i x_i^T$ and $\|C\|_S$ denotes the sum of absolute values of the elements of the matrix SC. Typically $S = \frac{n}{2}\lambda$ times all-one matrix and $\|C\|_S$ reduces to $\frac{n}{2}\lambda\|C\|_1$.

The dual formulation of problem 1

$$\arg\max_{W \succ 0} \frac{n}{2}[\log\det(W) - \frac{np}{2}] \; s.t. - S \leq \frac{n}{2}(W - A) \leq S \tag{2}$$

The optimality conditions for the above problems imply that

$$\begin{array}{c} W = C^{-1} \\ \frac{n}{2}W_{ij} - A_{ij} = S_{ij} \text{ for} C_{ij} > 0 \\ \frac{n}{2}W_{ij} - A_{ij} = -S_{ij} \text{ for} C_{ij} < 0 \end{array} \tag{3}$$

For large(small) λ, we get sparse(dense) C.

Note that the problems above are (SDP) problems which can be solved by IPM's in polynomial time. However;

- Each iteration requires $\mathcal{O}(p^6)$ operations.
- Sparsity of the solution is obtained only in the limit.

Alternative methods COVSEL and *glasso* apply a block coordinate descent method where at each iteration one row(column) is updated.

- COVSEL solves the subproblems via IPM's (second order cone quadratic)
- *glasso* solves the subproblems with an active-set algorithm.

1.2 SINCO and other algorithms

In this section, we compare SINCO and other methods and highlight their differences.

SINCO solves the primal problem directly with coordinate descent method. It updates either one diagonal or two off-diagonal entries in C. The solution to each subproblem is available in closed form as the root of a quadratic equation. In $\mathcal{O}(p^2)$ operations, a potential step is computed for all pairs (i, j). The step which provides the best function value improvement is chosen.

On the other hand, *glasso* and COVSEL require solution of a quadratic programming problem whose complexity always exceeds $\mathcal{O}(p^2)$. Other methods updates a whole row(column) which reduces the number of overall steps.

Another advantage of SINCO is that when initialized from a sparse(identity matrix) solution, it generates one or two nonzero entries at each step. This leads to less false-positive error. On the other hand, it might suffer from introducing too few non-zeros and miss some of the true positives on denser networks.

The most important aspect of SINCO which is relevant for this project is that each potential step in SINCO is computed for each (i, j) independently, and they can be done in **parallel**.

1.3 Algorithm description

At each iteration, the matrix C is updated by changing one element: $C + \theta(e_i e_j^T + e_j e_i^T)$. Exact line search optimizing the objective function along the direction $(e_i e_j^T + e_j e_i^T)$ reduces to a quadratic equation. Find the best function value improvement. After the step is chosen, the dual matrix $W = C^{-1}$ is updated in $\mathcal{O}(p^2)$ operations with Sherman-Morrison-Woodbury formula:

$$(X + ab^T)^{-1} = X^{-1} - X^{-1}a(1 + b^T X^{-1}a)^{-1}b^T X^{-1} \tag{4}$$

Formally the SINCO algorithm applies to the following formulation:

$$\begin{aligned} \max_{C',C''} \quad & \tfrac{n}{2}[\log\det(C' - C'') - tr(A(C' - C''))] - tr(S(C' + C'')) \\ \text{s.t.} \quad & C' \geq 0,\ C'' \geq 0, C' - C'' \succ 0 \end{aligned} \tag{5}$$

Algorithm

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.

2 Parallel SINCO

In second step of SINCO the potential function improvement is calculated for each (i, j) coordinate of the matrix. The function improvement can be calculated independent for each (i, j). SINCO is parallelized using this step. In the parallelized scheme, each process searches the maximum function improvement on part of the matrix. When all of them found their potential maximum coordinate they compare their results and continue with the coordinate which has the maximum potential among their results. After step 2 the algorithm continues as usual.

Using the parallelized scheme a parallel SINCO is developed for both shared memory and distributed memory architectures from the provided serial SINCO. Computational experiments are conducted.

3 Computational Experiments

Computational experiments are conducted for different sizes of the problem. CORAL machine polyps1 is used for the shared memory experiments. For the distributed

memory experiments CORAL machines in Table 1 are used.

Table 1: Machines used in the experiments

Name	Memory	CPU
carp	1000	Pentium IV 2.4GHz
dogfish	1000	Pentium IV 2.4GHz
eel	756	Pentium 4 2.0GHz (x2)
hobbes	756	Pentium 4 2.0GHz
parrotfish	756	Pentium 4 2.0GHz
snapper	756	Pentium 4 2.0GHz
squid	756	Pentium 4 2.0GHz
polyps1	—	—

OpenMP is used for developing the parallel SINCO for distributed memory architectures. OpenMPI is used for distributed memory architectures. Table 2 shows the results of the shared memory experiments.

Table 2: Results of shared memory experiments

p	serial		parallel		Efficiency
	thread	time	threads	time	
200	1	0m42.153s	7	0m31.457s	19%
400	1	4m28.599s	7	4m19.624s	15%
600	1	17m28.451s	7	14m32.374s	17%
800	1	36m26.841s	7	35m17.449s	15%
1200	1	128m12.952s	7	122m19.524s	15%

Experiments in the serial column is conducted by creating only one thread. The experiments on the serial column are conducted by creating 7 threads. The time of the execution of the algorithm is measured. The time measured is the wall clock time. Using the time measurements efficiency of parallelization is calculated using Equation 6.

$$E = \frac{T_{serial}}{threads \times T_{parallel}} \tag{6}$$

The execution time of the algorithm increases as the size of the problem increases as expected. The efficiency of the parallelization is around 15% for the shared memory architecture.

Table 3: Results of distributed memory experiments

p	serial			parallel	Efficiency
	execution time	step 2 time	ratio of step 2	time	
200	0m43.012s	9	21%	1m7.801s	9%
400	6m15.220s	77	21%	6m16.458s	14%
600	20m59.761s	176	14%	19m22.528s	15%
800	32m38.807s	300	15%	39m16.490s	12%
1200	139m52.880s	1184	14%	128m39.680s	16%

Table 4: Distributed memory, Step 2 execution time

p	serial		parallel		Efficiency
	processor	avg. time	processor	avg. time	
200	snapper	0.00224	1-7	0.00028	114%
400	snapper	0.00960	1-7	0.00098	140%
600	snapper	0.01541	1-7	0.00246	89%
800	snapper	0.02203	1-7	0.00403	78%
1200	snapper	0.06160	1-7	0.00863	101%

Table 3 shows the results of the experiments on distributed memory architecture.

The serial SINCO is executed on CORAL machine snapper. The parallel SINCO is executed on the machines given in the Table 1 (except polyps1). The execution times of both versions are given in the table. The efficiency of parallelization is also around 15% for distributed memory. In this table we have 2 extra columns. Step 2 time and its ratio to the all execution time. Step 2 time is the time of execution of the parallelized section of the SINCO. We observe that the ratio of the parallelized section the the all execution time is around 15%. This explains the efficiency of the algorithm. Since the parallelized section of the serial algorithm is only the 15% of the total execution time, there is an implied bound on the efficiency of the algorithm which is 15%. This means that in terms of the step 2 parallelization is very efficient and efficiency is close to linear (100%) efficiency.

Table 4 shows the average execution time of the Step 2 in serial and parallel versions. Efficiency in terms of these execution times are calculated.

Efficiency column in Table 4 is as expected. This indicated that the efficiency of the parallelization in the parallelized section of the algorithm is close to linear efficiency.

4 Conclusion

There are two main conclusions of this study. The first is the fact that the execution time of the search section of the current SINCO implementation is 15% of the total execution time. The second is that the Step 2 of the SINCO is very prone to parallelization and its parallelization efficiency is close to 100%. Using these two conclusions one can estimate the execution time of the current parallel SINCO. It can be estimated using Equation 7.

$$T_{parallel} = \frac{T_{serial}}{0.15 \times cluster\ size} \tag{7}$$

Parallelization of the Step 4 of the algorithm can be thought as a future research direction. In this step the inverse matrix is also updated for each (i, j) entry. Complexity of this step is also $O(p^2)$ like Step 2. Assuming that it can be parallelized as Step 2 and its ratio in the total execution time is also r_4, then the execution time of the new parallel algorithm can be estimated using Equation 8.

$$T_{parallel} = \frac{T_{serial}}{(0.15 + r_4) \times cluster\ size} \tag{8}$$

References

[1] . Scheinberg and I. Rish, 2009, SINCO - a greedy coordinate ascent method for sparse inverse covariance selection problem.

[2] . Scheinberg and S. Ma, 2010, Optimization methods for sparse inverse covariance selection problem.