Notes for running HdBCS

Guiyuan Lei*

27 Feb - 29 March, 2006

HdBCS (Bayesian Covariance Selection in High-Dimensions) is designed by Adrian Dobra (adobra@stat.duke.edu). It is a package to perform covariance selection for datasets with tens or possibly hundreds of thousands of variables. Codes can be downloaded from
http://www.stat.duke.edu/∼adobra/hdbcs.html

1 Run MPICH on single machine

This HdBCS is an MPICH program, there is master() and slave() in the code which means at least two processes are needed. A set of steps need to be done for ssh to work properly with mpich. Ssh should be configured to allow communication between processes/nodes without password. In order to test this mpich program in a single machine, ssh should be able to access localhost so that communication is allowed between (virtual) multiple processes within local node.

1.1 Configue ssh for MPI

1. ssh−keygen −t dsa −b 1024 −f ∼/.ssh/gl_ssh
   Then enter passphrase. Two files will be generated.
2. cd ∼/.ssh
   cat gl_ssh.pub ≫ authorized_keys
3. ssh−add ∼/.ssh/gl_ssh

*Guiyuan.Lei@ncl.ac.uk

4. run MPI program, for example
   mpirun -np 8 gibbsreg.exe

5. Deactivate the ssh agent by using ssh−add −d ∼/.ssh/gl_ssh. After deactivate, if you want to run ssh without password, you should run the command ssh−add ∼/.ssh/gl_ssh again.

1.2 Allow ssh localhost

In order to run MPICH on a single machine, ssh should be configured to allow communication between (virtual) processes within one node without password.
Add line:
sshd:127.0.0.1
in the file /etc/hosts.allow so that ssh is able to access localhost.

2 Compile mpich code: mix c++, fortran, blitz++ and lapack

The code was written in c++, also uses publicly available software packages including blitz++ and lapack, so it is a bit difficult to compile it. In my compg machine, I use the following command for STEP1:
g++ -c gibbsreg.cpp -I/usr/lib/mpich/include
mpif77 -lstdc++ gibbsreg.o -lblas2 -llapack -lg2c -lm

For STEP2, to compile:
g++ -c graphlist.cpp
g++ -c startpoint.cpp -I/usr/lib/mpich/include
mpif77 -o startpoint.exe -lstdc++ graphlist.o startpoint.o -lblas2 -llapack -lg2c -lm

Compiling STEP3 is simaliar to that of STEP2. Note that I have blas2 installed in my machine, so I use -lblas2 instead of -lblas. The header file of mpi and blitz (random number generator) maybe different in different machine, so need to be modified also.

3 One bug in STEP1: Filtering

This first step is for filtering possible predictors for each variable by selecting good regression models from thousands of candidate predictors. The possible predictors are stored in $pv(i)$ for each gene i. The neworks generated in this step are dependent networks.

When I use small number of genes and run the STEP1 (gibbsreg.cpp), I got "message truncation" error. This is because the size $(npred + 2)$ of array $workresults$ maybe larger than number of genes $(NumberOfGenes)$, so the number of int in MPI_Recv($workresults$,$NumberOfGenes$,...) is smaller than the number of int in MPI_Send(&$workresults$,$npred + 2$,...).

The array $workresults$ is used to record the possible predictors of "targetGene", the first and second elements of this array ($workresults[0]$ and $workresults[1]$) are used to store "targetGene" and "npred" (number of predictors), so the size of array $workresults$ is $(npred+2)$. The problem is that the number of possible predictors may be as large as $(NumberOfGenes-1)$ because all other genes can be possible predictors of the $targetGene$. So $(npred + 2)$ may be as large as $(NumberOfGenes + 1)$.

To correct this bug:
(1) The array $workresults$ should be defined (both in slave and master) as $workresults[NumberOfGenes+1]$ instead of $workresults[NumberOfGenes]$.
(2) In master(), replace MPI_Recv($workresults$,$NumberOfGenes$,...) with MPI_Recv($workresults$,$NumberOfGenes + 1$,...). Note that there are two MPI_Recv($workresults$,$NumberOfGenes$,...) should be replaced.

One thing related to this bug. As a rule, for sparse newtwork, the number of possible predictors shall be smaller than $(NumberOfGenes - 1)$. But I noted that the array $workresults$ is not for record the best model, but for union of all models during the search process.

in slave():
$j = 0$;
for($i = 0$;$i < NumberOfGenes$;$i++$)
{ if($varFreq[i] > 0$)
{$workresults[2 + j] = i$;

$j++;$
}
}

The above code means that all predictors appeared in any models during the model search process will be recorded in the array *workresults*. That is why *npred* would be $(NumberOfGenes - 1)$. Is it better to set threshold (like model average) here?

4 STEP2: Generating good starting models

This step is to generate compositional networks, that is Direct acyclic graph (DAG) here, from dependent networks. The idea is to assembly the whole DAG from collections of all regression models by ordering the nodes. The compositional predictors $cpv(i)$ is selected from $pv(i)$ (which is generated in first step).

Parameter "*nGraphsToGenerate*" is the number of starting models you want to generate. These starting models are used in SETP3 for improving mdoel.

If use 2 processes for MPI,
mpirun -np 2 startpoint_g100.exe (_g100 means dealing with 100 genes)
the run is terminated with the following error message:
"p4_error:interrupt SIGFPE:8
killed by signal 2"

If use 8 processes
mpirun -np 8 startpoint_g100.exe
It is ok.

This "number of processes" kind problem is strange, bug or some other issues? I don't know.

5 STEP3: Improving the starting models generated at STEP2

In this step, the starting points/models (which are generated in STEP2) are improved in a procedure (simulated annealing) that converges to local models of the posterior distribution over the space of DAGs.

1. This program should be run separate instance of the program from each starting model. So set the parameter "*startPoint*" as 1, 2, $\cdots$, *nGraphsToGenerate* respectlly and run the program *nGraphsToGenerate* times. In each run, why should use EXACTLY eight processors?

2. Removing un-necessary files by running rmdag.exe. Also should specify which starting model to use (parameter "*startPoint*");

3. Concatenate the ".sample" files associated with each starting point in an overall sample file with information about all the models generated. For example, type:
   cat jcgs.89x100.rma.dat.?.sample > jcgs.89x100.rma.dat.sample

4. Construct the moralized undirected graph associated with each DAG and calculates the average weight of each edge present in these graphs.