A NEW CONSTRAINT-BASED ALGORITHM TO LEARN BAYESIAN NETWORK STRUCTURE FROM DATA: CONTROL OF PAIR-WISE SPURIOUS INFORMATION (CSPI)

A THESIS IN

Computer Science

Presented to the Faculty of the University

of Missouri-Kansas City in partial fulfillment of

the requirements for the degree

MASTERS OF SCIENCE

by

PABLO DE MORAIS ANDRADE

B. S., State University of Campinas, 2005

Kansas City, Missouri

2011

A NEW CONTRAINT-BASED ALGORITHM TO LEARN BAYESIAN NETWORK STRUCTURE FROM DATA: CONTROL OF PAIR-WISE SPURIOUS INFORMATION

Pablo de Morais Andrade, Candidate for the Master of Science Degree

University of Missouri-Kansas City, 2011

ABSTRACT

A Bayesian network is a directed acyclic graphical representation of a set of variables. This representation occupies the middle ground between a causal network and a simple list of pairwise correlations by including information about dependencies between variables.

There are applications of Bayesian networks in many fields, such as financial risk management, bioinformatics and audio-visual perception, to name just a few. However, learning the network structure from data requires an exponential number of conditional independence tests; several algorithms have been proposed in order to reduce the runtime of this procedure.

We present a new constraint-based algorithm for learning Bayesian network structure from data, based on Control of Spurious Pairwise Information (CSPI). We limit the computational cost of learning by trading an increase in complexity of the initial steps for a substantial reduction in the complexity of conditional pairwise independence testing. We employ a logging and rollback strategy to reduce the number of missing edges.

We show that the CSPI algorithm outperforms several other algorithms in complexity and/or accuracy on benchmark datasets.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the School of Computing and Engineering, have examined a thesis titled "A New Constraint-Based Algorithm to Learn Bayesian Network Structure From Data: Control of Spurious Pairwise Information (CSPI)", presented by Pablo de Morais Andrade, candidate for the Masters of Science degree, and certify that in their opinion it is worthy of acceptance.

<u>Supervisory Committee</u>

Deendayal Dinakarpandian, Ph.D, Committee Chair

Department of Computer Science Electrical Engineering

Praveen R. Rao, Ph.D

Department of Computer Science Electrical Engineering

Yugyung Lee, Ph.D

Department of Computer Science Electrical Engineering

TABLE OF CONTENTS

LIST OF EQUATIONS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

A Bayesian network (BN) is a graphical model that represents the set of conditional probabilities associated with a group of variables (Cooper and Herskovits 1992; Heckerman 1995; Heckerman, Geiger and Chickering 1995). A BN is a Directed Acyclic Graph (DAG) where the vertices represent variables and the edges are directed according to the conditional probabilities associated with these variables as follows: if a vertex v_X represents a variable X, and the probabilities of the values assumed by the variable X are conditioned on the values assumed by the variables Y and Z represented, respectively, by the vertices v_Y and v_Z, there should be edges directed from the vertices v_Y and v_Z towards the vertex v_X . A simple example of unconditional and conditional probabilities with X, Y and Z binary variables is given in Tables 1 and 2. The corresponding BN is shown in Figure 1.

As a classifier, BN has a significant advantage over the naive Bayesian Classifier, since the latter does not take into consideration variables that are dependent or conditionally independent of each variable (Friedman, Geiger and Goldszmidt 1997). The task of learning the conditional probability table is simple when the structure of the network is known. For discrete variables, it is just a matter of calculating the frequency of each value assumed by a variable given the values assumed by its adjacent ancestors in the graph (Grossman and Domingos 2004; Needham, Bradford, Bulpitt and Westhead 2006). The problem of learning the structure, though, is NP-complete (Chickering 1996; Chickering, Heckerman and Meek 2004).

Several structure learning algorithms have been proposed. A tree based structure was proposed in Chow and Liu (1968). Algorithms to recover a DAG can be divided into two main approaches. In the first approach, known as Search and Score, the algorithm performs a search (greedy search, mostly hill-climbing) for the network that best suits the data based on a given score. Examples of scoring functions are: Minimum Description Length (Lam and Bacchus 1994), Conditional Mutual Information (Friedman, Geiger and Goldszmidt 1997), Bayesian Information Criterion (Elidan, Nachman and Friedman 2007) and BDe Metric (Heckerman, Geiger and Chickering 1995). In the second approach, constraint-based (CB) algorithms (Cheng et al. 2002; Yehezkel and Lerner 2009) use conditional independence tests in order to reduce the number of edges in the graph, consequently reducing the complexity of the problem, and orient the edges based on the results of the CI tests performed (see The IC algorithm in Pearl 2000). A hybrid approach (use of scoring function and CI tests) is found in Tsamardinos, Brown and Aliferis (2006). Bayesian Networks have been applied in many domains: risk management (Cornalba and Giudici 2004), audio-visual perception (Besson et al. 2010) and particularly Bioinformatics (Friedman, Linial, Nachman and Peer 2000; Needham, Bradford and Bulpitt 2006). A variety of tools to build networks are freely available (Scutari 2010; Wilczynski and Dojer 2009); Buntine (1996) offers a guide to literature and software related to Bayesian Networks.

The CSPI algorithm proposed in this thesis tries to overcome some of the problems found in CB algorithms. CB algorithms usually start (assuming there is no previous expert knowledge) with a complete or empty network. Consequently, CI tests usually have large condition sets, which commonly give unreliable results (Cheng et al. 2002; Yehezkel and Lerner 2009). Alternatively, CB algorithms start with the minimum spanning tree, which reduces the initial size of condition sets, but relies on a fallacy for the removal of the edges: the assumption that all edges representing stronger relations of dependences are true edges in the network. Figure 2 shows a typical example when this assumption fails. The variables X and W represented, respectively, by vertices v_X and v_Y are conditionally independent but the values assumed by both variables are dependent on the values assumed by variables Y and W (represented by the vertices v_Y and v_W). Although conditionally independent, in many cases, the correlation or information (Shannon 1948) between the variables X and W is higher than one or more of the ones found between X and Z, or Z and W. In order to overcome this problem, the CSPI algorithm starts with a procedure costlier in n (number of vertices) than the minimum spanning tree algorithm but makes use of a deletion log in order to facilitate the rollback. This reduces the number of conditional independence (CI) tests necessary. Figure 3 shows the initial procedure of CSPI while Table 3 shows the corresponding deletion log. Using this procedure, CSPI also starts with CI tests having small condition sets, as in the minimum spanning tree, except that for each edge removed on account of being considered spurious information, the vertices responsible for this removal are stored, allowing the algorithm to limit the number of CI tests. The purpose of the CSPI algorithm is to reduce the complexity with respect to the number of CI tests (and size of condition sets), which is the major factor responsible for the increase in run-time in CB algorithms (Cheng et al. 2002; Yehezkel and Lerner 2009), by trading an increase in the complexity with respect to the number of edges during the initial step of the algorithm. Other steps of the algorithm include corrections by shrinking the network and re-testing the edges removed from the network.

The independence and conditional independence tests were performed using a normalized version of the measures mutual information and conditional mutual information - normalized with respect to the entropy of the variables being tested. In empirical tests using benchmark datasets, these normalized versions had better results than the regular versions. Many normalized conditional mutual information versions have been proposed (Oude and Pavlin 2009; Strehl and Ghosh 2002; Richiardi 2007; Besson al. 2010), a list of the proposed measures can be found in Yao (2003).

Table 1 - Unconditional probability table of variables Y and Z

Y	p(Y)	Z	P(Z)
0	0.4	0	0.3
1	0.6	1	0.7

Table 2 - Conditional probability table of variable X given variables Y and Z

X	p(X|Y=0,Z=0)	p(X|Y=0,Z=1)	p(X|Y=1,Z=0)	p(X|Y=1,Z=1)
0	0.1	0.8	0.7	0.4
1	0.9	0.2	0.3	0.6

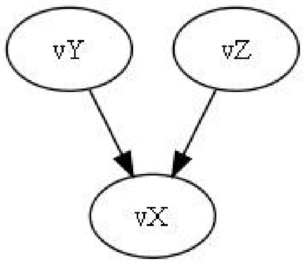

Figure 1 - Bayesian Network representing dependences between variables X, Y and Z corresponding to the probabilities in Tables 1 and 2

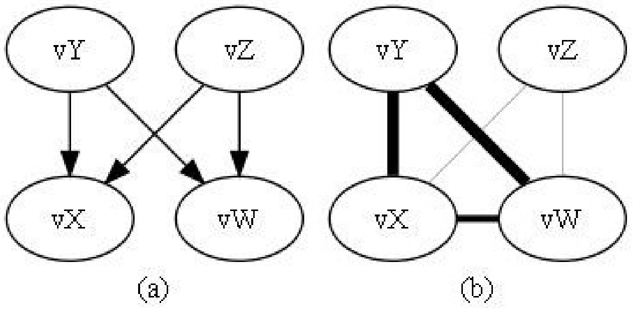

Figure 2 - BN representing high but spurious mutual information: (a) True Network, (b) Edge width representing measure of information (vX-vW higher than vZ-vX and vZ-vW)

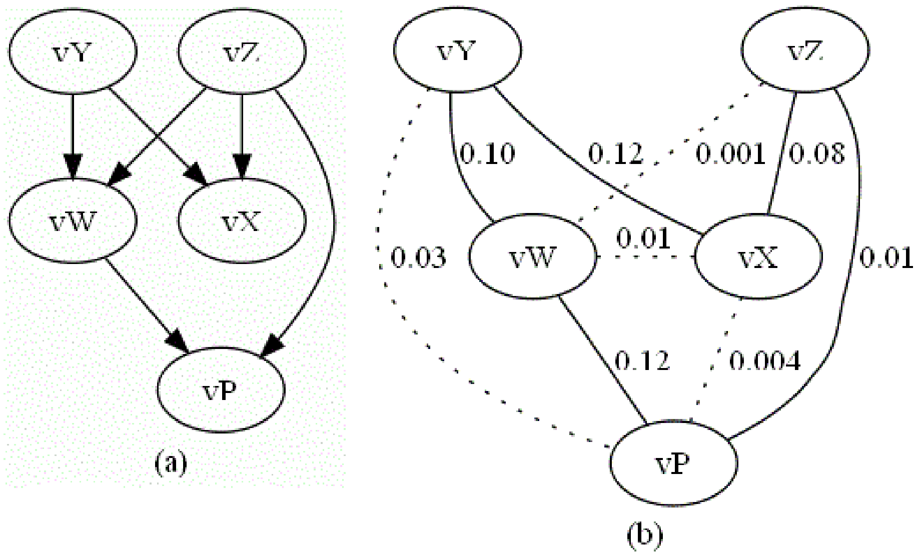

Figure 3 - BN representing temporarily removed edges: (a) True Network, (b) Dotted edges represent the temporarily removed edges, note that v_Z-v_W is removed by {v_Z-v_X; v_X-v_W}, but v_W-v_X is also removed by {v_W-v_Y; v_Y-v_X}; and v_X-v_P is removed by {v_X-v_W;v_W-v_P} a and {v_X-v_Y;v_Y-v_P}

Table 3 – Deletion log maintains a record of the sets of the vertices whose associated edges were responsible for every edge deleted from a fully connected network. This facilitates rollback from incorrect deletions (e.g., edge v_Z-v_W) in subsequent steps of the algorithm

	$\mathbf{v_Y}$	$\mathbf{v_Z}$	$\mathbf{v_W}$	$\mathbf{v_X}$	$\mathbf{v_P}$
$\mathbf{v_Y}$	---	---	---	---	v_W
$\mathbf{v_Z}$	---	---	v_X & v_P	---	---
$\mathbf{v_W}$	---	v_X & v_P	---	v_Y	---
$\mathbf{v_X}$	---	---	v_Y	---	v_Y & v_W & v_Z
$\mathbf{v_P}$	v_W	---	---	v_Y & v_W & v_Z	---

CHAPTER 2

BACKGROUND

2.1 Bayesian Network

For a given set of discrete random variables $X = \{X_1, \ldots, X_n\}$, a BN graphically represents the conditional probabilities associated with these variables. The graph is a DAG with parent variables pa_i of X_i, such that the variable X_i is independent of its non-descendants given its parents (Friedman, Linial, Nachman and Peer 2000). Therefore, the probability of a network structure is given by (Cooper and Herskovits 1992; Friedman, Geiger and Goldszmidt 1997; Yehezkel and Lerner 2009; Heckerman 1995):

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | pa_i)$$

Equation 1 - Probability of a Bayesian Network Structure

In order to find the set of parents of each variable X_i, we search for the set S_{ij} for each pair of variables X_i and X_j such that, conditioned on this set S_{ij}, the variables X_i and X_j are independent (Tsamardinos, Brown & Aliferis, 2006):

$$P(X_i, X_j | S_{ij}) = P(X_i | S_{ij}) P(X_j | S_{ij})$$

Equation 2 – Conditioned on the set S_{ij}, the variables X_i and X_j are independent

We use as measures of independence a normalized version of the measure unconditional mutual information, and a threshold ε_u to define independence (results lower than this threshold ε_u are considered unconditional independence). Equation 3 shows the measure of mutual information while Equation 7 shows the normalized version used.

$$I(X_i,X_j) = \sum_{ij} P(X_i = x_i, X_j = x_j)\log\left(\frac{P(X_i = x_i, X_j = x_j)}{P(X_i = x_i)P(X_j = x_j)}\right)$$

Equation 3 – Mutual Information

Equation 3 can also be written as:

$$I(X_i,X_j) = H(X_i) + H(X_j) - H(X_i,X_j)$$

Equation 4 – Mutual Information rewritten

Where H(*X*) is the entropy of variable *X* and is defined as (Shannon, 1948):

$$H(X_i) = -\sum_{i} P(X_i = x_i)\log(P(X_i = x_i))$$

Equation 5 – Entropy of variable X_i

Joint entropy is defined as:

$$H(X_i,X_j) = -\sum_{i,j} P(X_i = x_i, X_j = x_j)\log\left(P(X_i = x_i, X_j = x_j)\right)$$

Equation 6 – Joint entropy of variables X_i and X_j

We finally define normalized mutual information, based on the fact that the mutual information between two variables is always lower than the minimum entropy of these two variables (Yao 2003):

$$I^*(X_i,X_j) = \frac{I(X_i,X_i)}{\min\{H(X_i,X_j)\}}$$

Equation 7 – Normalized version of the measure mutual information

For the conditional mutual information tests, we define a second threshold ε_c (CI tests resulting below this threshold ε_c are considered to be indicative of conditional independence). Conditional mutual information (Cheng et al. 2002) is given by:

$$I(X_i,X_j|Z_k) = \sum_{i,j,k} P(X_i = x_i,X_j = x_j|Z_k = z_k)\log\left(\frac{P\,(X_i = x_i,X_j = x_j|Z_k = z_k}{P(X_i = x_i|Z_k)P(X_j = x_j|Z_k}\right)$$

Equation 8 – Conditional mutual information

The equation for conditional mutual information can be rewritten as:

$$I(X_i,X_j|Z_k) = H(X_i,Z_k) + H(X_j,Z_k) - H(X_i,X_j,Z_k) - H(Z_k)$$

Equation 9 – Conditional mutual information rewritten

A normalized version:

$$I(X_i,X_j|Z_k) = \frac{H(X_i,X_j) + H(X_j,Z_k) - H(X_i,X_j,Z_k) - H(Z_k)}{\min\{H(X_i,Z_k),H(X_j,Z_k)\}}$$

Equation 10 – Normalized version of conditional mutual information

Empirical tests using benchmark datasets showed that the normalized version of mutual information Equation 7 had less false negatives ($I(X_i,X_j) \leq \varepsilon_u$) when X_i and X_j are actually dependent) and false positives ($I(X_i,X_j) > \varepsilon_u$, when X_i and X_j are actually independent) than the regular version of mutual information in Equation 4. The results can be explained based on the fact that variables with low entropy will automatically have lower values of mutual information; setting a constant threshold ε_u, is necessary to correct the significance of the results. The same applies to conditional mutual information in Equation 9. The significance of the results will be influenced by the entropy of variables X_i, X_j, Z_k, particularly when the number of variables in the set Z becomes too large. We consider the normalization step necessary especially when using a greedy search for the set *Z*. Without normalization, the addition of several variables to the condition set might yield low scores for the CI tests that are suggestive of independence (lower than a threshold ε_c), even when that is not true.

2.2 Benchmark Networks

We compared the CSPI algorithm with the recently proposed algorithms Max-Min Hill-Climbing (MMHC), Recursive Autonomy Identification (RAI) and Three-Phase Dependence Analysis (TPDA) using the benchmark networks: Alarm, Insurance, Barley, Child and HailFinder.

A summary of these variables (number of nodes and number of edges) is given in Table 4. Figure 4 shows the Alarm Network and Figure 5, the Insurance Network. The task given is to recover these networks from a dataset.

Table 4 - Number of vertices and edges of Benchmark networks

Network	**Number of Vertices**	**Number of Edges**
Alarm	37	46
Insurance	27	52
Barley	48	84
Child	20	25
HailFinder	56	66

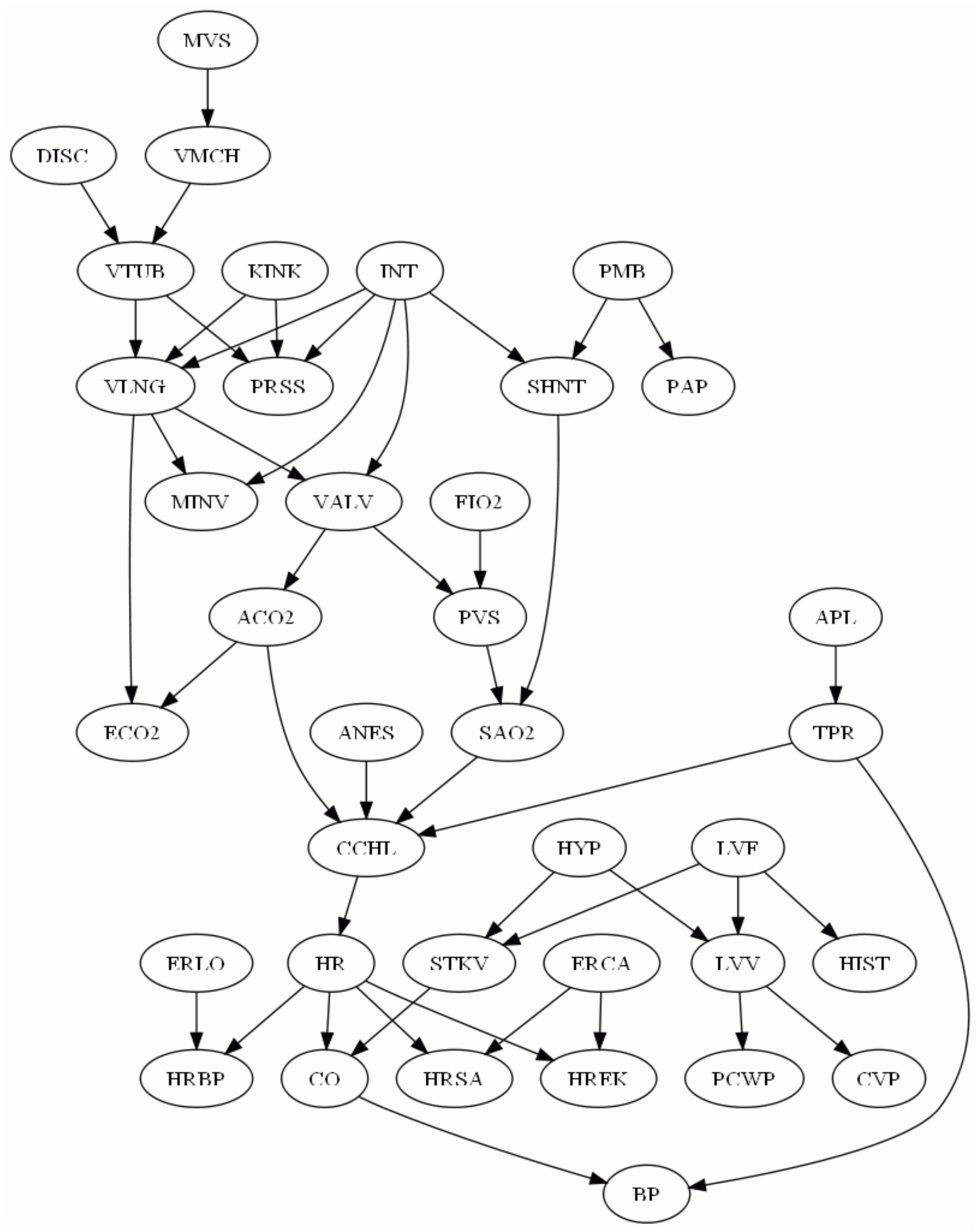

Figure 4 - Alarm Network

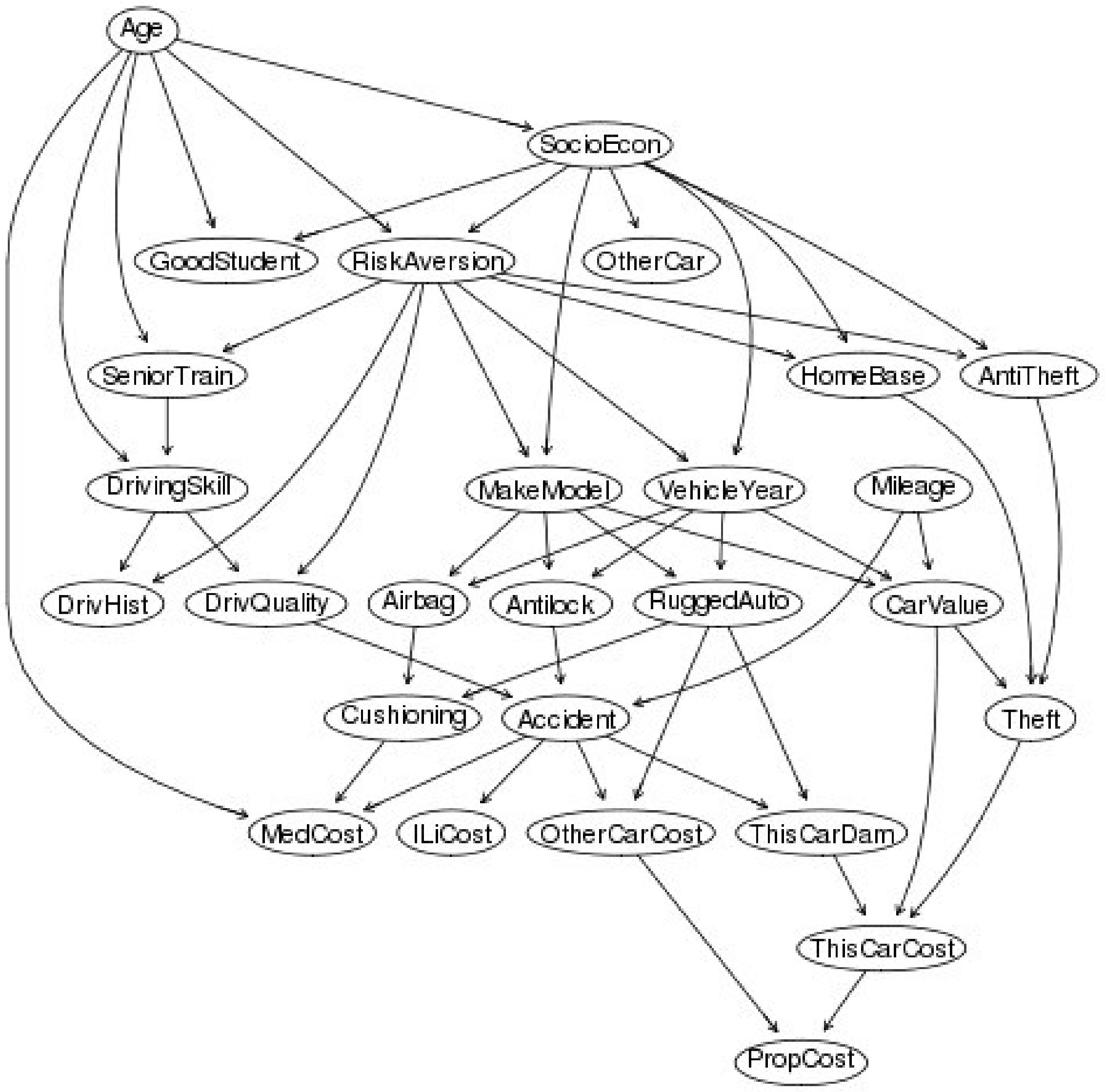

Figure 5 - The Insurance Network

2.3 Measure of Independence

As a measure of independence, we have used a normalized version of the measure *Mutual Information* and as measure of conditional independence a normalized version of *Conditional Mutual Information*, given respectively in Equations 3 and 10.

As we address in the section Future Work, we expect to explore other measures of independence and conditional independence, but for the first version of the software available, only the measures cited above are available.

2.4 Accuracy Measure

For a measure of accuracy, we have used structural Hamming distance, which is the sum of the following structural errors:

- Missing edge (ME)
- Edge missing direction (MD)
- Inverted edge (IE)
- Extra edge (EE)

Therefore, the structural Hamming distance is given by:

$$\mathrm{SHD} = \mathrm{ME} + \mathrm{MD} + \mathrm{IE} + \mathrm{EE}$$

Equation 11 – Structural Hamming distance

CHAPTER 3

THE CSPI ALGORITHM

3.1 Definitions

Mutual information matrix: *n*-by-*n* matrix with values of pairwise unconditional mutual information (n number of variables).

Clean mutual information matrix: *n*-by-*n* matrix with values of pairwise unconditional mutual information with the edges considered spurious set to 0.

Undefined edge: All the edges in the network are initially marked as undefined. An edge that is temporarily removed is also considered undefined (but it is assigned a value of 0 in the clean mutual information matrix).

Undirected edge: An undefined edge that cannot be removed will become an *undirected* edge.

True Negative (TN) and False Negative (FN): Edges considered spurious are temporarily removed from the graph. After a series of conditional independence tests, those not permanently removed will be considered False Negatives (not spurious edges). The edges permanently removed will be considered True Negatives (spurious edges).

Deletion log: Edges temporarily removed for appearing redundant are stored together with a list of vertices that are responsible for the deletion (i.e., vertex opposite the deleted edge when comparing each group of three edges).

setS: For each edge, there is a set of vertices *S* such that, when conditioned on this set, the vertices connected by the edge have the minimum value computed by the CI test. If the vertices are conditionally independent, the CI test value for these vertices conditioned on the set *S* should have a value lower than the threshold ε_c. Each vertex is classified as one of the following: included in this setS (value 1), not included (value 0) or considered a potential addition to the set (value 2, or *ToBeAdded*).

edgesetS: A bit map where a bit is set to one if an edge has a set *S* assigned to it (the matrix is initialized with zeros).

3.2 Description

The algorithm starts by calculating values for pairwise unconditional mutual information, with the result stored in a matrix called the mutual information matrix (*mutualinfo* in Algorithm 1). For each pair of vertices with result greater than a threshold ε_u, an undefined edge is assigned connecting the pair.

The function *RemoveSpuriousMI* (Algorithm 2) is called next. It checks each triple of vertices, marks the edge with the lowest value of mutual information for removal, storing the vertices opposite this edge in a deletion log. The edges removed have their records in the matrix *mutualInfoClean* set to zero (Algorithm 2 line 9). The network itself, though, is not updated since this is not a permanent removal – the edge is still considered undefined.

After removing what is considered spurious mutual information, the vertices are sorted according to the number of undefined edges connected to them and the function *PermanentlyRemoveSpurInfo* is called. This function performs CI tests for each vertex, trying to remove all the edges connected to this vertex that are considered spurious information, by using as condition set the remaining vertices connected to this vertex (its neighbors). The edges not permanently removed are considered false negatives and added back to the network one at a time while the edges permanently removed (true negatives) are no longer undefined but will have their records in the network matrix set to NOEDGE. For each vertex, the number of false negatives are calculated and stored in an array called edges2add (see Algorithm 1, line 15).

The function *ReconsiderFalseNegatives* (Algorithm 3) is then called and follows the order given by the sorted array of the number of edges to be added (array edges2add). This function checks which edges were considered spurious information, but not permanently removed (edge is still undefined and mutualinfoClean matrix is 0), and calls the function *ReconsiderEdgesFN* for each of these edges.

ReconsiderEdgesFN starts with a loop searching for the vertices responsible for the removal of the edge considered false negative (stored in the deletion log). For each of these vertices, it checks if any of the edges previously compared to this edge (in Algorithm 2) was also temporarily removed, in which case there is a recursive call for this edge (lines 6 and 11 of Algorithm 3). The algorithm then tries to remove these two other edges, first checking if there exists a set *S* already assigned to these edges, in which case the function *RemoveEdgeUsingSetS* is called (Algorithm 5). If there is no set *S* assigned to this edge, a similar function is called (*RemoveEdgeUsingNeigh*), which uses the neighbors of both vertices as condition set in a heuristic hill-climbing search (adding the neighbors one at a time, followed by a removal of the neighbors again, one at a time, searching for the lowest of value of CI test). At last, still within this function, the algorithm tries to remove the edge itself, considered false negative. If this edge cannot be removed, it is added back to the network and each of the two vertices connected by this edge is considered for addition to every set *S* of the edge's neighbors (see lines 30-41 of Algorithm 3).

The function *RemoveEdgeUsingSetS* (Algorithm 5) tries to remove a given edge using as condition set the set *S* assigned to this edge (the condition set that has the lowest result for a CI test so far) and a set of vertices that are considered as potential additions to this set. The algorithm then follows a hill-climbing search for the set S such that, conditioned on this set, the vertices connected by the edge being removed have the minimum value for a CI test.

The variable *currentCMI* keeps the value of the current conditional mutual information while *lowestCMI* is the value of the lowest one found so far. The composition of the set *S* is progressively updated by addition and removal of vertices, always searching for the lowest value of conditional mutual information. Finally, if the lowest value found is zero (when the result is lower than a threshold ε_C, i.e., the function *CITest* returns zero), the edge should be removed, and the algorithm removes the vertices connected by this edge from any set *S* where these vertices could have been added by mistake. This procedure is important when all the edges permanently removed are re-tested given their set S (Algorithm 1, line 26).

After the return of the function *ReconsiderFalseNegatives*, all the undefined edges are set to *undirected* and the functions *DirectVStructures* and *DirectRemainingEdges* are called, following *The IC Algorithm* given in Pearl (2000). The assignment of edge direction can fail when the network has an extra edge (edge that should have been removed), or missing edges are wrongly removed – not uncommon when given a large condition set as set *S*. Therefore the algorithm first deletes ”fake” parents by calling the function *ShrinkNetwork*, then re-tests all the edges removed given its set S by calling the function *RetestRemovedEdges*.

In *ShrinkNetwork*, the algorithm removes redundant edges. It performs a CI test between every vertex and each of its parents given the remaining parents. If the result is zero, the edge is removed, and its corresponding set S updated accordantly.

In *RetestRemovedEdges*, a CI test is performed between the vertices connected by each edge removed (network matrix has a value of NOEDGE), given the current set *S* assigned to this edge. If the CI test does not result in zero, the edge is added to the network as *undirected*.

The algorithm ends then with the calls to the functions *DirectVStructures* and *DirectRemainingEdges* in order to direct any undirected edge.

Table 5 - Algorithm describes the main function

Algorithm 1 Main Function

```
1: procedure CSPIBNLEARN(dataset, nvariables)
2:      mutualInfo «- {0.0}
3:      network «- {0}
4:      nundefedges «- {0}
5:      [mutualInfo; network] «- FindUnconditionalMI(dataset)
6:      [mutualInfoClean; removedby]  «- RemoveSpuriousMI(mutualInfo)
7:      sortednundefedges «- Sort(nundefedges)
8:      [network] «- PermanentlyRemoveSpurInfo(network, mutualInfo, mutualInfoClean)
9:      edge2add «- {0}
10:     for v «- 1, nvariables do
11:        edges2add[v] «- 0
12:        for i «- v+1; nvariables do
13:           if network[v][i] == undefined then
14:              if mutualInfoClean[v][i] == 0 then // edge is a false negative
```

Table 5 (continued)

```
15:             edges2add[v]++
16:           end if
17:         end if
18:     end for
19: end for
20: edges2add  «- Sort(edges2add)
21: setS «-{0}
22: [network; setS]  «- ReconsiderFalseNegatives(network, removedby, mutualInfoClean)
23: for v1 «- 1, nvariables do
24:     for v2 «- v+1, nvariables do
25:         if network[v1][v2] == undefined then
26:             network[v1][v2] undirected
27:         end if
28:     end for
29: end for
30: [network] «- DirectVStructures(network, setS)
31: [network] «- DirectRemainingEdges(network, setS)
32: [network] «- ShrinkNetwork(network)
33: [network] «- RetestRemovedEdges(network, setS)
34: [network] «- DirectVStructures(network, setS)
35: [network] «- DirectRemainingEdges(network, setS)
36: end procedure
```

Table 6 - Algorithm describes function to remove the spurious mutual information

Algorithm 2 Remove Spurious Mutual Information

```
1: procedure REMOVESPURIOUSMI(mutualInfo)
2:    markforremoval «- {0}
3:    mutualInfoClean «- mutualInfo
4:    for v1 «- 1,nvariables do
5:       for v2 «- v1+1,nvariables do
6:           for v3 «- 1,nvariables do
7:                [vx,vy] «- min{mutualinfo[v1][v2], mutualinfo[v1][v3], mutualinfo[v2][v3]}
8:                markforremoval[vx][vy] «- 1
9:                mutualInfoClean[vx][vy] «- 0
10:               vz  «- {v1,v2,v3} – {vx, vy}
11:               removedby[vx][vy][vz] «- 1
12:          end for
13:      end for
14:   end for
15:   return [mutualinfoClean, removedby]
16: end procedure
```

Table 7- Algorithm describes function to reconsider all the false negative edges

Algorithm 3 Reconsider False Negatives

```
1:  procedure RECONSIDERFALSENEGATIVES(network, removedby, mutualInfoClean)
2:      for v1 «- 1,nvariables do
3:          for v2 «- v1+1, nvariables do
4:              if mutualInfoClean[v1][v2] == 0 then
5:                  if network[v1][v2] == undefined then
6:                      ReconsiderEdgeFN(network, removedby, mutualInfoClean, v1, v2)
7:                  end if
8:              end if
9:          end for
10:     end for
11:     return [mutualinfoClean, removedby]
12: end procedure
```

Table 8 - Algorithm describes function to reconsider a false negative edge

Algorithm 4 Reconsider Edge False Negative

```
1:  procedure RECONSIDEREDGEFN(network; removedby;mutualIn f oClean;v1;v2)
2:    for v 1,nvariables do
3:      if removedby[v1][v2][v] then // vertex v is responsible for removal of v1-v2
4:        if mutualInfoClean[v1][v] == 0 then
5:          if network[v1][v] == undefined then // edge v1-v is also a FN
6:              ReconsiderEdgeFN(network; removedby;mutualInfoClean,v1,v)
7:          end if
8:        end if
9:        if mutualInfoClean[v][v2] == 0 then
10:         if network[v][v2] == undefined then // edge v-v2 is also a FN
11:           ReconsiderEdgeFN(network, removedby, mutualInfoClean, v, v2)
12:         end if
```

Table 8 (continued)

```
13:       end if
14:       if edgesetS[v1][v] then // There's a set S assigned to edge v1-v
15:          setS[v1][v] «- setS[v1][v]+v2 // Add v2 to the set S of v1-v
16:   remv1v «- RemoveEdgeUsingSetS(network, removedby, mutualInfoClean,v1,v, setS )
17:       else
18:          remv1v «- RemoveEdgeUsingNeigh(network; removedby,mutualInfoClean,v1,v)
19:          edgesetS[v1][v] «- 1 // A set S is assigned to edge v1-v
20:       end if
21:       if edgesetS[v][v2] then // There's a set S assigned to edge v-v2
22:          setS[v][v2] «- setS[v][v2]+v1 // Add v1 to the set S of v-v2
23:     remvv2 «- RemoveEdgeUsingSetS(network; removedby;mutualInfoClean,v,v2, setS)
24:       else
25:          remvv2 «- RemoveEdgeUsingNeigh(network; removedby;mutualInfoClean;v;v2)
26:          edgesetS[v][v2] «- 1 // A set S is assigned to edge v-v2
27:       end if
28:       remv1v2 «- RemoveEdgeUsingNeigh(network; removedby;mutualInfoClean;v1;v2)
29:    end if
30:    if !remv1v2 then // If edge v1-v2 cannot be removed, ...
31:       for i «- 1,nvariables do
32:           if network[v1][i]! = NOEDGE && edgeset[v1][i] then
33:                    / / ... and one of v1's neighbours has a set S assigned to it ...
34:              setS[v1][i][v2] = TOADD // ... v2 should be considered to be added to this set.
35:           end if
36:           if network[v2][i]! = NOEDGE && edgeset[v2][i] then
37:                    //. .. and one of v2's neighbours has a set S assigned to it ...
38:              setS[v2][i][v1] = TOADD // .. v1 should be considered to be added to this set
39:           end if
40:       end for
41:    end if
42: end for
```

Table 9 - Algorithm describes function to remove an edge using the set S

Algorithm 5 Remove edge using set S

```
1:  procedure REMOVEEDGEUSINGSETS(network, removedby, mutualInfoClean, v1, v2, setS)
2:    v1v2setS «- {0}
3:    v1v2addTosetS «- {0}
4:    for v «- 1,nvariables do
5:       if setS[v1][v2][v] == 1 then
6:          v1v2setS «- 1
7:       else if setS[v1][v2][v]==TOADD then
8:          v1v2addTosetS «- 1
9:       end if
10:   end for
11:   currentCMI «- 0
12:   setSchanged «- 0
13:   lowestCMI «- CITest(v1,v2,v1v2setS)
14:   for all v in v1v2addToset do
15:       Add v to v1v2setS
16:       currentCMI «- CITest(v1;v2;v1v2setS)
17:       if currentCMI > lowestCMI then
18:          Remove v from v1v2setS
19:       else
20:          lowestCMI  «- currentCMI
21:          setSchanged «- 1
22:       end if
23:   end for
24:   if setSchanged then
25:     for all v in v1v2setS do
26:        Remove v from v1v2setS
```

Table 9 (continued)

```
27:        currentCMI «- CITest(v1;v2;v1v2setS)
28:        if currentCMI > lowestCMI then
29:          Add v to v1v2setS
30:        else
31:          lowestCMI  «- currentCMI
32:        end if
33:    end for
34:  end if
35:  if lowestCMI == 0 then
36:    network[v1][v2] = NOEDGE
37:       // Correct set S for possible mistakes, when edge v1-v2 is removed
38:    for v «- 1,nvariables do
39:      if setS[v1][v][v2] then
40:        if setS[v1][v] == NOEDGE && setS[v2][v] == NOEDGE then
41:            setS[v1][v][v2] «- 0
42:        end if
43:      end if
44:      if setS[v2][v][v1] then
45:         if setS[v1][v] == NOEDGE && setS[v2][v] == NOEDGE then
46:             setS[v2][v][v1] «- 0
47:         end if
48:      end if
49:    end for
50:  end if
51:  return (lowestCMI == 0)
52: end procedure
```

3.3 Time complexity of algorithm

Bayesian Network structure learning algorithms typically have their performance measured by the number of conditional independence tests executed. The CSPI algorithm starts with a procedure comparing each three vertices of the graph ($O(n^3)$). If all three are connected, the edge representing the weakest dependence (lowest value of mutual information or correlation) is temporarily removed from the graph, assuming that this edge represents spurious information. For each edge removed during this procedure, the third vertex (opposite this edge) is stored in a deletion log as responsible for the removal of this edge. Using this procedure, CSPI starts with CI tests having small condition sets, allowing the algorithm to limit the number of CI tests by using the following rationale: for each three vertices connected by three edges, at most one of the edges represents spurious information. This way, when adding back an edge previously considered spurious information, the algorithm applies CI tests, trying to remove the edges responsible for its removal. Since the number of comparisons (and removals) has complexity $O(n^3)$, the complexity in N (number of CI tests) would be $O(N^3)$, but in fact the algorithm adds and removes the vertices from the condition set S one at a time, searching for the set S that gives the lowest result for a CI test. The final complexity is then of the order $O(N^4)$. Therefore in the worst case, the CSPI algorithm performs $O(N^4)$ CI tests. Tests performed on benchmark dataset for this paper show that the heuristic search used keeps the number of CI tests considerably lower than that. CSPI avoids this ways an exponential number of CI tests, which would be necessary to test all the possible sets for each pair of variables (Chickering 1996; Chickering, Heckerman and Meek 2004). And CSPI is guaranteed to work if the CI results are always correct (See section 3.4 for details).

3.4 Proof of correctness

If two variables A and B are conditionally independent, there is a set S_{AB} such that, conditioned on S_{AB} the CI test I(A;B| S_{AB}) is equal to zero (or lower than a threshold ε). Assuming the independence tests will be correct, we have to prove that the test I(A;B| S_{AB}) will be performed by the algorithm – removing the edge A-B.

If a variable X belongs to S_{AB}, X is a parent of A or B (or both) and is unconditionally dependent on both A and B. As shown in Figure 7, it is possible that the variables X and Y belong to S_{AB}, but the variables Z and W certainly do not since they are independent of A and B.

Therefore, during the first step of CSPI algorithm, edges A-B will be temporarily removed by a true edge that belongs to S_{AB}, or a true edge that belongs to S_{AB} will be temporarily removed by A-B as in Figure 8; the deletion log will keep this information as in Table 5. Assuming that adding X and Y to a set S will not increase the value of I(A;B|S), and no true edge will be removed by any CI test, all the edges connecting A and B to a parent in S_{AB} will eventually be added back to the network, and the test I(A;B|S_{AB}) will be performed.

A mistake could occur in a situation similar to the one described in Figure 9 and Table 6, where the edge X-B is temporarily removed by the edge X-A, which is also temporarily removed. In this case if the algorithm tried to removed X-B first, it could fail to perform the CI test I(A; B|S_{AB}). But as described in Section 3.3, CSPI will first make a recursive call to take care of the edge X-A as described in Figure 10.

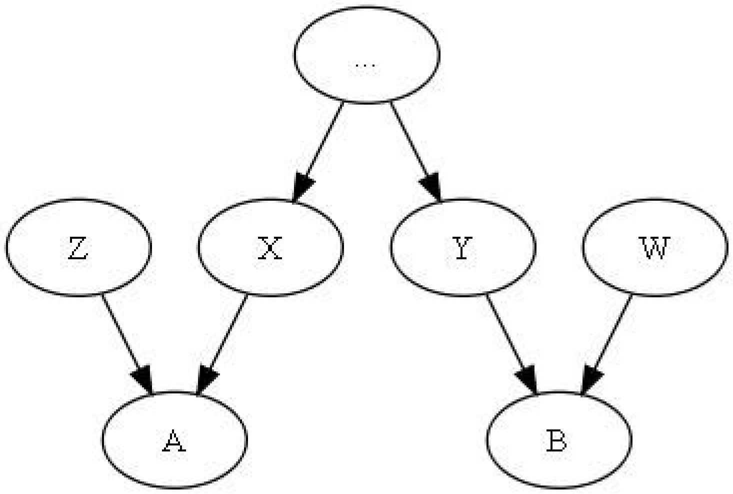

Figure 6 - Network representing conditional independence between *A* and *B*: variables *X* and *Y* can belong to S_{AB}, *Z* and *W* cannot because they are independent of *A* or *B*

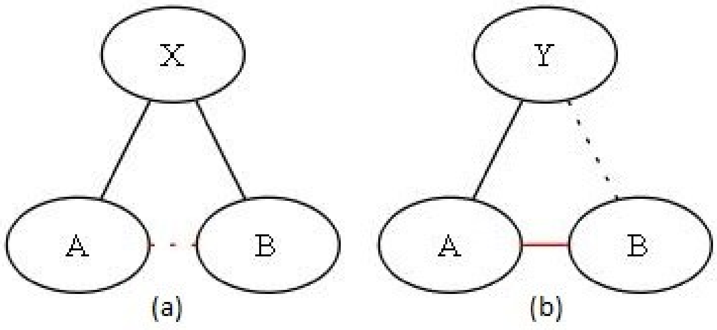

Figure 7 - Edge representing spurious information *A-B*: Or *A-B* is removed by a true edge (a), or it is responsible for the removal of a true edge (b), the corresponding deletion log is shown in Table 10

Table 10- The deletion log represented in Figure 8, spurious edge A-B is removed by a true edge or is responsible for the removal of one

	A	***B***	***X***	***Y***
A	---	X	---	---
B	X	---	---	A
X	---	---	---	---
Y	---	A	---	---

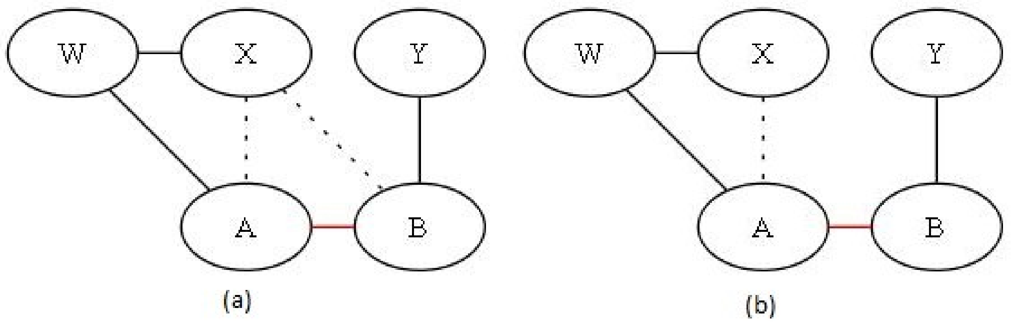

Figure 8 - A mistake could be made if the algorithm first removes edge *X-B* (I(*X;B*|*Y*)< ε), in this case I(*A;B*|*X;Y*) could never be performed, the corresponding deletion log is shown in Table 11

Table 11 - Deletion log representing mistake that could be made, represented in Figure 9

	A	***B***	***X***	***Y***	***W***
A	---	---	W	---	---
B	---	---	A	---	---
X	W	A	---	---	---
Y	---	---	---	---	---
W	---	---	---	---	---

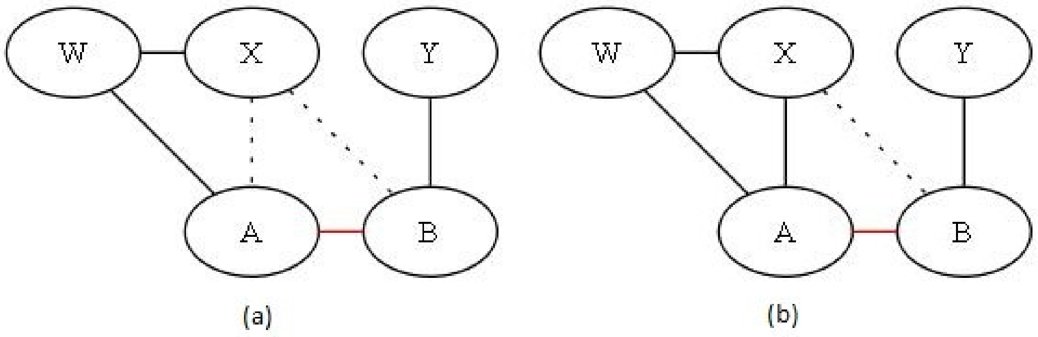

Figure 9 - When algorithm tries removing edge *X-A* first, it will not be able to do it (since it is a true edge) and *X-A* will be added back to the network before the algorithm tries to remove *X-B*, and I(*A;B*|*X;Y*) will be performed

CHAPTER 4

RESULTS AND DICUSSION

4.1 Results

We compared the CSPI algorithm with other cutting-edge algorithms with respect to structure correctness and number of statistical tests (which impacts the run-time of the algorithm). For structure correctness, we used the measure structural Hamming distance (SHD) – sum of the numbers of missing edges, edges missing direction, inverted edges and extra edges. We also compared the number of conditional independence (CI) tests performed by CSPI with the algorithms Max-Min Hill climbing (Tsamardinos, Brown and Aliferis 2006) and Recursive Autonomy Identification (Yehezkel, and Lerner 2009).

In order to perform the tests for the Parent Candidate (PC) algorithm, Three Phase Dependency Analysis (Cheng et al. 2002) and MMHC, we used the Causal Explorer Package (Aliferis, Tsamardinos, Statnikov and Brown 2003); for the RAI algorithm, we used the code provided by its authors.

4.1.1 Alarm and Insurance Networks

We first performed tests for a broad range of conditional independence thresholds using 20 samples of training sets (each containing 20,000 observations) from the benchmark networks Alarm Network and Insurance Network. We then selected the threshold that gave the best results (lowest SHD), and using this threshold we performed tests with 20 other samples (test samples), each containing 20,000 observations. We report the results of structure correctness SHD and number of CI tests for these test datasets. For the tests performed with the CSPI algorithm, we used two different thresholds for unconditional mutual information (ε_u) and conditional mutual information (ε_c). The reason for the adoption of different thresholds is the different normalization methods used in Equation 7 and Equation 10, and as we show, the best results were found when $\varepsilon_u \neq \varepsilon_c$.

Figure 10 shows the average SHD found for the CSPI algorithm for unconditional mutual information threshold within the range *[4e-04:5e-02]* and conditional mutual information *[1e-03:5e-02]* using 20 training sample datasets of the Alarm network. The results for the test sets are shown in Table 12 with mean and standard deviation, and Table 13 shows average and standard deviation of the number of CI tests performed.

Figures 11 and 12 shows the results for the training set for algorithms PC and TPDA , and Figures 13 and 14, the results for RAI and MMHC.

For the samples of the Alarm Network tested, the algorithm Max-Min Hill-Climbing had the best results of structure correctness, followed by the CSPI algorithm, and CSPI had the lowest number of CI tests (Tables 12 and 13).

Figure 15 shows the average SHD found for the CSPI algorithm for unconditional mutual information threshold within the range *[1e-04:1e-02]* and conditional mutual information *[4e-04:1e-02]* using 20 training sample datasets of the Insurance network. Figures 16 and 17 shows the results for algorithms TPDA and PC, and Figures 18 and 19 for RAI and MMHC.

For the samples of Insurance Network, MMHC had the lowest network error followed by CSPI, but MMHC had a large number of CI tests (Table 13). The lowest number of CI tests was found for the RAI algorithm followed by CSPI.

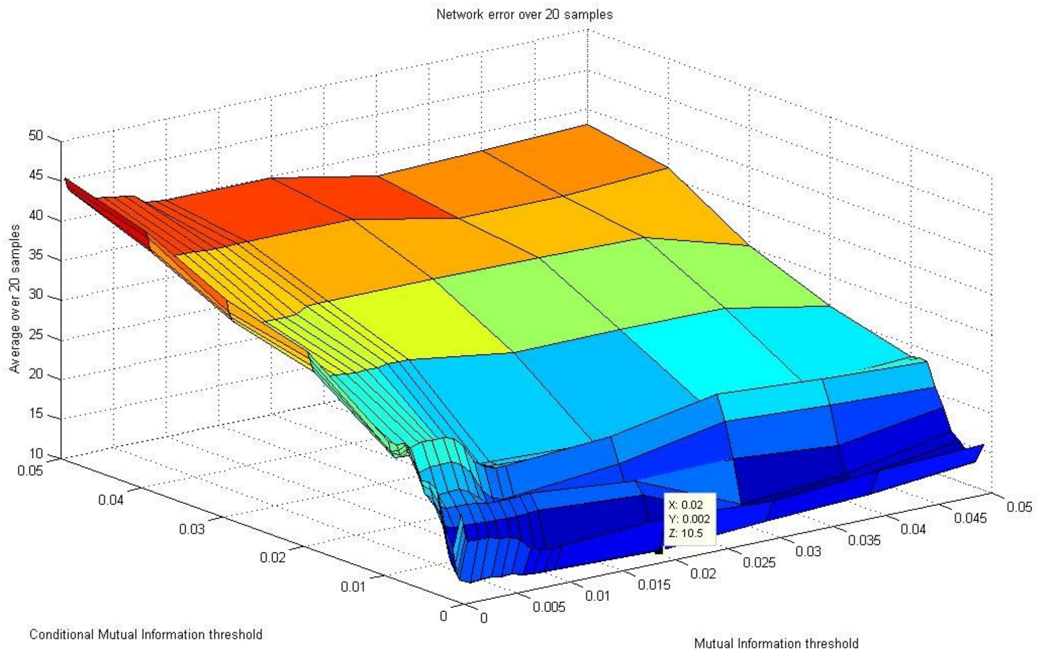

Figure 10 - SHD Results for CSPI Algorithm using 20 training datasets containing 20,000 observations of Alarm network. Minimum network error found: SHD=10.5 for $\varepsilon_{\underline{u}}$=2e-02 and ε_c=2e-03

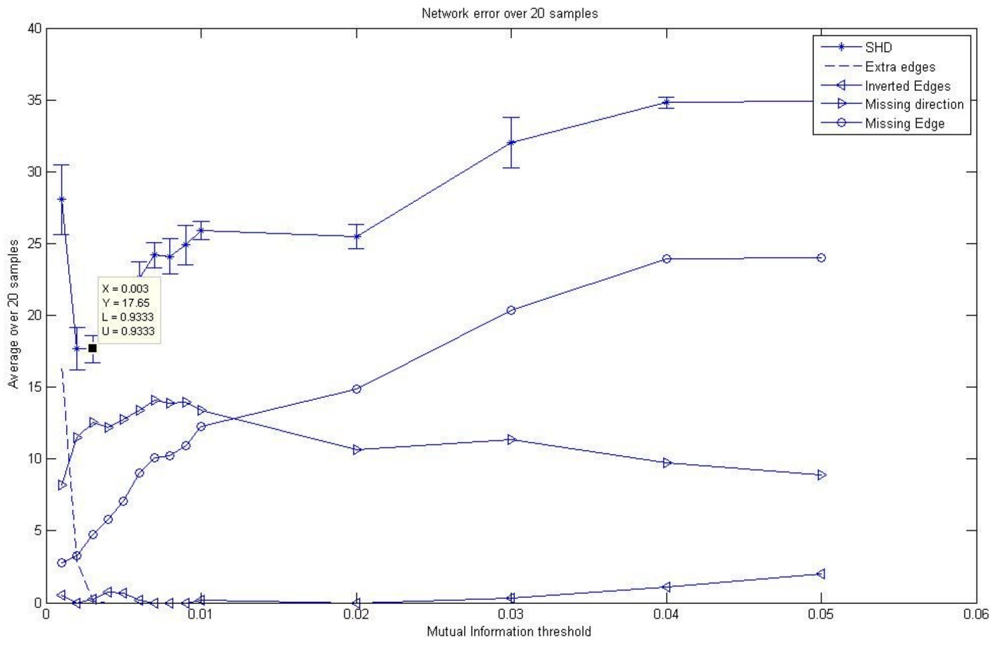

Figure 11 - SHD Results for PC Algorithm using 20 training datasets containing 20,000 observations of Alarm network. Minimum network error found: SHD=17.65 for ε=3e-03

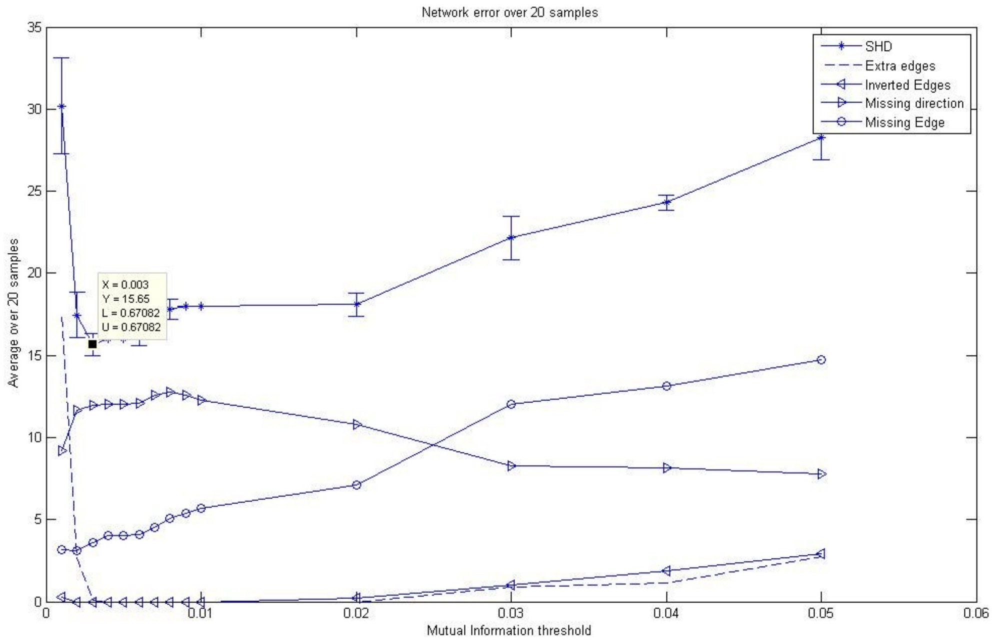

Figure 12 - SHD Results for TPDA Algorithm using 20 training datasets containing 20,000 observations of Alarm network. Minimum network error found: SHD=15.65 for ε=3e-03

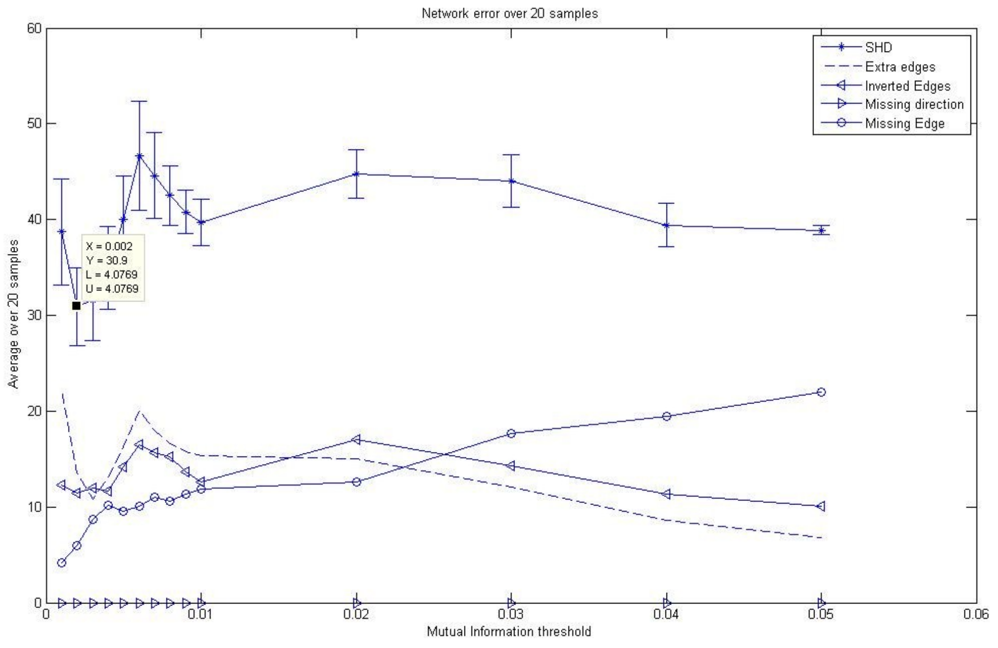

Figure 13 - SHD Results for RAI Algorithm using 20 training datasets containing 20,000 observations of Alarm network. Minimum network error found: SHD=30.9 for ε=2e-03

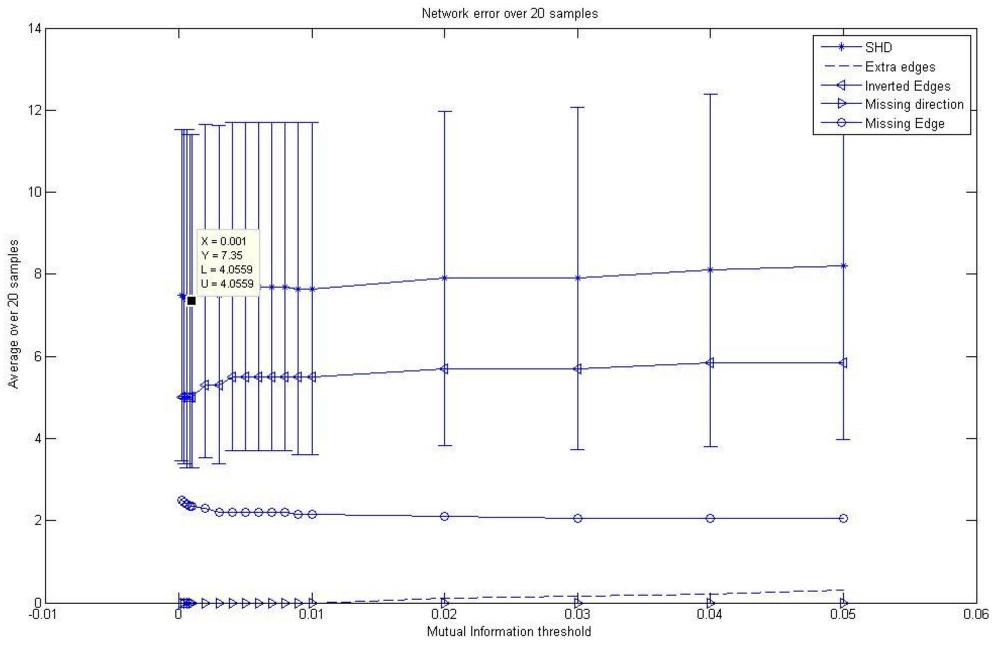

Figure 14 - SHD Results for MMHC Algorithm using 20 training datasets containing 20,000 observations of Alarm network. Minimum network error found: SHD=7.35 for ε=1e-03

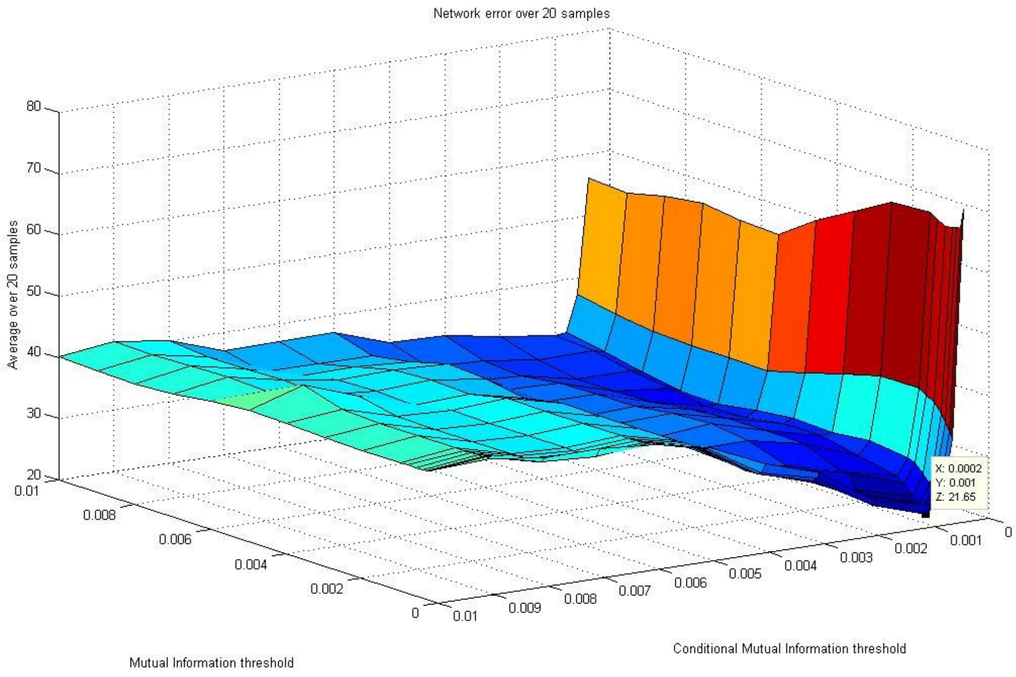

Figure 15 - SHD Results for CSPI Algorithm using 20 training datasets containing 20,000 observations of Insurance network. Minimum network error found: SHD=21.5 for ε_u=2e-04 and ε_c=1e-03

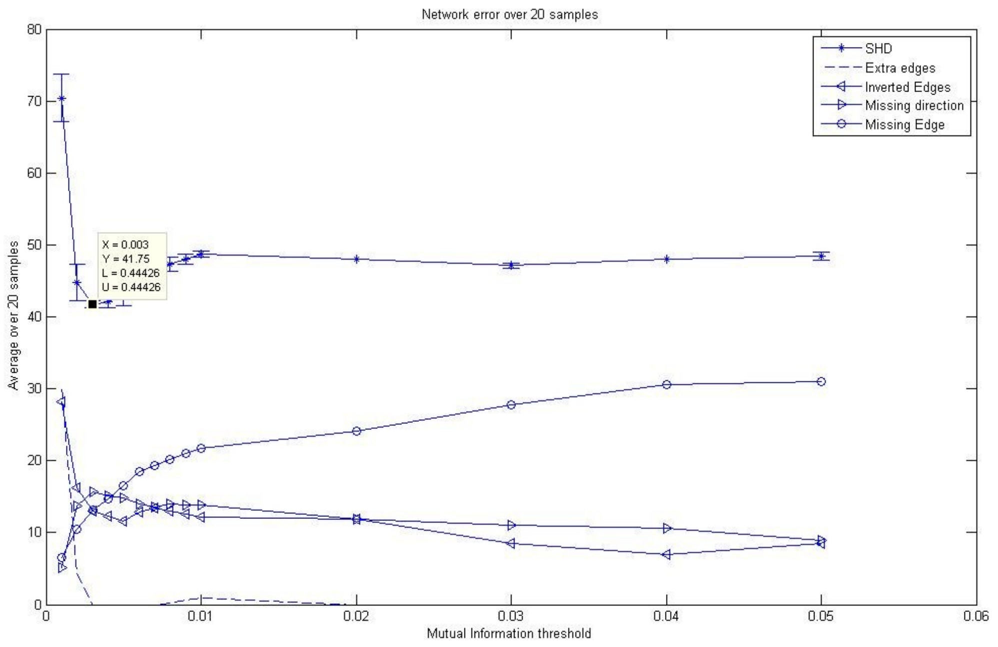

Figure 16 - SHD Results for PC Algorithm using 20 training datasets containing 20,000 observations of Insurance network. Minimum network error found: SHD=41.75 for ε=3e-03

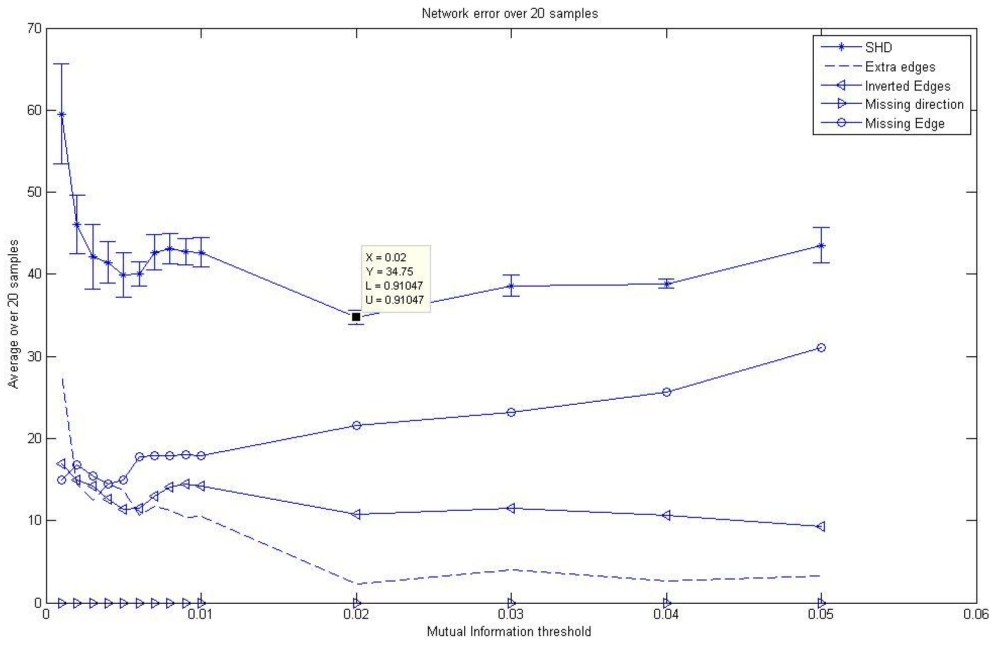

Figure 17 - SHD Results for TPDA Algorithm using 20 training datasets containing 20,000 observations of Insurance network. Minimum network error found: SHD=39.65 for ε=3e-03

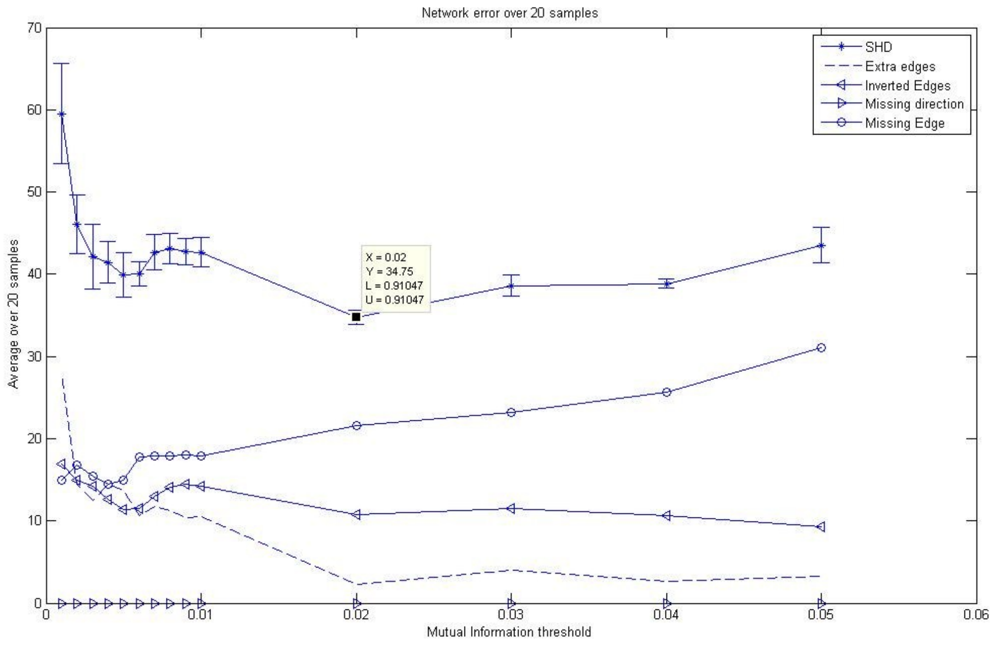

Figure 18 - SHD Results for RAI Algorithm using 20 training datasets containing 20,000 observations of Insurance network. Minimum network error found: SHD=34.74 for ε=2e-03

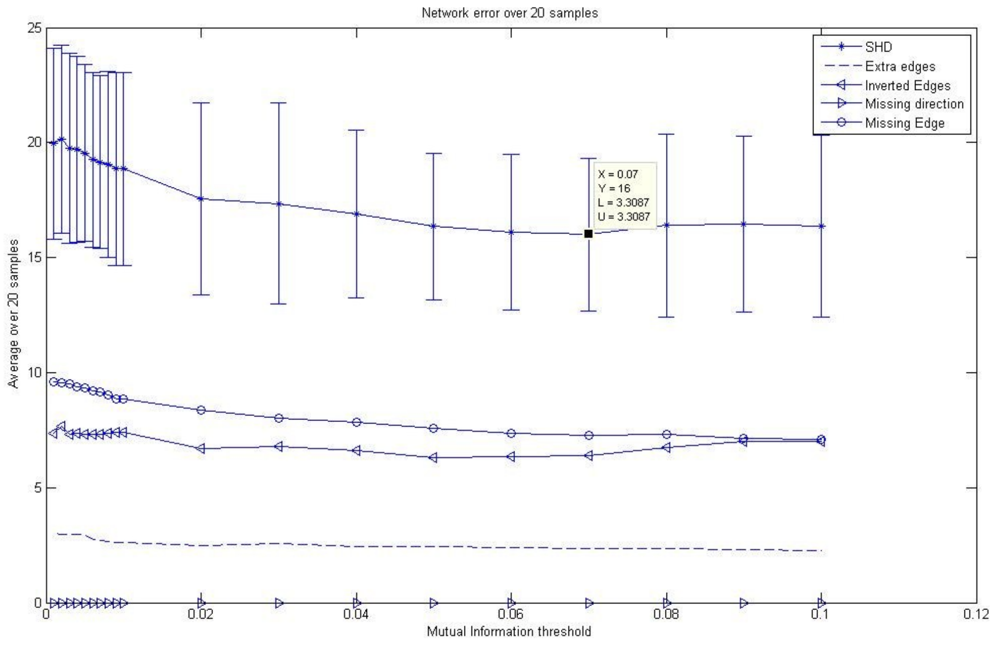

Figure 19 – SHD Results for MMHC Algorithm using 20 training datasets containing 20,000 observations of Insurance network. Minimum network error found: SHD=16 for ε=7e-02

Table 12 – Network error (SHD) for 20 samples of 20,000 observations

	Alarm Network		Insurance Network	
Algorithm	threshold	result [mean(std)]	threshold	result [mean(std)]
MMHC	0.0010	5.95 (2.8924)	0.07	16.70 (3.0625)
CSPI	0.02/0.002	10.50 (0.9459)	0.0002/0.001	22.40 (4.2227)
RAI	0.002	30.90 (4.0769)	0.02	35.50 (0.7609)
TPDA	0.003	15.85 (0.9333)	0.003	39.50 (1.8778)
PC	0.003	17.50 (1.1471)	0.003	41.50 (0.7609)

Table 13 - Number of CI test – Alarm and Insurance

	Alarm Network	Insurance Network
Algorithm	number CI tests [mean(std)]	number CI tests [mean (std)]
MMHC	1,690.00 (18.0380)	13,262.00 (916.6056)
CSPI	893.50 (11.4409)	2,391.80 (154.0173)
RAI	1,500.00 (254.4058)	346.05 (7.6400)

4.1.2 Barley, Child and HailFinder

Another round of tests was performed using the training datasets and test datasets given in the Casual Explorer package. Three of the networks - Barley, Chid and HailFinder – were tested using the algorithms CSPI, MMHC and RAI, following the same procedure used for the network Alarm and Insurance. We found the thresholds that gave the best result for structure correctness using 10 training datasets of 10,000 observations each. Using these thresholds we found the SHD and number of CI tests for the algorithms CSPI, MMHC and RAI using 10 other datasets (test sets), also containing 10,000 observations each.

The results found for average SHD are shown in Table 14, and the number of CI tests performed in Table 15.

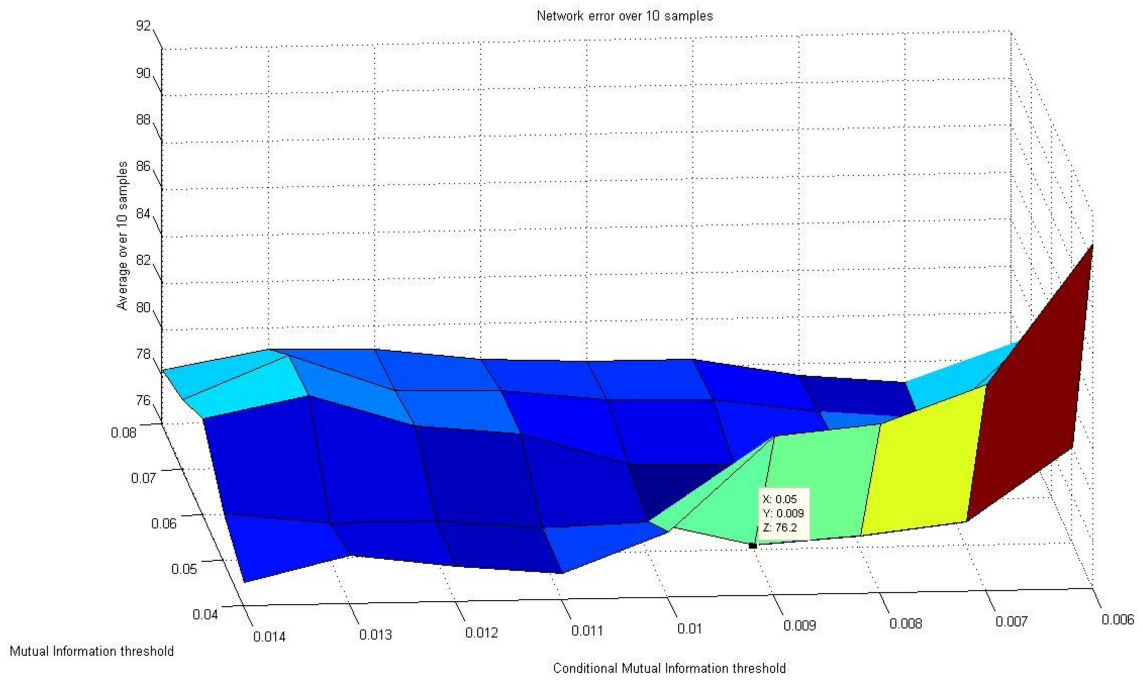

Figure 20 - SHD Results for CSPI Algorithm using 10 training datasets containing 10,000 observations of Barley Network. Minimum network error found: SHD=76.2 for ε_u=5e-02 and ε_c=9e-03

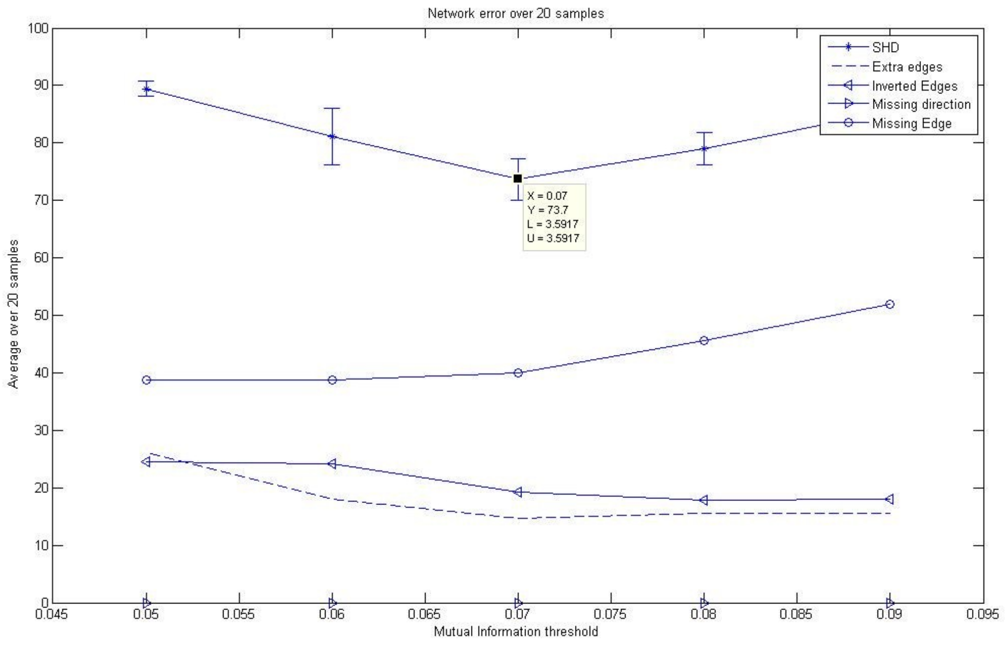

Figure 21 - SHD Results for RAI Algorithm using 10 training datasets containing 10,000 observations of Barley Network. Minimum network error found: SHD=73.7 for ε=7e-02

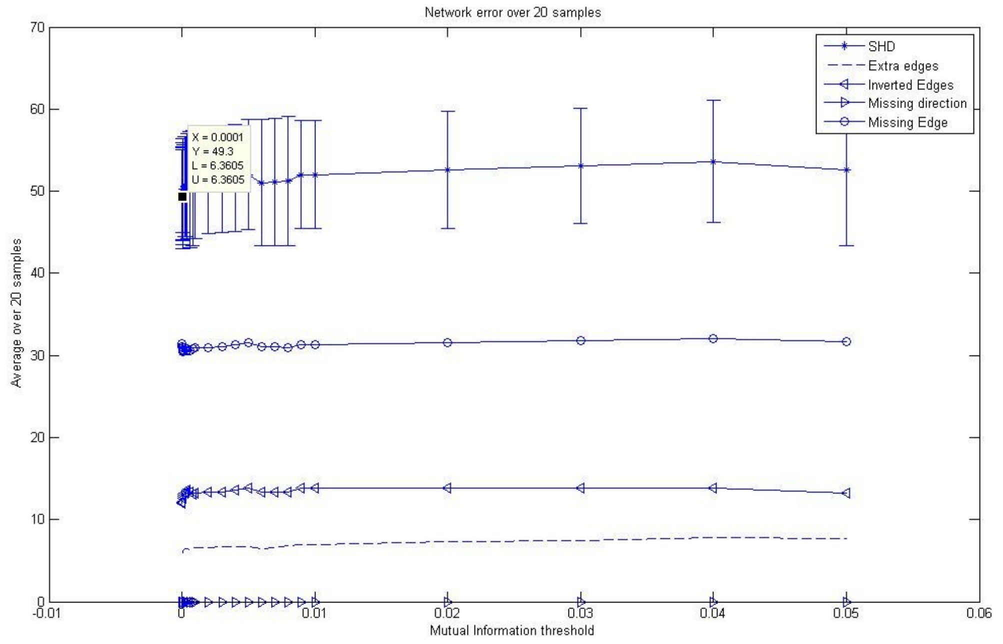

Figure 22 - SHD Results for MMHC Algorithm using 10 training datasets containing 10,000 observations of Barley Network. Minimum network error found: SHD=49.3 for ε=1e-04

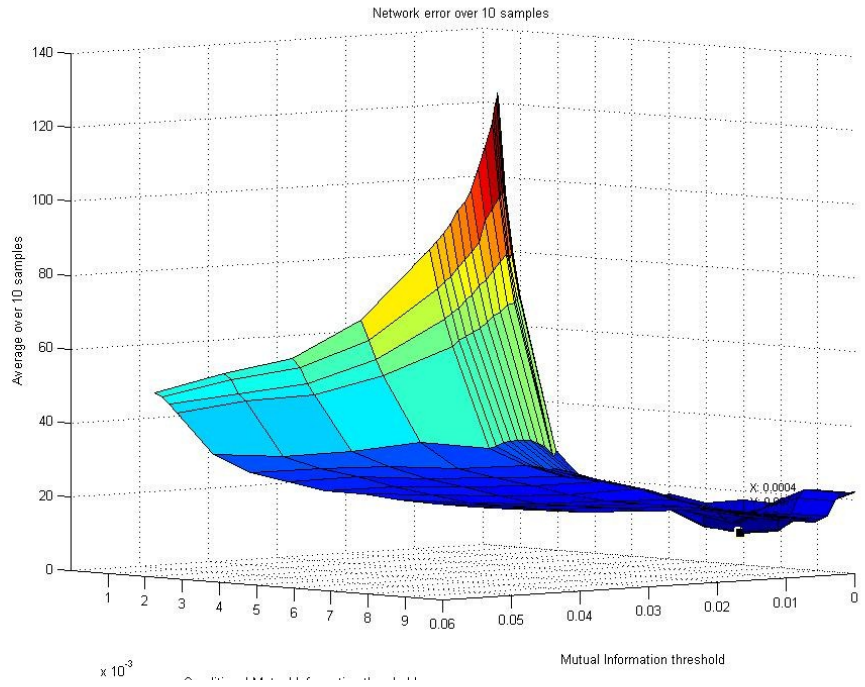

Figure 23 - SHD Results for CSPI Algorithm using 10 training datasets containing 10,000 observations of Child Network. Minimum network error found: SHD=8.5 for εu=4e-04 and εc=7e-03

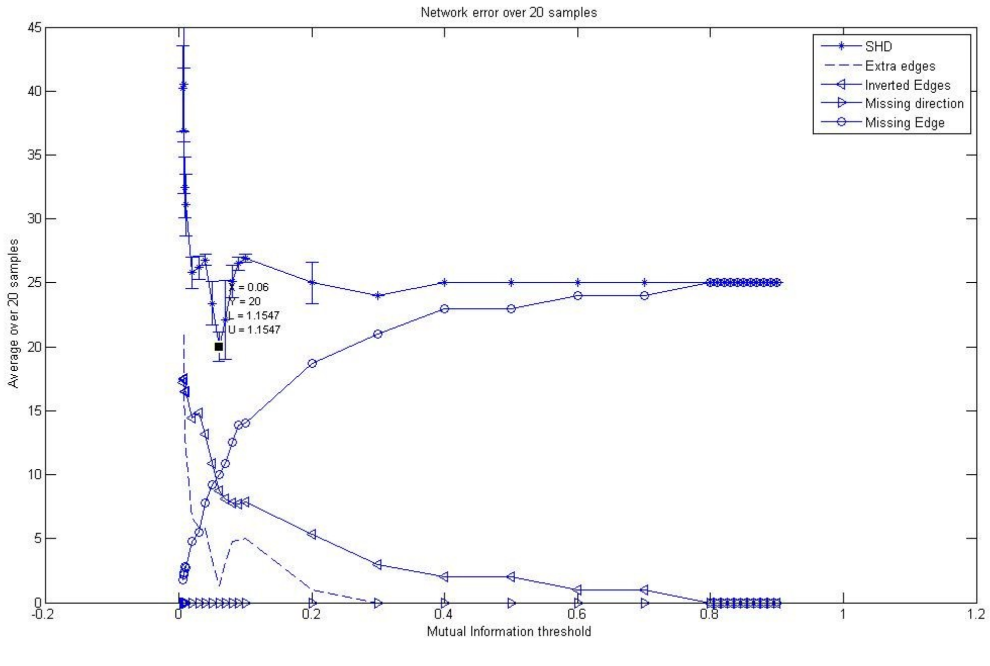

Figure 24 - SHD Results for RAI Algorithm using 10 training datasets containing 10,000 observations of Child Network. Minimum network error found: SHD=20 for ε=6e-02

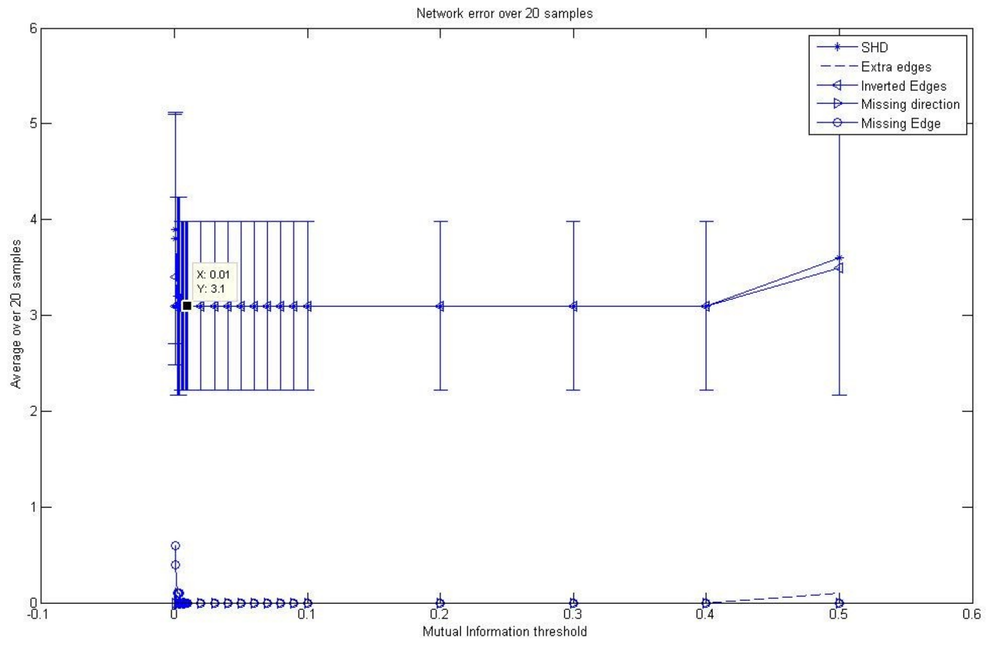

Figure 25 - SHD Results for MMHC Algorithm using 10 training datasets containing 10,000 observations of Barley Network. Minimum network error found: SHD=3.1 for ε=1e-01

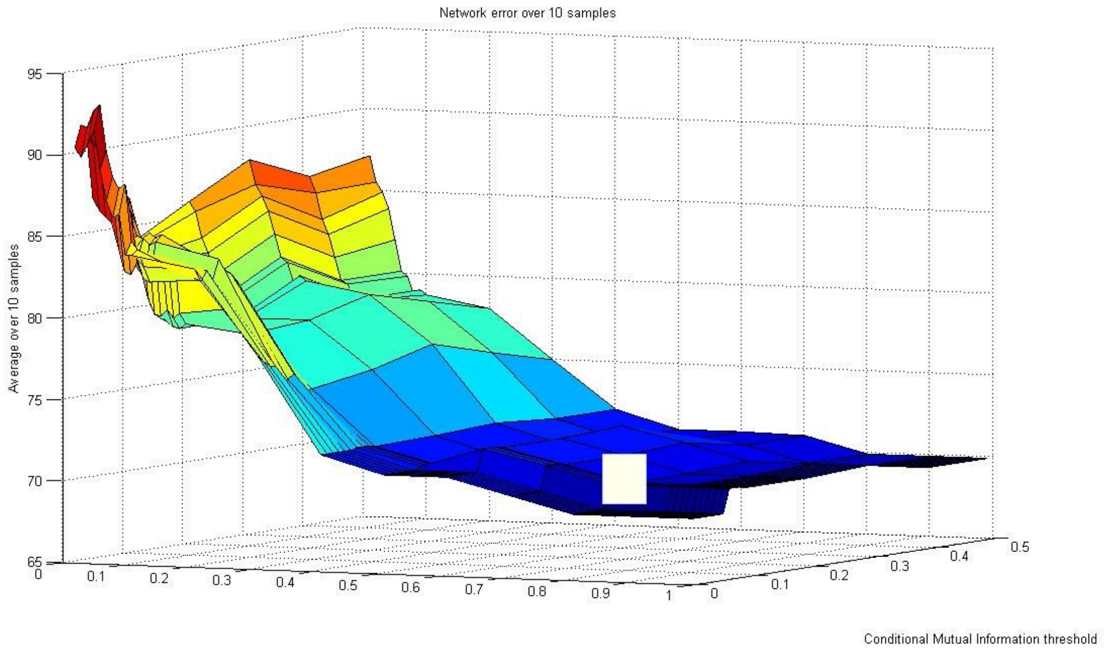

Figure 26 - SHD Results for CSPI Algorithm using 10 training datasets containing 10,000 observations of Child Network. Minimum network error found: SHD=69 for ε_u=8e-01 and ε_c=5e-02

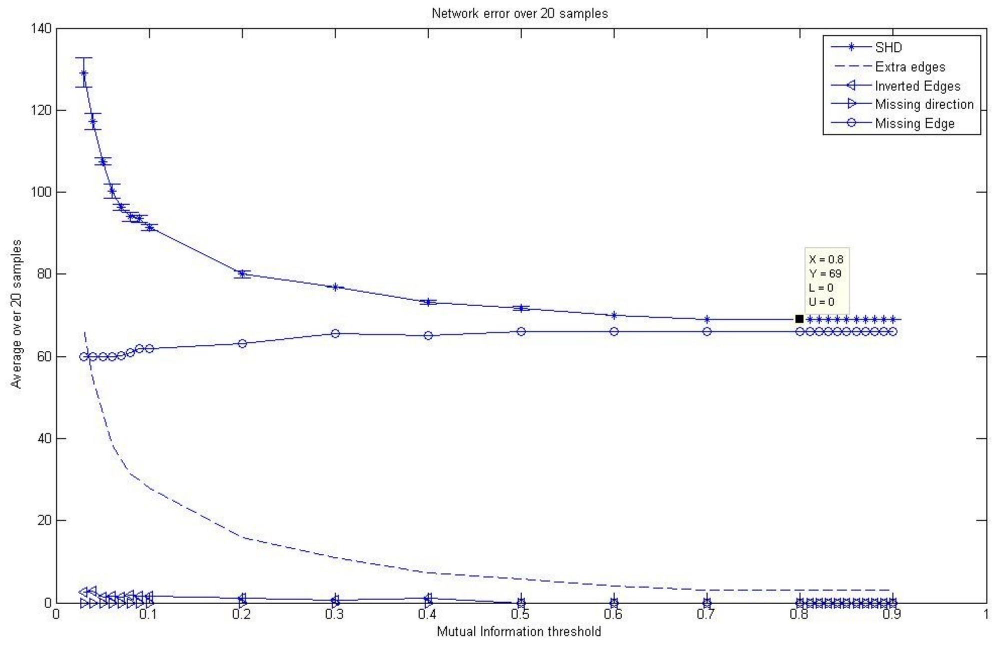

Figure 27 - SHD Results for RAI Algorithm using 10 training datasets containing 10,000 observations of Child Network. Minimum network error found: SHD=69 for ε=8e-01

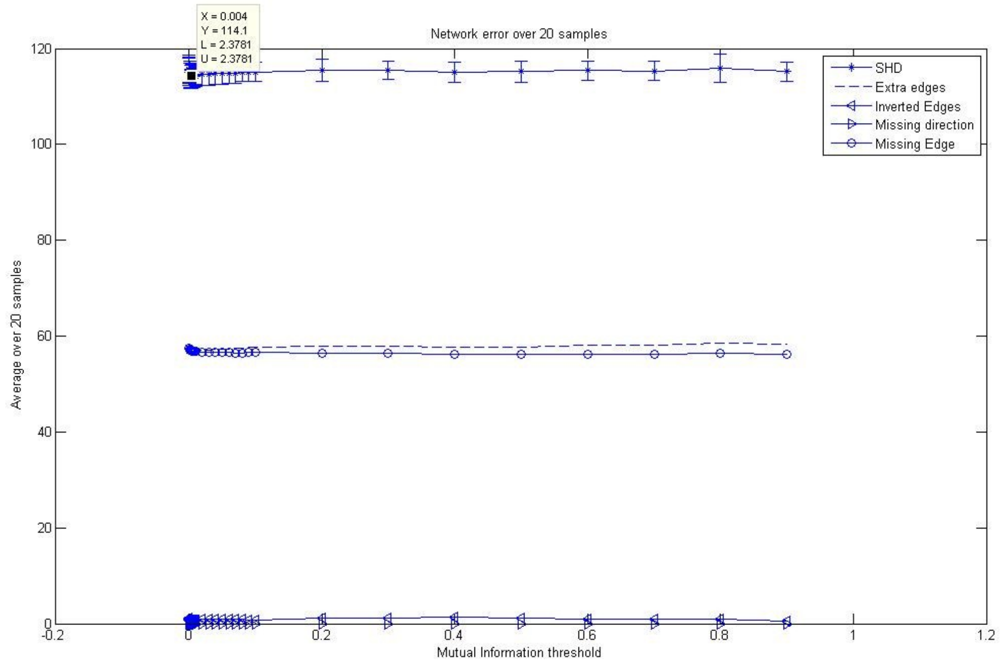

Figure 28 - SHD Results for MMHC Algorithm using 10 training datasets containing 10,000 observations of Barley Network. Minimum network error found: SHD=114.1 for ε=4e-03

Table 14 - Network error (SHD) for 10 samples of 10,000 observations

	Barley Network		Child Network		HailFinder Network	
Algorithm	threshold	[mean(std)]	threshold	[mean(std)]	threshold	[mean(std)]
MMHC	0.0001	56.3 (7.9310)	0.01	3.8 (1.3166)	0.004	114.10 (1.4491)
CSPI	0.05/0.009	76.3 (2.1628)	0.0004/0.007	8.4 (1.4298)	0.8/0.05	69 (0)
RAI	0.07	76.4 (1.4298)	0.8	21.5 (1.7798)	0.8	69 (0)

Table 15 - Number of CI tests - Barley, Child and HailFinder

	Barley Network	Child Network	HailFinder Network
Algorithm	CI tests [mean(std)]	CI tests [mean(std)]	CI tests [mean(std)]
MMHC	3,633.5 (84.5317)	2,329.80 (56.8952)	23,903.0 (803.2591)
CSPI	2,651.4 (56.4962)	846.80 (31.8427)	1,566.0 (6.7897)
RAI	1,465.5 (9.1928)	288.10 (18.3573)	1,544 (0)

4.2 Discussion

As our results show, the CSPI algorithm had equal or better results for structure correctness for all the networks tested when compared to the RAI algorithm, and had equal or lower number of CI tests for all the networks tested when compared to the MMHC algorithm. A summary of the results is shown in Figures 29 and 30.

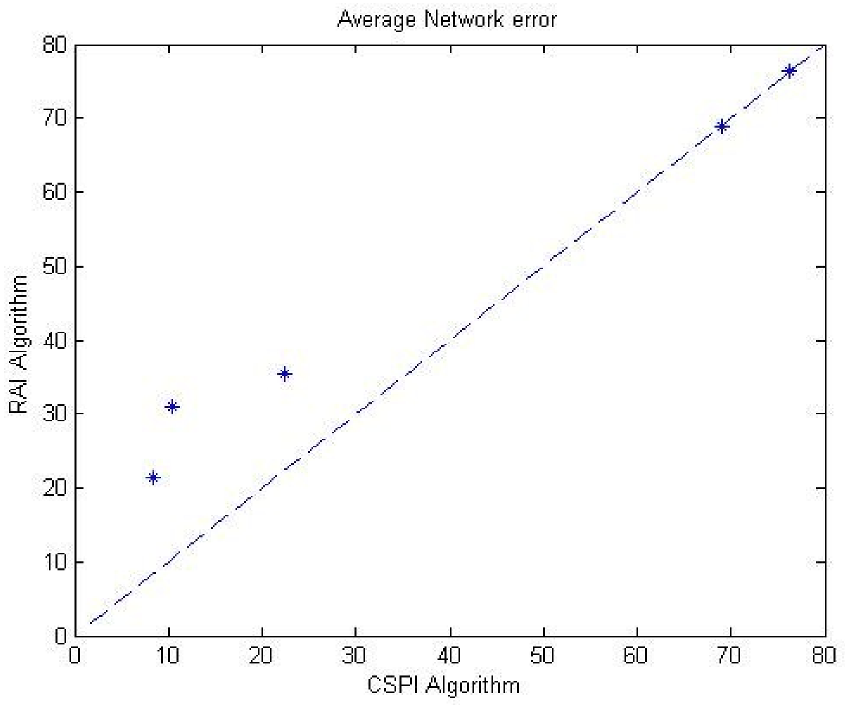

Figure 29 - Comparison of structure correctness (SHD) between CSPI and RAI Algorithms for the 5 networks mentioned in this paper, the dotted line represents the same network error for both algorithms

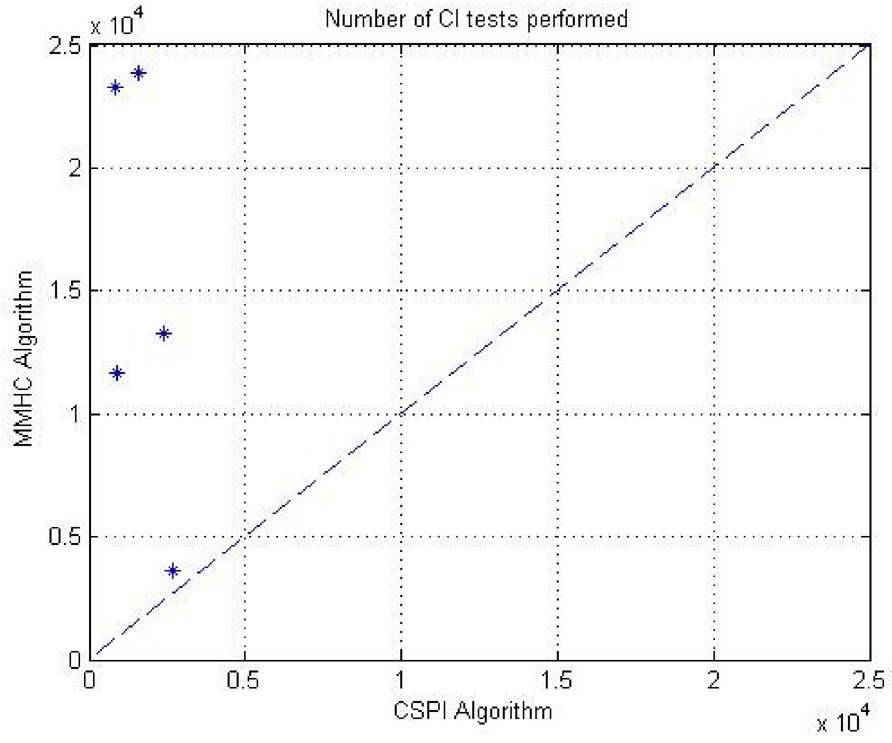

Figure 30 - Comparison of number of CI tests performed between CSPI and MMHC Algorithms for the 5 networks mentioned in this paper, the dotted line represents the same number of CI tests for both algorithms

CHAPTER 5

CONCLUSION AND FUTURE WORK

We have presented a new algorithm for Bayesian network structure learning called Control of Spurious Pairwise Information (CSPI). The algorithm proposes the reduction of number of conditional independence tests, consequently reducing the run-time by allowing an increase in complexity with respect to the number of vertices of the network, without affecting the accuracy of the structure learnt. The algorithm combines greedy hill-climbing search to limit the size of condition sets and a mechanism of storing edges responsible for the removal of other edges that limits the number of CI tests performed.

Comparing the CSPI algorithm with PC, TPDA, MMHC and RAI algorithms, in tests performed for this paper, the CSPI algorithm had very satisfactory results especially with regard to the reduction of number of CI tests performed, consequently reducing the run-time of the algorithm without compromising the accuracy of the discovery of the networks. Some of the limitations of the current algorithm are the exclusive use of mutual information as measure of dependence and the need for discrete data. As future work, we intend to implement and allow the user the selection of other measures of dependence and the use of continuous data.

REFERENCES

Aliferis, C. F., Tsamardinos, I., Statnikov, A., & Brown, L. E. (2003). Causal explorer: A causal probabilistic network learning toolkit for biomedical discovery. In International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences (METMBS 03), 371-376.

Besson, P. et al. (2010). Bayesian networks and information theory for audio-visual perception modeling. Biological Cybernetics, 103, 213-226.

Buntine, W. (1996). A guide to the literature on learning probabilistic networks from data. IEEE Trans. on Knowledge and Data Engineering, 8, 195-210.

Cheng, J. et al. (2002). Learning Bayesian networks from data: An information-theory based approach. Artificial Intelligence, 137, 43-90.

Chickering, D.M. (1996). Learning Bayesian networks is NP-complete. In Fisher,D. & Lenz, H.-J. (eds). Learning from Data: Artificial Intelligence and Statistics V (pp. 121-130). Springer-Verlag.

Chickering, D.M. (2002). Learning equivalence classes of Bayesian-network structures. Journal of Machine Learning Research, 2, 445-498.

Chickering, D.M., Heckerman, D., & Meek, C. (2004). Large-sample learning of Bayesian networks is NP-hard. Journal of Machine Learning Research, 5, 1287-1330.

Chow, C.K., & Liu, C.N. (1968). Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory, 14, 462-467.

Cooper, G.F., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. Machine Learning, 9, 309-347.

Elidan, G., Nachman, I., & Friedman, N. (2007). ”Ideal Parent” structure learning for continuous variable Bayesian networks. Journal of Machine Learning Research, 8, 1799-1833.

Cornalba, C., & Giudici, P. (2004). Statistical models for operational risk management. Physica A, 338, 166-172.

de Oude, P., & Pavlin, G. (2009). Dependence discovery in modular bayesian networks. Technical report, Informatics Institute, University of Amsterdam, The Netherlands.

Friedman, N. (2004). Inferring cellular networks using probabilistic graphical models. Science, 303, 799-805.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. Machine Learning, 29, 131-163.

Friedman, N., Goldszmidt, M., & Wyner, A. (1999). Data analysis with Bayesian networks: A bootstrap approach. In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI), 206-215.

Friedman, N., Linial, M., Nachman, I., & Peer, D. (2000). Using Bayesian networks to analyse expression data. Computational Biology, 7, 601-620.

Friedman, N., Nachman, I., & Peer, D. (1999). Learning Bayesian network structure from massive datasets: The ”Sparse Candidate” algorithm. In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI ’99)’, 196-205.

Strehl, A. & Ghosh, J. (2002). Cluster Ensembles - A knowledge reuse framework for combining multiple partitions. Journal of Machine Learning Research, 3, 583-617.

Grossman, D., & Domingos, P. (2004). Learning Bayesian network classifiers by maximizing conditional likelihood. In Proceedings of the Twenty-first International Conference on Machine learning, 65, 31-78.

Heckerman, D. (1995). A tutorial on learning with Bayesian networks. Technical Report TR-95-06, Microsoft Research.

Heckerman, D., Geiger, D. & Chickering, D.M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. Machine Learning, 20, 197-243.

Lam, W. & Bacchus, F. (1994). Learning Bayesian belief networks: An approach based on the MDL principle. Computational Intelligence, 10, 269-293

Needham, C.J., Bradford, J.R., Bulpitt, A.J., & Westhead, D.R. (2006). Inference in Bayesian networks. Nature Biotechnology, 24(1), 51-53

Pearl, J. (2000). Causality, models, reasoning, and inference (pp. 45-51). Cambridge University Press.

Richiardi, J. (2007). Probabilistic models for multi-classifier biometric authentication using quality measures. Ph.D. thesis, University of Lausanne.

Scutari, M. (2010). Learning Bayesian networks with the bnlearn R Package. Journal of Statistical Software, 35(3), 1-22

Shannon, C.E. (1948). A mathematical theory of communication. The Bell System Technical Journal, 27, 379-423, 623-656

Tsamardinos, I., Brown, L.E., & Aliferis, C.F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. Machine Learning, 65, 31-78.

Wilczynski, B. & Dojer, N. (2009). BNFinder: exact and efficient method for learning Bayesian networks. Bioinformatics, 25, 286-287.

Yao, Y.Y. (2003). Information-theoretic measures for knowledge discovery and data mining. In Entropy Measures, Maximum Entropy Principle and Emerging Applications (pp. 115-136). Karmeshu (ed.), Springer.

Yehezkel, R., & Lerner, B. (2009). Bayesian network structure learning by recursive autonomy identification. Journal of Machine Learning Research, 10, 1527-1570.

VITA

Pablo de Morais Andrade*,* received B.S. in Computer engineering from State University of Campinas, Campinas/SP, Brazil. He is pursuing M.S. in Computer Science at University of Missouri Kansas City. He is a member of the Upsilon Pi Epsilon honor society. His research interests include machine learning/computational intelligence, Bayesian networks, and applications in bioinformatics.