GPU Acceleration of EDA-based Algorithms to Learn Bayesian Networks

Takashi Mori[†*], Yuma Yamanaka[†], Takatoshi Fujiki[†], and Takuya Yoshihiro[‡**]
[†]Graduate School of Systems Engineering, Wakayama University, Japan
[‡]Faculty of Systems Engineering, Wakayama University, Japan
[*] s171053@sys.wakayama-u.ac.jp
[**] tac@sys.wakayama-u.ac.jp

Abstract - Bayesian Network is a graphical model that expresses causal relationship among events, which is regarded to be useful in decision making in various practical scenes. A number of algorithms to learn Bayesian Network structure from data have been proposed so far, but since the problem to learn Bayesian Network structure is proved to be NP-hard, it takes considerable time to learn sub-optimal structures. As one of the efficient approximation algorithm to obtain good (but not optimal) solution in practical time, EDA-based genetic algorithms are used. However, it still takes time to compute the solution on a CPU. Thus, in this paper, we propose a method to accelerate the EDA-based algorithm by designing parallel execution of the algorithm using GPUs. Through evaluation, we show that the proposed algorithm runs about 14-times faster than the original one executed on CPU. We also compare the quality of Bayesian network models created by major approaches in the literature, and found that EDA-based algorithm is superior to the others. We concludes that the proposed algorithm to learn Bayesian network structures is good in both quality and computational speed.

Keywords: Bayesian Networks, GPU, EDA, PBIL

1 INTRODUCTION

Bayesian Networks (BNs) are regarded as useful graphical models used to analyze causal relationship among events. There are so many practical fields in which Bayesian Networks are effectively utilized, such as bioinformatics research, medical analyses and diagnosis, computer security, system diagnosis and monitoring, etc. Recently, because we are surrounded by so much data coming from the Internet, sensors embedded to the environment, or various information systems, the importance of Bayesian Networks as analytic tools is continuously growing larger and larger.

A large number of studies have been dedicated to learn good BNs efficiently in the literature. It is well-known that BN structure learning can be formulated as an optimization problem that optimizes a model score defined as an information criterion. However, because it is proved to be NP-hard [1], to solve the problem approximately within practical time is significantly important. One traditional method is K2 [2], which introduce the constraint on variable order to reduce the search space. The variable order is the constraint on events $n_1, n_2, \ldots, n_k$ where n_i can be a parent of n_j only if $n_i \prec n_j$. However, because the order constraint such that $n_i \prec n_j$ cannot be defined frequently (i.e., it is possibly defined only in the apparent case, for example, n_i occurs before n_j in time), there are many cases in practice in which order constraint is not applicable.

To solve the optimization problem without order constraint, many studies tried to find sub-optimal BN models. Recently, algorithms based on genetic algorithms (GAs) are well-studied, as shown in the survey article [6]. Among them, we in this paper focus on a kind of GA-based algorithm called EDA (Estimation of Distribution Algorithms), in which the distribution of model scores on the graph space is estimated in order to find better-score Bayesian Network models efficiently. Note that EDA is one of the representative methods to compute BN models efficiently and approximately.

Specifically, we treat a EDA-based algorithm called PBIL (Probability-Based Incremental Learning) [12], which is reported to be the best to learn BNs among EDAs-based algorithms [7]. The problem here is that the above algorithms including PBIL-based one take significant time to learn BNs. To cope with the large data available today, acceleration of those algorithms to run within shorter time is important.

In this paper, we present a method to accelerate a PBIL-based BN-learning algorithm to run much faster using GPU computation. Our method incorporates parallel computation on GPU based on the coalesce access technique to accelerate the calculation of model scores such as AIC. We evaluated the proposed method using well-know bench-mark data sets, and found that the algorithm achieve about 14-times faster running speed than the original CPU-executed algorithm using consumer-class GPU hardware. We further compared the quality (i.e., optimality) of the solusions between major algorithm to learn Bayesian Networks in order to confirm the performance of PBIL-based algorithms. As a result, we found that the PBIL-based algorithm is superior to the other major approaches to learn BNs. From above, we conclude that the algorithm proposed in this paper computes good solutions within a short time by utilizing GPUs.

This paper is organized as follows. In Sec. 2, we describe the major approaches of algorithms to learn BNs in the literature. In Sec. 3, we introduce the optimization problem to learn BN structures and give the specific description of PBIL-based BN-learning algorithm. After we concisely explain the GPU architecture related to our work in Sec. 4, we describe the proposed method to accelerate the PBIL-based algorithm using GPU computation in Sec. 5. In Sec. 6 we evaluate the running speed and the quality of solution of the proposed method, we finally conclude the work in Sec. 7.

2 RELATED WORK

In this section, we concisely introduce the major approaches to learn BNs presented in the literature.

Local search is one of the most basic approaches to solve large-space optimization problems, including BN learning. Greedy hill climbing (GHC), which is the most basic one, explores the search space by moving to the best-score point among the vicinity of the current point. In learning BNs, GHC is usually applied to search the graph space. There are several variants in this sort of algorithms such as MMHC[4].

Simulated annealing (SA) is also a well-known local-search-based algorithm used in learning BNs [5], which is different from GHC in that SA does not always move to better-score point; SA has a parameter T called temperature, which is decreased gradually as iteration proceeds. In each iteration, SA selects randomly a point in the vicinity, and moves to it if the score of the new point is better, otherwise moves to it with the probability determined by T.

Genetic algorithms (GAs) are frequently used in NP-hard problems to obtain approximate solutions in a practical time. In GAs, *crossover* and *mutation* operators are applied to obtain a set of next-generation individuals. For learning BNs, one major strategy is called K2GA[3][8][9], in which GA is applied to evolve the order constraint from which K2 heuristic generates a graph structure. There are other ways to use GA to learn BNs such as graph evolution [10] and co-evolution algorithms [11] that evolve directly the graph structure.

As a kind of GA, EDA-based algorithms also have applied to learn BNs. In EDA, the distribution of scores in the search space is estimated using a set of individuals in a generation, and update the probability vector to generate individuals of the next generation. PBIL (Probability Based Incremental Learning), which is one of EDA, is first proposed in [12], first applied to learn BNs in [13], and reported to outperform other EDAs in learning BNs[7]. PBIL stops running when the probability vector converges. To avoid terminating and continue searching, several mutation operators such as bitwise mutation (BM)[14], transpose mutation (TM)[7], and probability mutations (PM)[15] have been proposed. Later, efficient repetition technique called PBIL-RS is proposed in [16] and reported to outperform the above mutation operators.

A few parallel algorithms to learn Bayesian Networks have been proposed in the literature. Nikolova, et al., proposed a parallel algorithm that searches graph space in parallel using conditional independence tests [19]. However, since this algorithm runs on multi-CPU platforms, it is not comparable to our algorithm that runs on GPU. Of course, utilizing muiti-CPU hardware is a possible choice. However, our choice, i.e., using GPUs, has an advantage that we can use well-populated and cheeper hardware to accelerate the structure learning of Bayesian networks. Linderman[20] and Wang[21], respectively, proposed an algorithm that accelerate MCMC sampling in the ordering space. Althrough MCMC-based algorithm could be a powerful choice to learn Beysian Networks, they are usually applied to the case in which each node (i.e., events) takes a real value and its statistical distribution is assumed. However, since in this paper we treats the case of discrete values, especially binary values in many cases, MCMC-based methods are too heavy and time-consuming. No parallel algorithm for GPU that is suitable to treat discrete-value cases has not been proposed so far.

3 PRELIMINARIES

3.1 Problem Formulation

A Bayesian Network model is a graphical model that represents the causal relationship among events. A Bayesian Network model has a structure represented by a directed graph where events are denoted by nodes while causal relationships are denoted by directed edges. In many cases (including this work), each node takes multinomial discrete values, and conditional probabilities among them are expressed by a model. See Fig. 1 for a concise example. Nodes n_1, n_2, and n_3 represent distinct events, where they take 1 if the corresponding events occur, and take 0 if the events do not occur (in this case we show a binomial case for conciseness). Edges $n_1 \rightarrow n_3$ and $n_2 \rightarrow n_3$ represent causal relationships, which mean that the probability of occurrence for each n_3 value depends on the values of n_1 and n_2. If edge $n_1 \rightarrow n_3$ exists, we call that n_1 is a parent of n_3 and n_3 is a child of n_1. Because nodes n_1 and n_2 do not have their parents, they have own prior probabilities $P(n_1)$ and $P(n_2)$. On the other hand, because node n_3 has two parents n_1 and n_2, it has a conditional probability $P(n_3|n_1, n_2)$. In this example, the probability that n_3 occurs is 0.890 under the assumption that both n_1 and n_2 occur. Note that, from this model, Bayesian inference is possible: if n_3 is known, then the posterior probability of n_1 and n_2 can be determined, which enables us to infer more accurately the occurrence of events.

The Bayesian Networks model can be learned from the data obtained through the observation of events. Let $N = \{n_i\}$, $(1 \leq i \leq |N|)$ be a set of events, and $O = \{o_j\}$, $(1 \leq j \leq |O|)$ be a set of observations, where $|N|$ is the number of events and $|O|$ that of observations. Let $o_j = (x_{j1}, x_{j2}, \ldots, x_{j|N|})$ be j-th observation, which is a set of observed values x_{ji} on event n_i for all $i (1 \leq i \leq |N|)$. We try to learn a good Bayesian Network model m from the given set of observations. Note that, good Bayesian Network model m is the one that creates data sets similar to the original observation O. As an model score (i.e., evaluation criterion) to measure the level of fitting between m and O, several information criteria such as AIC (Akaike's Information Criterion) [17] are used. Formally, the problem of learning Bayesian Networks that we consider in this paper is defined as follows:

Problem 1: From the given set of observations O, find a Bayesian Network model m that has the lowest model score.

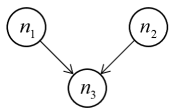
3.2 PBIL

In PBIL, an individual creature m is defined as a vector $m = \{e_1, e_2, \ldots, e_L\}$, where $e_i (1 \leq i \leq L)$ is the i-th element that takes a value 0 or 1, and L is the number of elements that consist of an individual. Let $P = \{p_1, p_2, \ldots, p_L\}$ be a probability vector where $p_i (1 \leq i \leq L)$ represents the probability to be $e_i = 1$. The algorithm of PBIL is described as follows:

(1) As initialization, we let $p_i = 0.5$ for all $i = 1, 2, \ldots, L$.

(2) Generate a set M that consists of $|M|$ individuals according to probability vector P, i.e., element e_i of each

n_1	$P(n_1)$
0	0.79
1	0.21



n_2	$P(n_2)$
0	0.88
1	0.12

n_1	n_2	$P(n_3 \mid n_1, n_2)$	
		0	1
0	0	0.999	0.001
0	1	0.710	0.290
1	0	0.360	0.640
1	1	0.110	0.890

Figure 1: An Example of Bayesian Network Models

P		Parent Node					
		n_1	n_2	…	n_i	…	$n_{\|N\|}$
Child Node	n_1	0.0	0.5	…	p_{i1}	…	0.5
	n_2	0.5	0.0	…	p_{i2}	…	0.5
	⋮	⋮	⋮	⋱	⋮	…	⋮
	n_j	p_{1j}	p_{2j}	…	p_{ij}	…	p_{Nj}
	⋮	⋮	⋮	⋮	⋮	⋱	⋮
	$n_{\|N\|}$	0.5	0.5	…	p_{iN}	…	0.0

Figure 2: A Probability Vector

individual is determined by the corresponding probability p_i.

(3) Compute the score for each individual $m \in M$.

(4) Select a set of individuals M^{Topk} whose members have evaluation scores within top k in M, and update the probability vector according to M^{Topk}. Specifically, the formula applied to every p_i to update the probability vector is shown as follows.

$$p_i^{\mathrm{new}} = ratio(i) \times \alpha + p_i \times (1 - \alpha), \qquad (1)$$

where p_i^{new} is the updated value of the new probability vector (p_i is replaced with p_i^{new} in the next generation), $ratio(i)$ is the function that represents the ratio of individuals in M^{Topk} that include edge i (i.e., $e_i = 1$), and α is the parameter called learning ratio.

(5) Repeat steps (2)-(4) until P converges.

By merging top-k individuals, PBIL evolves the probability vector such that the good individuals are more likely to be generated. Different from other genetic algorithms, PBIL does not include "crossover" between individuals. Instead, it evolves the probability vector as a "parent" of the generated individuals.

3.3 PBIL-based Bayesian Networks Learning

In this section, we describe a PBIL-based algorithm that learns BN models. Because our problem (i.e. Problem 1) to learn BN models is a little different from the general description of PBIL shown in the previous section, a little adjustment is required. In our problem, individual creatures correspond to each BN model. Namely, with the set of events N, an individual model is represented as $m = \{e_{11}, e_{12}, \ldots, e_{1|N|}, e_{21}, e_{22}, \ldots, e_{|N|1}, e_{|N|2}, \ldots, e_{|N||N|}\}$ where e_{ij} corresponds to the edge from an event n_i to n_j, i.e., if $e_{ij} = 1$, the edge from n_i to n_j exists in m, and if $e_{ij} = 0$ it does not exist. Similarly, we have the probability vector P to generate individual models as $P = \{p_{11}, p_{12}, \ldots, p_{1|N|}, p_{21}, p_{22}, \ldots, p_{|N|1}, p_{|N|2}, \ldots, p_{|N||N|}\}$ where p_{ij} is the probability that the edge from n_i to n_j exists. A probability vector can be regarded as a table as illustrated in Fig. 2. Note that, because BNs do not allow self-edges, p_{ij} is always 0 if $i = j$. The process of the BN-learning algorithm is basically obtained from the steps of the general PBIL, as described in the following (See also Fig. 3 that illustrate these steps).

(1) Initialize the probability vector P as $p_{ij} = 0$ if $i = j$, and $p_{ij} = 0.5$ otherwise, for each $i, j (1 \leq i, j \leq |N|)$.

(2) Generate M as a set of $|M|$ individual models according to P.

(3) Compute the evaluation scores for all individual models $m \in M$.

(4) Select a set of individuals M^{Topk} whose members have top-k evaluation values in M, and update the probability vector according to the formula (1).

(5) Repeat steps (2)-(4) until P converges.

Same as the genearal PBIL, the BN-learning algorithm evolves the probability vector so that we can generate better individual models. However, there is a constraint specific to BNs, that is, a BN model is not allowed to have cycles in it. To consider this constraint in the algorithm, step 2 is detailed as follows:

(2a) Consider every pair of events (i, j) where $1 \leq i, j \leq |N|$ and $i \neq j$, create a random order of them.

(2b) For each pair (i, j) in the order created in step (2a), determine the value e_{ij} according to P; every time e_{ij} is determined, if e_{ij} is determined as 1, we check whether this edge from n_i to n_j creates a cycle with all the edges determined to exist so far. If it creates a cycle, let e_{ij} be 0.

(2c) Repeat steps (2a) and (2b) until all the pairs in the order are processed.

These steps enable us to learn good BN models within the framework of PBIL.

3.4 PBIL-RS

Note that PBIL introduced above does not include mutation operators. Therefore, naturally, it easily converges to a local minimum solution. To avoid converging to the local minimum solution and to continuously improve the solution after that, several mutation operators have been proposed such as Bitwise Mutation (BM) [14], Transpose Mutation (TM) [7], and Probability Mutation (PM) [15]. PBIL-RS (PBIL-Repeated Search) [16] is also a method to avoid converging to local minimum solution, which, when it detects convergence,

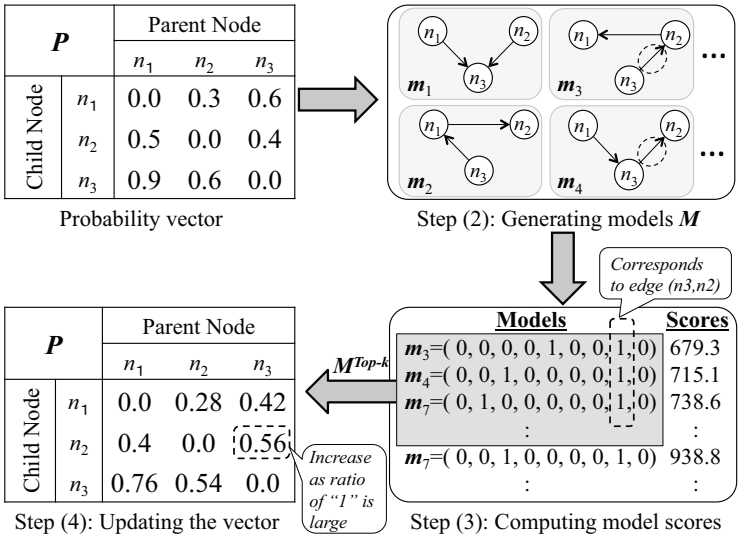


Figure 3: Overview of PBIL

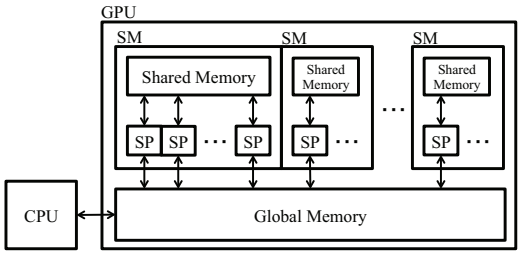


Figure 4: GPU Architecture

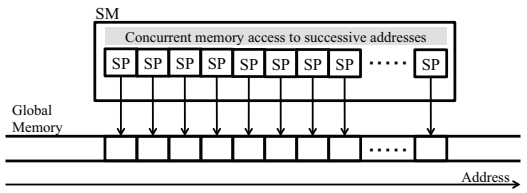


Figure 5: Coalesce Access

spreads the search area again. PBIL-RS is shown to find better solutions compared to the mutation-based methods such as BM, TM, and PM by repeating spreading and converging [16].

4 GPU ARCHITECTURE

4.1 GPU Structure

GPU (Graphics Processing Unit) is an arrayed processor that is originally developed to accelerate graphical computing, which currently is used to accelerate general scientific computation. A GPU structure is illustrated in Fig. 4. A GPU has a hierarchical structure where it consists of multiple (tens to hundreds of) Streaming Multi-processors (SMs) and a SM further includes multiple (tens to hundreds of) Streaming Processors (SPs). Since each SP in a SM concurrently executes a fragment of program code, a GPU executes a number of fragmented codes in parallel, which potentially results in significant performance.

To exchange data between a CPU and a GPU, a memory called *global memory* is prepared in the GPU that can be accessed by both the CPU and the GPU. Also, to accelerate parallel computation, each SM has a small high-speed memory called *shared memory* that can be accessed by all SPs in the same SM. A thread runs in each SP, and the threads in the same SM runs the same bytecode in parallel with different values of variables. Thus, memory accesses of threads are expected to occur simultaneously. To optimize the efficiency of the parallel access is the key issue to design algorithms for GPU. Note that, if the number of threads to execute exceeds the number of SPs in a SM, a single SP executes multiple threads in turn until all of them are executed.

4.2 Coalesce Access to Global Memory

To have as much performance gain as possible from GPU, one of the most basic techniques is to consider the efficient access to the global memory called *coalesce access*, which is illustrated in Fig. 5. Coalesce access is a synchronized parallel access technique in which threads in a SM simultaneously access the successive addresses in the global memory to achieve high-throughput memory access. When the access is scheduled completely to the successive addresses, the SM read/write the memory block in a single action that completes the access of all SPs. To utilize coalesce accesses is an important technique in designing GPU algorithms.

5 ACCELERATING PBIL WITH GPU

5.1 Overview

The method we propose in this paper extends PBIL and its family algorithms (such as PBIL-RS and the mutation extensions) to run in significantly shorter time by means of parallel computation of GPU. We re-designed Step (3) of PBIL described in Sec. 3 for GPU execution to compute the model scores for a collection of models M. Because, in PBIL, hundreds of models are to be computed in a single generation to estimate a distribution of model scores, introducing parallel computation in Step (3) is significantly effective.

Specifically, we detailed the step (3) in the following.

(3a) Transporting data from CPU to the global memory.

(3b) For each model m, we compute the evaluation score by executing the following substeps (3b-1) and (3b-2).

(3b-1) Counting the occurrences in the observation set that match each value pattern.

(3b-2) Computing evaluation scores from the counts.

(3c) Transporting the computed scores back to CPU.

As written in Step (3b), we first count the number of occurrences in the observation data O that match each value pattern of events. Here, value patterns are defined on each

event as a set of values taken by the event and its parent events. For definition, see Fig. 1 again. The value patterns on event n_3 is the combination of values of n_3 and its parents n_1 and n_2. Since those three variables take binomial values (i.e., 0 or 1), we have 8 value patterns such as $(n_1, n_2, n_3) = (0,0,0), (0,0,1), \ldots, (1,1,1)$. For conciseness, we denote it by $n_1n_2n_3 = \{000, 001, \ldots, 111\}$. In general, the set of value patterns for event n_i is denoted by $V_i = p_1p_2 \ldots p_rn_i = \{0..00, 0..01, \cdots, 1..11\}$ where the parents of n_i in model m is $\mathrm{pa}(n_i) = \{p_1, p_2, \ldots, p_r\}$ and r is the number of parents of n_i.

Then, our task in step (3b) is to compute the number of occurrences $\mathcal{N}_{iv}$ in O that takes value pattern v, for every event $i \in N$ and value pattern $v \in V_i$. Although there are several information criterion such as AIC, BIC and MDL that are used as model scores in learning Bayesian Network structure, they are all computed from the counts of value patterns, as shown in Sec. 5.3.

The basic strategy for counting value patterns is to assign a SM to a single model m, and to use all SPs in the SM in parallel to count all value patterns for all events in m. By assigning a model m to a single SM, we compute the model score of m in the SM. Each SM in a GPU processes models one by one in parallel to compute model scores for all models in M. In the following subsections, we describe the algorithm to process m within a SM to show how to make efficient manipulation of data, especially to gain from the coalesce access of global memory.

5.2 Data Structure

In Step (3a), we transport the data required to compute model scores to the global memory. We declare three arrays that represent the following sets, respectively, in the global memory.

(i) The observation set O.

(ii) The model set M.

(iii) An array to record the computed model scores.

The pseudo code to define these data items is shown in the following.

```
u_int8_t observation[N][O];
boolean model[M][N][N];
float modelScore[M];
```

Here, variables `M`, `N`, `O` represents $|M|$, $|N|$, and $|O|$, respectively. We represent the observation set O as a two-dimensional array `observation` where the 1st dimension is events and the 2nd observations. We represent The model set M as a three-dimensional array `model` where 1st dimension is used for the index of models and 2nd and 3rd dimensions are used to describe each model. Each model is expressed as an adjacency matrix such that `model[m][i][j]` is true if there is a directed link from event i to j in m. The array `modelScore` is used to retain the value computed by the algorithm.

We use the Shared Memory to place the counters that retain the counts of every value patterns for all events in a model, as follows.

```
u_int16_t counter[Vi];
```

Here, `Vi` denotes the number of value patterns on event $i \in N$. Note that `Vi` is determined depending on the number of parents of i in the model m, and the number of their multinomial values. Thus, this array may exceed the capacity of the shared memory. (Consider that, if p_j may take a value from $w(p_j)$ distinct values, `Vi` $= w(i)w(p_1)w(p_2)\ldots w(p_r)$.) If the shared memory can afford to store this array in size, we execute fast counting algorithm that we call *case-1* shown in Sec. 5.4, and otherwise, we use alternative algorithm that we call *case-2* shown in Sec. 5.5.

5.3 Model Scores

Note that we count the number of observations in each case to compute evaluation scores. Although there are several information criterion such as AIC, BIC and MDL, used as model scores in learning Bayesian Network structure, they all are computed from the number of observations in each case. For instance, AIC is computed with the following formula:

$$AIC = -2l(\theta|O) + 2k, \tag{2}$$

where θ denotes the parameter set, $l(\theta|O)$ denotes the likelihood of θ under observation O, and the term k represents the number of parameters. Here, we further show that the function $l(\cdot)$ is represented by the following formula

$$l(\theta|O) \propto \sum_{i \in N} \sum_{v \in V_i} (\mathcal{N}_{iv}) \log \theta_{iv}, \tag{3}$$

where i denotes an event, v denotes a value pattern, $\mathcal{N}_{iv}$ denotes the number of occurrences in observations O that match i and v, and θ_{iv} is a parameter computable from $\mathcal{N}_{iv}$. This means that, by counting the number of observations for each value pattern, we can compute the evaluation score of the model m such as AIC. Note that other information criteria used in Bayesian Networks such as BIC, MDL, etc., also can be computed through counting the matching value patterns in the observation set O.

5.4 Computing Model Scores (Case 1)

If the size of the array `counter[Vi]` is within the capacity of shared memory, the procedure described here (i.e., *case-1*) is applied. Before computing the model score of the given model $m \in M$, the SM responsible to this task computes $\mathcal{N}_{iv}$ for all $i \in N$ and $v \in V_i$. In the procedure, the SM proceeds each node i sequentially. Thus, we now fix $i \in N$ and focus on computing $\mathcal{N}_{iv}$ for all $v \in V_i$.

Our strategy to do this is to process occurrences in O in parallel using SPs. Thus, we designed the procedure such that each thread reads a single occurrence of O and increments the corresponding value in `counter`. Namely, we have as large number of threads as the occurrences of O. By the scheduler of GPU that assigns threads to SPs, we can read occurrences from successive addresses of the global memory as shown in Fig. 6 (remember that the addresses of occurrences for $n_i, p_1, p_2, \ldots, p_r$ are successive in array `observation`, respectively), gaining from coalesce access.

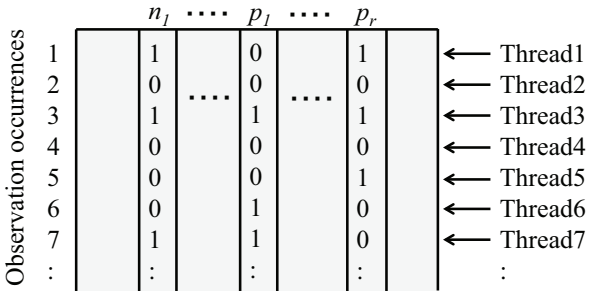


Figure 6: Counting Value Patterns (Case 1)

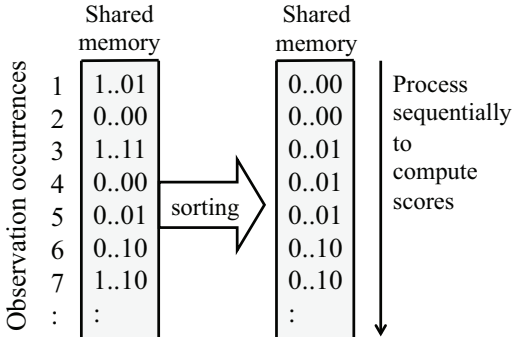


Figure 7: Counting Value Patterns (Case 2)

As the result of the above procedure executed for all events $i \in N$, we can obtain $\mathcal{N}_{iv}$ for all $i \in N$ and $v \in V_i$.

After computing $\mathcal{N}_{iv}$ for all $i \in N$ and $v \in V_i$, we compute the model score of m from them. This is simply done by computing the value according to formula (2). To compute it in parallel, we assign a SP for each $i \in N$ and sum up the model scores on the shared memory using a technique called *parallel reduction*. We finally store the computed score into the array `modelScore` in the global memory.

5.5 Computing Model Scores (Case 2)

If the size of the array `counter[Vi]` exceeds the capacity of the shared memory, we have to choose a less efficient algorithm. In the Alarm Network used in the evaluation of this paper, we can use the *case-1* algorithm only when the number of parents r is less than 7, where each event takes 4 distinct values and we have about 48KBytes shared memory.

In the *case-2* algorithm, instead of the array `counter[Vi]`, we define the array `patternValue[O]` as follows.

```
u_int16_t patternValue[O];
```

We simply use this array by storing pattern values of each occurrences of O. Retrieval of the pattern values from global memory can be done using coalesce access in the similar way to *case-1*. After retrieving the pattern values, we sort the values as shown in Fig. 7. Note that a GPU-specific sorting algorithm called *bitonic sort* can be used for high-throughput sorting. Then, we trace through the array sequentially in order to count each value pattern and sum up the model score. Although it takes a little longer than *case-1*, we can compute the model score in relatively short time.

Table 1: Evaluation Environment

OS		CentOS 5.0
CPU		Intel Core i7 4770k (3.50GHz)
Memory		32GBytes
GPU	Model	nVidia GeForce GTX TITAN Black (0.98GHz)
	# of SM	15
	# of SP per SM	192 (2880 SPs in total)
	Global Memory	6143 MBytes
	Shared Memory	49152 Bytes per SM
	GPU Library	CUDA 6.0
Compiler		g++ 4.1.2

6 EVALUATION

We evaluate the proposed method in terms of both running time and quality of the output model.

First, to clarify the performance of the proposed method to accelerate computational speed, we compare the running time of the proposed algorithm executed on GPU with its base PBIL-based algorithm executed on CPU. Note that their output is the same, only running time is different.

Second, as for the quality of the output model, we clarify how good is the BN models computed by PBIL-based algorithm. The quality of BN models can be measured by the model score such as AIC. By comparing the model scores obtained from major BN learning approaches, we show that PBIL-based algorithm is the most excellent among them.

By combining the results of running time and model quality, we would clarify that the proposed method is better than other major algorithms in the literature as it outputs higher-quality BN models within shorter running time.

6.1 Computational Time

We compare the running time of the proposed algorithm executed on GPU with its base algorithm PBIL-RS executed on CPU. We implemented both algorithms in C++ language with CUDA library for GPU processing. The execution environment is shown in Table. 1. We used Alarm Network [18] including 37 nodes as the base BN model; we generate an observation set including 1024 occurrences based on Alarm Network and learn BN models using each algorithm.

Figure 8 shows the computational time as generation proceeds. We see that the proposed algorithm that runs on GPU is about 14-times faster than PBIL-RS that runs on CPU only. In this figure, we also show a variant of the proposed algorithm seen as "case-2 only" that always runs *case-2* algorithm instead of *case-1* even if the number of parents is small. This variant takes about 1.6-times longer than the both-algorithm case, which indicates that *case-1* algorithm is considerably faster than *case-2*. To see the difference more precisely, we show the execution ratio of *case-1* and *case-2* in each generation in Fig. 9. Because mostly *case-1* algorithm is executed with 100% ratio, we can estimate that the *case-1* algorithm is about 1.6-times faster than *case-2*.

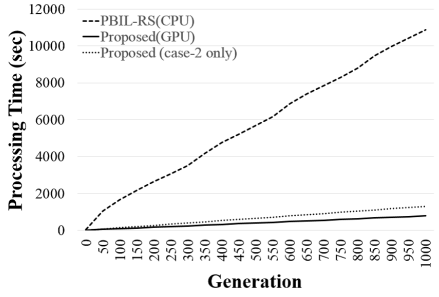


Figure 8: Execution Time (CPU vs. GPU)

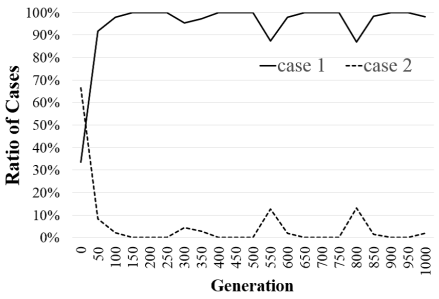


Figure 9: Ratio of Case 1 and 2

6.2 Model Quality

Next, we compare the quality of major BN-learning algorithms. As major algorithms to learn BN structure via optimization over model scores, we selected greedy hill climbing (GHC), simulated annealing (SA), PBIL, PBIL-RS, and K2GA. Note that GHC, SA, and PBIL are the type of algorithms that stop running when converged, and other two continue running until they are stopped by users.

Specific behaviors and parameter values are as follows:

GHC, in each iteration, always moves to the best model within the vicinity where we define that the two models are within their vicinity if one is generated by adding or deleting one edge from the other.

SA selects the next model in the vicinity and moves there if the score of the next model is better than the current one, and otherwise, stay at the current model with probability $e^{-\Delta/T}$ where Δ is the difference of the scores of the current and next models, and T is a parameter called temperature. T is initialized as 2500 and decreased at each iteration by multiplying $r = 0.99999$. As the vicinity to move in the next iteration, we apply the operation for every pair of nodes to add the edge (or delete if the edge already exists) in probability 0.004 (or 0.01 for deletion). These parameter values are determined to have the best performance through preliminary tests.

PBIL has several parameters as described in Sec. 3. In this evaluation, the number of individuals in each generation $|M|$ is 1000, the number of selected individuals k is 10, and learning ratio α is 0.1, all of which are determined to have the best performance through preliminary tests.

PBIL-RS is not specifically described in this paper due to space limitation, but all the definitions and parameter values are the same as [16].

K2GA applies GA to search the ordering space where K2 heuristic is used to convert from a node order to a graph structure. We use a typical crossover and mutation operator. Note that there are various crossover and mutation operators, but Ref. [3] reported that the performance is not so much different.

Table 2: Final Scores (Average)

Method	Final Score	Execution Time
PBIL	8777.7	3716 (Sec)
Simluated Annealing	9191.9	71590.7 (Sec)
Greedy Hill Climbing	8841.5	3142.1 (Sec)
PBIL-RS	8746.1	10881 (Sec)
K2GA	10011.9	302266.8 (Sec)

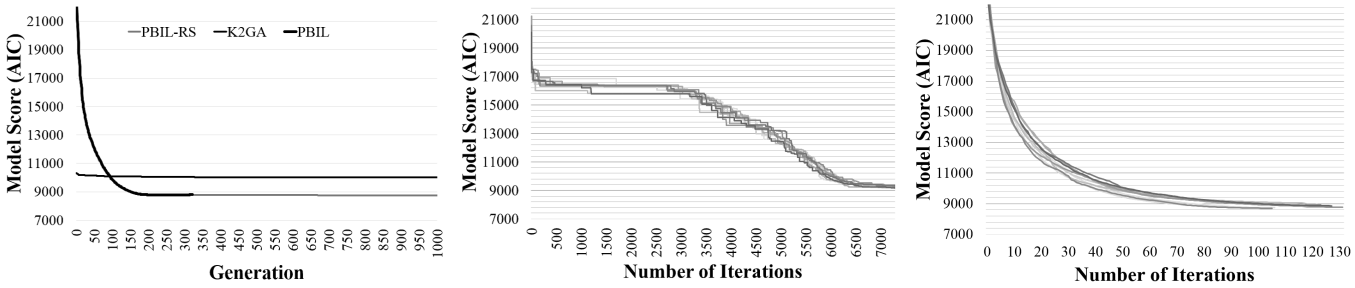
We show the comparison result of the GA-based algorithms K2GA, PBIL, and PBIL-RS in Fig. 10(a), which is the average of 10 repetitions. It is apparent that PBIL-based ones compute far better models than K2GA. Note that PBIL and PBIL-RS make almost the same curve, but PBIL stops around 320th generation due to convergence to a local minimum solution, whereas PBIL-RS continues running and find better models even after that.

As for non-GA algorithms, we show the score transition of 10 repetition of SA and GHC in Figs. 10(b) and 10(c), respectively. We see that all executions of each method make a similar curve until they converge and stop running. From those curves in Fig. 10(a)(b)(c), PBIL makes the steepest curve, meaning that the speed to approach better solutions is the highest among them.

Finally, in Table 2, we present the final scores and the running time of each algorithm. As for PBIL-RS and K2GA, we use the values of 1000th generation where improvement is scarcely seen in both algorithms, and for others we use the final values when they stop running. Among three algorithms that stop running, PBIL takes the best score, and PBIL-RS improves it by taking more execution time. Although GHC runs in a short time due to its simplicity, the score is lower than PBIL by more than 60. We also notice that the score of GHC would not improve by using more time, whereas the score of PBIL is continuously improved with time by using the technique of PBIL-RS. From above, we conclude that, in terms of scores, PBIL family would be the best algorithm among those compared.

7 CONCLUSION

We proposed a method to accelerate PBIL-based BN learning algorithms using GPU computation. We make the most of the coalesce access technique of GPU computation to reduce computation time using consumer level hardware. Through evaluation, we confirmed that the proposed method achieves about 14-times faster in running speed than the original PBIL-RS executed on only CPU. Also, we compared the quality of the computed BN models among several major approaches in the literature. As a result, we found that PBIL-based algorithms outperform other algorithms such as K2GA, greedy hill climbing, and simulated annealing. From above, we conclude that the proposed algorithm have an excellent performance in both speed and quality of solutions.



(a) PBIL(-RS) vs. K2GA (b) Simulated Annealing (SA) (c) Greedy Hill Climbing (GHC)

Figure 10: Score Transition

ACKNOWLEDGMENT

This work was partly supported by "the Program for Promotion of Stockbreeding" of JRA (Japan Racing Association).

REFERENCES

[1] D.M. Chickering, D. Heckerman, and C. Meek, "Large-Sample Learning of Bayesian Networks is NP-Hard," Journal of Machine Learning Research, Vol.5, pp.1287–1330 (2004).

[2] G.F. Cooper and E. Herskovits, "A Bayesian Method for the Induction of Probabilistic Networks from Data," Machine Learning, Vol.9, pp.309–347 (1992).

[3] P. Larrañaga, C.M.H. Kuijpers, R.H.Murga, and Y.Yurramendi, "Learning Bayesian Network Structures by Searching for the Best ORdering with Genetic Algorithms," IEEE Transactions on Systems, Man, and Cybernetics, Vol.26, No.4 (1996).

[4] I. Tsamardinos, L.E. Brown, and C.F. Aliferis, "The max-min hill-climbing Bayesian Network structure learning algorithm," Machine Learning, Vol.65, Issue.1, pp.31–78 (2006).

[5] A.S. Hesar, "Structure Learning of Bayesian Belief Networks Using Simulated Annealing Algorithm," Middle-east journal of Scientific Research, Vol.18, No.9, pp.1343–1348 (2013).

[6] P. Larrañaga, H. Karshenas, C. Bielza, and R. Santana, "A review on evolutionary algorithms in Bayesian Network learning and inference tasks," Information Sciences, Vol.233, No.1, pp.109-125 (2013).

[7] D.W. Kim, S. Ko, and B.Y. Kang, "Structure Learning of Bayesian Networks by Estimation of Distribution Algorithms with Transpose Mutation," Journal of Applied Research and Technology, Vol.11, pp.586–596 (2013).

[8] W.H. Hsu, H. Guo, B.B. Perry, and J.A. Stilson, "A Permutation Genetic Algorithm for Variable Ordering in Learning Bayesian Networks from Data," In Proc. GECCO'02, pp.383–390 (2002).

[9] E. Faulkner, "K2GA: Heuristically Guided Evolution of Bayesian Network Structures from Data," In Proc. IEEE CIDM'07, pp.18–25 (2007).

[10] A.P. Tonda, E. Lutton, R. Reuillon, G. Squillero, and P.H. Wuillemin, "Bayesian Network Structure Learning from Limited Datasets through Graph Evolution," In Proc. EuroGP'12, pp.254–265 (2012).

[11] O. Barriére, E. Lutton, and P.H. Wuillemin, "Bayesian Network Structure Learning Using Cooperative Coevolution," In Proc. GECCO'09, pp.755–762 (2009).

[12] S. Baluja, "Population-Based Incremental Learning: A method for Integrating Genetic Search Based Function Optimization and Competitive Learning," Technical Report CMU-CS-94-163 (1994).

[13] R. Blanco, I. Inza, and P. Larrañaga, "Learning Bayesian Networks in the Space of Structures by Estimation of Distribution Algorithms," International Journal of Intelligent Systems, Vol.18, pp.205–220 (2003).

[14] H. Handa, "Estimation of Distribution Algorithms with Mutation," Lecture Notes in Computer Science, Vol.3448, pp.112–121 (2005).

[15] S. Fukuda, Y. Yamanaka, and T. Yoshihiro, "A Probability-based Evolutionary Algorithm with Mutations to Learn Bayesian Networks," International Journal of Artificial Intelligence and Interactive Multimedia, Vol.3, No.1, pp.7–13 (2014).

[16] Yuma Yamanaka, Takatoshi Fujiki, Sho Fukuda, and Takuya Yoshihiro, "PBIL-RS: An Algorithm to Learn Bayesian Networks Based on Probability Vectors," In Proc. IWIN'2015 (2015).

[17] H. Akaike, "Information theory and an extension of the maximum likelihood principle," In Proc. ISIT'73, pp.267-281 (1973).

[18] 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," In Proc. AIME'89, Vol. 38, pp.247-256 (1989).

[19] O. Nikolova and S. Aluru, "Parallel Bayesian Network Structure Learning with Application to Gene Networks," In Proc. CS'12, Article No.63 (2012).

[20] M. D. Linderman, R. Bruggner, V. Athalye, T. H. Meng, and N. B. Asadi, "High-throughput Bayesian Network Learning using Heterogeneous Multicore Computers," In Proc. ICS'10, pp.95-104 (2010).

[21] Y. Wang, W. Qian, S. Zhang, X. Liang, and B. Yuan "A Learning Algorithm for Bayesian Networks and Its Efficient Implementation on GPUs," IEEE Transactions on Parallel & Distributed Systems, Vol.27, No.1 (2016).

(Received October 10, 2016)

Takashi Mori received the B.E. degree from Wakayama University in 2016. He is currently a Master-course student in Wakayama University. He is interested in data mining, machine learning, and GPU computing. He is a student member of IPSJ.

Yuma Yamanaka received his B.E. and M.E. degrees from Wakayama University in 2015 and 2017, respectively. He is currently with KYOCERA Communication Systems, Co., Ltd. He is interested in data mining, machine learning, and Bayesian modeling.

Takatoshi Fujiki received his B.E., M.E. and Ph.D. degrees from Wakayama University in 2010, 2012 and 2017, respectively. He is currently with HIBIYA Resource Planning, CO., LTD. He is interested in data mining, machine learning, and bioinformatics.

Takuya Yoshihiro received his B.E., M.I. and Ph.D. degrees from Kyoto University in 1998, 2000 and 2003, respectively. He was an assistant professor in Wakayama University from 2003 to 2009. He has been an associate professor in Wakayama University from 2009. He is currently interested in the graph theory, dis- tributed algorithms, computer networks, medial applications, and bioinformatics, and so on. He is a member of IEEE, IEICE, and IPSJ.