PARALLEL LEARNING OF BELIEF NETWORKS

T. Chu, Y. Xiang
Department of Computer Science, University of Regina
chut@cs.uregina.ca, yxiang@cs.uregina.ca

Abstract

Learning belief networks from a large dataset over a large domain can be computationally expensive even with a single-link lookahead search. It has been shown that a class of probabilistic domain models cannot be learned correctly by several existing algorithms which employ a single-link lookahead search. When a multi-link lookahead search is used, the computational complexity of the learning algorithm further increases. We study how to use parallelism to speed up the learning process. A parallel algorithm for learning belief networks is proposed. Our implementation in a parallel computer demonstrates the effectiveness of the algorithm.

1 INTRODUCTION

A belief network represents probabilistic dependence relationships among variables in a problem domain. The network structure can be a directed or undirected graph, where each node corresponds to a variable in the domain and each link corresponds to dependence between the variables connected. The structure of a Bayesian network (BN) is a directed acyclic graph and that of a decomposable Markov network (DMN) is a chordal graph. As the applicability of belief networks has been demonstrated in different domains, and many effective inference techniques have been developed, the acquisition of such networks from domain experts through elicitation becomes a bottleneck. As an alternative to manual knowledge acquisition, many researchers have actively investigated methods for learning such networks from data [3, 4, 5, 6, 7, 8].

Since learning belief networks in general is NP-hard [2], it is justified to use heuristic search in learning. Many algorithms developed use a scoring metric combined with a search procedure. In these algorithms, a single-link lookahead search is commonly adopted for efficiency. In a single-link lookahead search, consecutive network structures adopted differ by only one link. However, it has been shown that there exists a class of domain models termed pseudo-independent (**PI**) models which cannot be learned correctly by a single-link lookahead search [10, 9].

One alternative for learning PI models is to use multi-link lookahead search, where consecutive network structures may differ by more than one link. Increasing the number of links to search, however, increases the complexity of learning computation.

To speed up the learning process, we propose a parallel algorithm focused on learning DMNs, although our results can be generalized to learning BNs as well. Section 2 illustrates the needs for parallel learning. The parallel learning algorithm is presented in section 3. Section 4 discusses job-balancing which is critical to the efficiency of parallel learning. In section 5, we present our method for balancing jobs among multiple processors. The experimental results are described in section 6.

2 NEED FOR PARALLEL LEARNING

A critical task in learning belief networks is to discover the domain dependence structure. In a systematic single-link lookahead search, $O(N^2)$ alternative structures are checked before one link is added, where N is the number of variables in the domain. Examining each alternative structure takes $O(n)$ time in computing the score of the structure, where n is the total number of cases in the dataset. In general, $O(N^2)$ links can be added. Hence, the complexity of search is $O(N^4\ n)$. Therefore, Learning belief networks in a large domain using a large dataset can be computationally expensive.

A class of probabilistic domain models, called the pseudo-independent (**PI**) models, has been identified which displays a special pattern of dependence relations [9]. It has been shown that several existing algorithms [3, 5, 6, 7], all using a single link lookahead search, cannot learn the correct structure if the underlying domain is a PI model even though different scoring metrics were used such as entropy, conditional independence, the minimal description length and the Bayesian score. PI models do exist in practice. For example, the *parity* problems and the *Modulus addition* problems can both be shown to be special cases of PI models [11].

Table 1: An example of PI models

(X_1, X_2, X_3, X_4)	$P(N)$	(X_1, X_2, X_3, X_4)	$P(N)$
(0,0,0,0)	0.0225	(1,0,0,0)	0.020
(0,0,0,1)	0.2025	(1,0,0,1)	0.180
(0,0,1,0)	0.0050	(1,0,1,0)	0.010
(0,0,1,1)	0.0200	(1,0,1,1)	0.040
(0,1,0,0)	0.0175	(1,1,0,0)	0.035
(0,1,0,1)	0.0075	(1,1,0,1)	0.015
(0,1,1,0)	0.1350	(1,1,1,0)	0.120
(0,1,1,1)	0.0900	(1,1,1,1)	0.080

A PI model with $N = 4$ variables X_i $(i = 1, 2, 3, 4)$ is shown in Table 1. More examples can be found in [9]. It can be easily verified that X_1 and X_4 are conditionally independent given X_2 and X_3. Therefore, X_1 and X_4 should not be *directly* connected. In the subset $\{X_2, X_3, X_4\}$, each pair is marginally dependent, e.g., $P(X_2, X_3) \neq P(X_2)P(X_3)$, and is still dependent given the third, e.g., $P(X_2|X_3, X_4) \neq P(X_2|X_4)$. The corresponding structure should have each pair *directly* connected. However, a special dependence relation exists in the subset $\{X_1, X_2, X_3\}$. Although each pair is dependent given the third, e.g., $P(X_1|X_2, X_3) \neq P(X_1|X_2)$, X_1 and X_2 are marginally independent, i.e., $P(X_1, X_2) = P(X_1)P(X_2)$, so are X_1 and X_3. Since each pair is dependent given the third, the pair should be *directly* connected in the structure. The correct DMN structure is shown in Figure 1 (a).

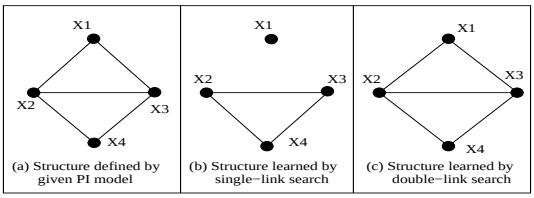

Figure 1: Comparison of learning results

Suppose learning starts with an empty graph (with all the nodes/variables but without any link). In single link lookahead search, since X_1 and X_2 are marginally independent, they will not be directly connected. The same is true for X_1 and X_3, which results in the learned DMN structure in Figure 1 (b). This illustrates the failure of a single-link search.

On the other hand, double link search can effectively tests the dependence among X_1, X_2 and X_3, e.g., whether $P(X_1|X_2, X_3) = P(X_1|X_2)$ holds, and the two links (X_1, X_2) and (X_1, X_3) will be added. The learned structure is shown in Figure 1 (c).

Although PI models can be learned correctly by multi-link search as illustrated above, the number of alternative structures to be checked at each step will increase quickly with the number of links to be added at each step. Consider a problem domain with 37 variables. In a single-link search starting from an empty structure, only 666 structures of a single link are to be checked before one link is added. In a double-link search, it needs to check $666 \times 656/2 = 218448$ structures. For a triple link search, the number of structures becomes 47621664. Clearly, managing the increased computational complexity is critical if we are to learn PI models correctly and effectively. Some alternative solutions are discussed in [10, 11]. In this paper, we study the alternative of parallel learning.

3 THE PARALLEL LEARNING ALGORITHM

The parallel algorithm presented here is an extension to the sequential learning algorithm in [10]. The sequential algorithm learns a DMN from a dataset of cases. Each case is described by the values of a set of discrete variables. Learning starts with an empty structure (graph). Search is organized according to the number i of links to add at each step. At each step, alternative structures that differ from the current structure by i links are evaluated according to the cross entropy score. The one with the best score (decreasing the cross entropy with the largest amount) will be chosen as the new current structure. Search continues until no link may be added to decrease the cross entropy significantly.

We parallelize the learning process based on the following observation: At each step of search, the exploration of alternative structures are coupled only through the current structure, i.e., given the current structure, tests of alternative structures are independent of each other. Hence the tests can be performed in parallel.

We focus on a parallel environment where communication among processors is performed through message passing. We configure the processors as follows. One processor is designated as the search *manager* and the others are structure *explorers*.

Algorithm 1

Input:
A dataset D over a set N of variables, a maximum size η of clique, a maximum number $\kappa \leq \eta(\eta-1)/2$ of lookahead links, the total number n of explorers, and a threshold δh for cross entropy decrement.
begin
send D and η to each explorer;
initialize an empty graph $G = (N, E)$;
$G' := G$;
for $i = 1$ to κ, do
repeat
initialize the cross entropy decrement $dh' := 0$;
partition all graphs differed from G by i links into n sets;
send one set Θ of graphs & G to each explorer;
for each explorer
receive dh^* and G^*;
if $dh^* > dh'$ then $dh' := dh^*$, $G' := G^*$;
if $dh' > \delta h$, then $G := G'$, $done := false$;
else $done := true$;
until done = true;
send a termination signal to each explorer;
return G;
end

Algorithm 2

begin
receive a dataset D and a maximum size η of clique
repeat
receive current graph $G = (N, E)$ and a set Θ of graphs from manager;
if the termination signal is received, then terminate;
initialize the cross entropy decrement $dh^* := 0$ and $G^* := G$;
for each graph $G' = (N, L \cup E)$ in Θ, do
if G' is chordal and L is implied by a single clique of size $\leq \eta$, then compute dh';
if $dh' > dh^*$, then $dh^* := dh'$, $G^* := G'$;
send dh^* and G^* to the manager;
end

The manager executes Algorithm 1. It is responsible for generating alternative graphs based on the current graph. It then partitions all possible graphs into n sets and distributes one set to each explorer. Each explorer executes Algorithm 2. It first checks the chordality for each graph received. Furthermore, to keep the search effective, a graph will be considered only if the set L of new links are contained in a subgraph induced by a single clique. When this is the case, L is said to be *implied* by a clique. The explorer then computes the cross entropy decrement for each valid graph. It chooses the best graph G^* and reports to the manager. The manager collects the reported graphs from all explorers, selects the best, and then starts the next search step.

Figure 2 illustrates the parallel learning with two explorers and a dataset of four variables u, v, x and y. Only a single-link search is performed for simplicity. The manager starts with an empty current graph in (a). It sends six alternative graphs in (b) through (g)

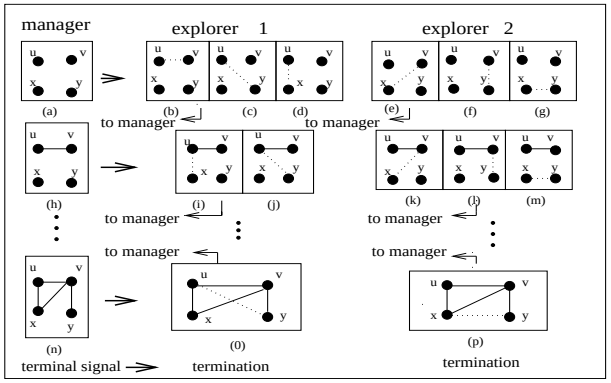

Figure 2: An example of parallel learning of DMN

to explorers 1 and 2. The explorer 1 checks graphs in (b), (c) and (d), selects the one in (b), and reports to the manager. The explorer 2 reports the one in (e) to the manager. After collecting the two graphs, the manager chooses the one in (b) as the new current graph. It then sends graphs in (i) through (m). Repeating the above process, the manager finally gets the graph in (n) and sends graphs in (o) and (p) to explorers. Since none of them decreases the cross entropy significantly, the manager chooses the graph in (n) as the final result and terminates the explorers.

4 NEED FOR JOB BALANCING

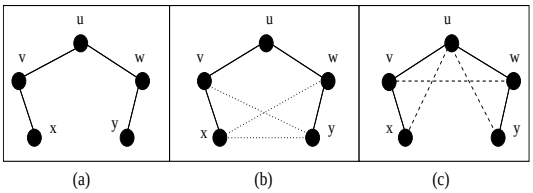

Figure 3: Two types of alternative structures

In Algorithm 1, even allocation of alternative graphs to explorers is used. However, the amount of computation to check each graph tends to switch between two extremes. If a graph is not chordal, it is ignored immediately without having to compute the cross entropy decrement. For example, suppose the current graph is shown in Figure 3 (a). There are six graphs that differ from it by only one link. If any of the dotted links in (b) is added to (a), the resultant graph is non-chordal. The amount of computation to check each of them is very small. If any of the dashed links in (c) is added to (a), the resultant graph is chordal. The amount of computation to check each of them is much larger. As a result, even job allocation may require significantly different amount of computation among explorers. As the manager must collect reports from all explorers before a decision on the new current graph can be made, some explorers

will be idle while other explorers are completing their jobs.

Figure 4 shows the time taken by each of the six explorers in a particular search step. The dataset contains 37 variables. Explorer 1 takes much longer than others. This illustrates the needs for more sophisticated job allocation strategy in order to improve the efficiency of the parallel system.

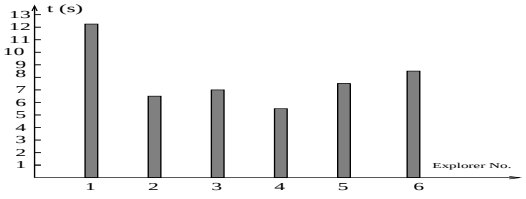

Figure 4: The time needed for each explorer

5 JOB BALANCING STRATEGY

The job balancing problem can be abstracted as follows: Let L_0 be the total number of job units each corresponding to a graph to be checked, and n be the total number of explorers. Each job is either type 0 corresponding to the non-chordal case or type 1 corresponding to the chordal case. It takes time T_0 to process each unit of type 0 job and T_1 for type 1. After an explorer has finished a set of jobs, it takes time T_c to send another set to the explorer. The goal is to find a strategy to allocate jobs to explorers such that the sum of idle time of all explorers is reduced during the completion of L_0 job units. Before deriving our strategy, we state some assumptions.

Assumption 1 *T_0 and T_c are constants.*

T_0 is the computation time to detect a non-chordal graph which can be performed efficiently. T_c is the time taken to send one set of jobs from the manager to an explorer. Both vary slightly and can be treated as constants.

Assumption 2 *T_1 is a constant and is much larger than T_0 and T_c.*

T_1 is the computation time to process one unit of type 1 job which involves computing the cross entropy decrement of a chordal graph. It is much larger than T_0 and T_c. For example, in learning from a dataset with 37 variables, we recorded T_0 between 0.007 to 0.009 seconds and T_c about 0.017 seconds in our parallel computing environment. T_1 is at least 0.06 seconds. However, the assumption that T_1 is a constant is less accurate. A maximal set of nodes pairwise linked in a graph is called a *clique*. When the variation of clique sizes in a chordal graph is small, T_1 tends to be close to a constant. When the variation is large, T_1 tends to vary from job unit to unit. Still, we found the assumption to be useful in deriving the following simple and effective strategy, and we are currently working on relaxing this assumption as discussed in Section 7.

Suppose the manager first allocates J_0 job units to each of the n explorers. Let Q_i and B_i denote the type 1 jobs and type 0 jobs in the J_0 job assignment to explorer i, respectively. Let Q denote the total type 1 jobs in the n J_0 units. Let $\beta_i = Q_i/J_0$ be the percentage of type 1 jobs in the job assignment for explorer i. Let $\beta = Q/(n\ J_0)$ be the percentage of type 1 jobs in the total n J_0 job units. Without losing generality, suppose $\beta_1 = \max_{i=1}^{n}(\beta_i)$ and we denote β_1 by β_{max}.

The time t_i for explorer i to process J_0 job units is

$$\begin{aligned} t_i &= Q_iT_1 + B_iT_0 = \beta_i J_0 T_1 + (1-\beta_i)J_0T_0 \\ &= J_0(\beta_i(T_1 - T_0) + T_0). \end{aligned} \quad (1)$$

Let T be the sum of the idle time of explorers 2 through n while explorer 1 is processing its job. We can derive

$$\begin{aligned} T &= \sum_{i=2}^{n}(t_1 - t_i) \\ &= \sum_{i=2}^{n} J_0((\beta_{max}(T_1 - T_0) + T_0) - (\beta_i(T_1 - T_0) + T_0)) \\ &= \sum_{i=2}^{n} J_0\beta_{max}(T_1 - T_0) - \sum_{i=2}^{n} J_0\beta_i(T_1 - T_0). \end{aligned} \quad (2)$$

Substituting $\sum_{i=2}^{n} J_0\beta_i = Q - Q_1 = nJ_0\beta - J_0\beta_{max}$ into equation (2), we get

$$\begin{aligned} T &= (n-1)J_0\beta_{max}(T_1 - T_0) - (nJ_0\beta - J_0\beta_{max})(T_1 - T_0) \\ &= nJ_0(\beta_{max} - \beta)(T_1 - T_0). \end{aligned} \quad (3)$$

To improve the efficiency, we need to reserve some job units which can be distributed to explorers who finish their first J_0 job units before explorer 1. Let $L_1 = L_0 - nJ_0$ be the number of job units to be reserved and β_r be the percentage of type 1 jobs in L_1 units. Ideally, we would like to distribute the L_1 units to explorers 2 through n such that they will be fully engaged during the t_1 time period and all L_1 units are completed at time t_1.[1] This ideal condition can be expressed as

$$T = L_1(\beta_r(T_1 - T_0) + T_0) + MT_c, \quad (4)$$

where M is the total number of job assignments to be performed. This value of M is dependent on the job

[1] Note that we have used t_1 to denote both the length of the time interval from 0 to t_1 and the instant t_1.

assignment strategy which we shall discuss shortly. Solving equation (3) and (4), we get

$$J_0 = \frac{L_0(\beta_r(T_1 - T_0) + T_0) + MT_c}{n((\beta_{max} - \beta + \beta_r)(T_1 - T_0) + T_0)}. \quad (5)$$

In order to compute J_0, we need the values for β, β_{max}, β_r and M. However, these values are unknown at the beginning of the search step when J_0 is to be computed. We estimate the values of β and β_{max} based on the following assumption:

Assumption 3 *The difference between the values of β (β_{max}) from successive search steps is small.*

Assumption 3 usually holds since the graphs involved in successive steps differ by only a few links. Figure 5 shows the values of β and β_{max} from search step 5 to 75 in learning the ALARM network, which provides an empirical justification of the assumption.

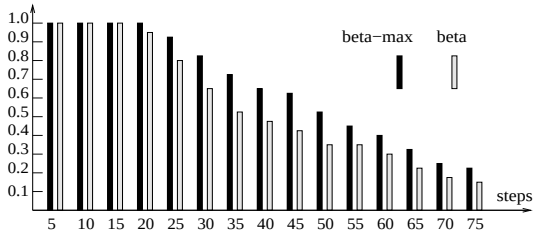

Figure 5: β and β_{max} with 8 structure explorers

The value of β_r usually varies from $\beta_{min} = \min_{i=1}^{n}(\beta_i)$ to β_{max}. We can replace β_r by β_{min} in equation 5 in calculating J_0. Since $\beta_{min} \leq \beta_r$ and $L_0 \gg n$, the value of J_0 will be reduced by the replacement. This implies that more job units will be reserved. The consequence is that the reserved jobs may not be completed by t_1 (by all explorers) and additional job assignments need to be performed. Since the time T_c for each job assignment is very small, a few additional job assignments will not reduce the efficiency significantly.

Finally, we consider the estimation of M. From the numerator of equation 5, we can see that the effect of an inaccurate estimation of M is small. This is because $\beta_{min}(T_1 - T_0) + T_0$ is larger than T_c and L_0 is much larger than M.

Based on Assumption 3 and the above analysis, the manager can collect the values β', β'_{min}, β'_{max} and M' from the last search step to calculate the value for J_0:

$$J_0 \approx \frac{L_0(\beta'_{min}(T_1 - T_0) + T_0) + M'T_c}{n((\beta'_{max} + \beta'_{min} - \beta')(T_1 - T_0) + T_0)}. \quad (6)$$

We now discuss the strategy for allocating the remaining L_1 job units. Consider Figure 4. Suppose that the histogram depicts the computation time of the first J_0 job units by each explorer. Explorer 4 is the first that finishes. Let J_1 be the job units the manager allocates to explorer 4 at this moment. The most conservative strategy will be to allocate $J_1 = L_1/(n-1)$ units, which effectively assumes that every explorer finishes at the same moment. In general, other explorers will finish later and hence this strategy will under-allocate to explorer 4. However, since the under-allocation only slightly increases the number M of job assignments and the effect to the overall efficiency is minor as discussed above, we have adopted this conservative strategy.

In general, let L_2 be the remaining job units after the allocation of J_1 units to the explorer that finishes the first, L_3 be the remaining job units after the allocation of J_2 units to the explorer that finishes the second, and so on. The job units to be allocated to the explorer that finishes the ith place will be

$$J_i = \begin{cases} \frac{L_i}{n-1} & \text{when } L_i \geq 2(n-1) \\ 1 & \text{when } L_i < 2(n-1) \end{cases} \quad (7)$$

where $i = 1, 2, \ldots, M$, and $L_{i+1} = L_i - J_i$.

Note that in equation (7), if the remaining job is less than $2(n-1)$ units, the manager will allocate the job unit by unit.

6 Experimental results

The parallel algorithm is implemented on a MIMD parallel computer of 64 processors. Each processor has its local memory (no shared memory) and communication can only be performed by message passing.

The performance of a parallel program are commonly measured by *speed-up* (S) and *efficiency* (E). Given a task, let $T(1)$ be the execution time of a sequential program and $T(n)$ be that of a parallel program with n processors. The two measurements are defined as $S = T(1)/T(n)$ and $E = S/n$.

Table 2: Experimental results

	Even job allocation			Balanced job allocation		
n	time(s)	S	E	time(s)	S	E
1	3238	1	1	3238	1	1
2	1756	1.84	0.922	1654	1.96	0.979
3	1229	2.63	0.878	1090	2.97	0.990
4	973	3.33	0.832	830	3.90	0.975
5	776	4.17	0.835	686	4.72	0.944
6	720	4.50	0.750	578	5.60	0.934
7	601	5.39	0.770	525	6.17	0.881
8	573	5.65	0.706	471	6.87	0.859
9	531	6.10	0.678	448	7.23	0.803
10	492	6.58	0.658	410	7.90	0.790
11	482	6.72	0.611	381	8.50	0.773
12	467	6.93	0.578	378	8.57	0.714

Table 2 lists the experimental results for learning ALARM network (37 variables) [1] from a dataset

of 10000 cases. Each row is the results obtained by using n explorers as indicated in the first column. The learned structure in each case is identical to the one learned by using sequential learning. Columns 2 through 4 correspond to the results obtained using even job allocation and columns 5 through 7 correspond to the results obtained using the job balancing strategy in Section 5.

Columns 3 and 5 show that as the number of explorers increases, the speed-up increases as well when either job allocation strategy is used. It demonstrates that our parallel algorithm can effectively reduce the learning time. This provides the positive evidence that parallelism is an alternative to tackle the computational complexity in learning belief networks.

Comparing column 3 with 6 and column 4 with 7, it can be seen that the balanced job allocation further speeds up the learning process and improves the efficiency beyond the even job allocation. For example, when six explorers are used, the speed-up is 4.5 and efficiency is 0.75 for even job allocation, and 5.6 and 0.934 respectively for balanced job allocation.

The results also show a gradual decrease in efficiency as the number of explorers increases. We attribute this efficiency decrease mainly to the job allocation cost. The manager must allocate J_0 job units to each explorer sequentially at the beginning of each search step. Therefore, each explorer is idle after its report from the previous search step is submitted and before the next J_0 job units are assigned to it.

7 REMARKS

In this paper, we have studied the alternative of parallelism in speeding up the computation in learning belief networks. We have proposed an algorithm to decompose the learning task such that multiple processors can be used in parallel. The learning result is identical to what will be obtained by a sequential algorithm. We have developed a balanced job allocation strategy which can further increase the learning speed and efficiency. We have implemented our algorithm in a parallel computer and our preliminary experiment showed positive results.

Several directions for further investigation can be identified. We have assumed that the computation time T_1 for processing a type 1 job unit is a constant. This assumption is not valid in general. We are working towards relaxation of this assumption. It is expected to further improve the efficiency. We are also experimenting with a two stage method toward job balancing, where explorers only check the chordality and reports to the manager in the first stage, and the manager distributes all chordal graphs to explorers for processing in the second stage. Our implementation has been tested in a relatively small domain (37 variables) that does not contain PI models[2]. We are currently extending our experiments on larger and more complex domains.

Acknowledgement

This work is supported by grants OGP0155425, CRD193296 from the Natural Sciences and Engineering Research Council of Canada, and by the Institute for Robotics and Intelligent Systems in the Networks of Centres of Excellence Program of Canada.

References

[1] I.A. Beinlich , H.J. Suermondt, R.M. Chavez, and G.F. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks, *Tech. Report KSL_88-84, Knowledge Systems Lab., Medical Computer Science, Stanford Univ.*, 1989.

[2] D. Chickering, D. Geiger and D. Heckerman. learning Bayesian networks: search methods and experimental results, In *Proc. 5th Conf. on Artificial Intelligence and Statistics*, pages 112-128, 1995.

[3] G.F. Cooper and E.H. Herskovits. A Bayesian method for the induction of probabilistic networks from data, *Machine Learning*, (9), pages 309-347, 1992.

[4] D. Heckerman, D. Geiger, and D.M. Chickering. Learning Bayesian networks: the combination of knowledge and statistical data. *Machine Learning, 20*, pages 197-243, 1995.

[5] E.H. Herskovits and G.F. Cooper. Kutato: an entropy-driven system for construction of probabilistic expert system from database, In *Proc. 6th Conf. on Uncertainty in Artificial Intelligence*, pages 54-62, 1990.

[6] W. Lam and F. Bacchus. Learning Bayesian networks: an approach based on the MDL principle. *Computational Intelligence*, 10(3) pages 269-293, 1994.

[7] P. Spirtes and C. Glymour. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, 9(1), pages 62-73, 1991.

[8] S.K.M. Wong and Y. Xiang. Construction of a Markov network from data for probabilistic inference. In *Proc. Third International Workshop on Rough Sets and Soft Computing*, San Jose, CA, pages 562-569, 1994.

[9] Y. Xiang, S.K.M. Wong, and N. Cercone. Critical remarks on single link search in learning belief networks. *Twelfth Annual Conference on Uncertainty in Artificial Intelligence* (UAI-96), pages 564-571, 1996.

[10] Y. Xiang, S.K. Wong and N. Cercone. A 'Microscopic' Study of Minimum Entropy Search in Learning Decomposable Markov Networks. *Machine Learning*, Vol.26, No.1, pages 65-92, 1997.

[11] Y. Xiang, Learning belief networks in pseudo-independent domains, Tech. Report, CS-96-07, Univ. of Regina, 1996.

[2]Whether the domain contains PI models does not affect how parallelism should be performed as can be seen from Algorithms 1 and 2.