

LMMTool User's Manual
Light Mutual Min (LMM) Algorithm for Structural Learning of Bayesian Networks
Release date: 05/16/2013

Copyright © 2013,
Department of Genetic Medicine
Weill Cornell Medical College
New York, NY, USA 10065

References for citation:

- Rami N. Mahdi and Jason Mezey. “*Sub-Local Constraint Based Learning of Bayesian Networks Using A Joint Dependence Criterion*”. Journal of Machine Learning Research, under review.

Authors:

- Rami N. Mahdi (ramimahdi@yahoo.com)
- Jason Mezey (jgm45@cornell.edu)

Description

LMM software tool is a Java implementation of the light mutual min (LMM) algorithm which was proposed as a fast and robust constraint based learning of Bayesian networks (BN) from observational data. The tool provides implementation for both, learning the skeleton of the Bayesian network and the orientation of the edges in the skeleton to recover the completed partially directed acyclic graph (CPDAG). The tool is developed with sufficient flexibility to easily allow users to incorporate their own conditional independence methods. Also, other researchers will be able to use either of the skeleton recovery or the orientation of the edges with a third party method. For example, researchers, who work on skeleton recovery methods, can use the edges orientation part of LMM to orient their skeleton into CPDAG or vice-versa.

Specifications

- LMMTool was developed and tested using Java version 6.
- LMMTool supports both the recovery of the skeleton and the recovery of the completed partially directed acyclic graph (CPDAG) from a given skeleton.
- Current version supports the partial correlation for conditional independence which can be used with multivariate continuous data.

Installation

- Java version 6 or higher needs to be installed to be able to compile and run LMMTool.
- Non-expert java users are advised to download Java development environment such as JDeveloper (<http://www.oracle.com>).
- Dependence on external packages: The following packages has to be available in the classpath:
 - o Commons-math package: (<http://commons.apache.org/math/>).
 - o Jama package: (<http://math.nist.gov/javanumerics/jama/>).

Currently Not-Supported Features

- The current version only supports the partial correlation for conditional independence measurement. However, users can use the guidelines bellow to develop their own independence methods and incorporate it into the tool.
- Current version does not encapsulate any tool for visualization. Users are advised to use external java packages such as the JUNG package available at (<http://jung.sourceforge.net/index.html>).
- Current version does not offer a tool for model selection in terms of the size of the skeleton to be recovered (number of edges). Users are assumed to have a rough estimation of the of the number of edges to be recovered, otherwise, external tool such as the BNLearn can be used to provide scores for multiple recovered networks such as the BIC score. The BNLearn tool is available in R language at (<http://cran.r-project.org/web/packages/bnlearn/index.html>).

LMM for Skelton Recovery

LMM recovers skeleton using two phases, forward selection and backward elimination. Based on our experience, it is advised that the forward selection adds as many edges as possible. For example, if the user desire a skeleton of 1000 edges, he should let the algorithm add 50% or 100% more edges than that in the forward selection phase and relies on the backward elimination to remove them to reach the desired size. In addition, the algorithm is designed to produce multiple skeletons of different sizes from one run

of backward elimination as follows.

```
// Partial correlation independence/dependence tool
GContinuousData dataSource = new GContinuousData("C:/simulation/data.txt");
IndependenceTesterInterface tester = new PartialCorrelationTester_EMP_Basic (dataSource);

/* Using correlation most edges get ignored till of the rest of the algorithm except the top correlated maxCandidateEdges of
pairs of nodes. However, this maximum number of candidate edges should be large and as a heuristic, we set this number to
100 times the number of nodes. */
int maxCandidates = (int)(tester.getObservationsObject().getDimCount()*100);

// maximum number of edges to add in the forward selection Stage before starting backward elimination
int maxEdgesToAdd = (int)(tester.getObservationsObject().getDimCount()*3) ;

// maximum size of any conditioning set
SetX.maxSubsetSize = 4;
double jCPD_Threshold = 0.00001; //  $\gamma$  in LMM paper
double relaxationThreshold = 1E-6; //  $\omega$  parameter in LMM paper

// call LMM for skeleton recovery
LMMAlg alg = new LMMAlg();
ArrayList<UDEdge> rankedEdges= alg.learnSkeleton(tester, jCPD_Threshold, relaxationThreshold, maxEdges, maxCandidates);

// save edge list to file
DataLoader.file_put_contents("/output.edgeList.txt", UDEdge.toStringList(rankedEdges) );
```

LMM for CPDAG Recovery

Once a list of edges are available to represent the skeleton, LMM_EO can be used to orient the edges to convert the skeleton into a CPDAG as follows

```
// user input file or the LMM for skeleton recovery create a list of undirected edges.
ArrayList<UDEdge> edgeList = ???

// Create a Skeleton object of the list of edges
Skeleton skel = new Skeleton( edgeList, data[0].length );

// Orient the given skeleton into a CPDAG
PDAG cpdag = LMMAlg_EO.getCPDAG(tester, skel);

// save partially oriented edges to file
DataLoader.file_put_contents("output.CPDAG.txt", cpdag.toString() );
```

Customized Conditional Dependence Testing

To use external implementation of conditional independence/dependence methods, you will have to create a class that implements the **IndependenceTesterInterface** interface. The provided class **PartialCorrelationTester** is an example of such implementation. Any new class that implements **IndependenceTesterInterface** will have to implement the following two abstract methods

```
/* return the posterior probability of the alternative hypothesis that the conditional dependence between x and y when  
 * conditioning on set z is not zero;  
 * Also, the size of the superset that is being used to compute the one-sided CPPD should be given as a parameter  
 * See Paper for details or the implementation of PartialCorrelationTester for example */  
public abstract double getCD_Prob(int x, int y, SetX z , int superSetSize);  
  
/* return the object that contains the data used to compute conditional dependence  
 * New types of data will have to create a new class that implements the interface BNObservationsInterface.  
 * See the GContineousData class for example */  
BNObservationsInterface getObservationsObject();
```

Easy example

The following example from the class **MainTest** loads one data object, extract a skeleton and convert them into CPDAGs

```
// Partial correlation independence/dependence tool  
GContineousData dataSource = new GContineousData(("C:/simulation/data.txt");  
IndependenceTesterInterface tester = new PartialCorrelationTester_EMP_Basic (dataSource);  
  
// Edges with the least correlation get ignored till of the rest of the algorithm except the top correlated maxCandidateEdges  
int maxCandidateEdges = (int)(tester.getObservationsObject().getDimCount()*99);  
  
// maximum number of edges to add in the forward selection Stage Before Starting backward elimination  
// As a heuristic it is set to 2.5 times the number of nodes  
int maxEdgesToAdd = (int)(tester.getObservationsObject().getDimCount()*2.5) ;  
  
// maximum size of any conditioning set  
SetX.maxSubsetSize = 4;  
double jCPPD_Threshold = 0.00001; //  $\gamma$  in LMM paper  
double relaxationThreshold = 1E-6; //  $\omega$  parameter in LMM paper  
  
// call LMM for skeleton recovery  
LMMAlg alg = new LMMAlg();  
ArrayList<UEdge> rankedEdges= alg.learnSkeleton(tester, jCPPD_Threshold, relaxationThreshold, maxEdges, maxCandidates);  
DataLoader.file_put_contents("out.edgeList.txt", UEdge.toStringList(rankedEdges) );  
  
// user input file or the LMM for skeleton recovery create a list of undirected edges.  
ArrayList<UEdge> subEdgeList = new ArrayList<UEdge> (rankedEdges.subList(0, 100));  
// Create a Skeleton object of the list of edges  
Skeleton skel = new Skeleton( subEdgeList, data[0].length );  
// Orient the given skeleton into a CPDAG  
PDAG cpdag = LMMAlg_EO.getCPDAG(tester, skel);  
// save partially oriented edges to file  
DataLoader.file_put_contents("output.CPDAG.txt", cpdag.toString() );
```