Parallel Simulated Annealing with a Greedy Algorithm for Bayesian Network Structure Learning

Sangmin Lee and Seoung Bum Kim

Abstract—We present a hybrid algorithm called parallel simulated annealing with a greedy algorithm (PSAGA) to learn Bayesian network structures. This work focuses on simulated annealing and its parallelization with memoization to accelerate the search process. At each step of the local search, a hybrid search method combining simulated annealing with a greedy algorithm was adopted. The proposed PSAGA aims to achieve both the efficiency of parallel search and the effectiveness of a more exhaustive search. The Bayesian Dirichlet equivalence metric was used to determine an optimal structure for PSAGA. The proposed PSAGA was evaluated on seven well-known Bayesian network benchmarks generated at random. We first conducted experiments to evaluate the computational time performance of the proposed parallel search. We then compared PSAGA with existing variants of simulated annealing-based algorithms to evaluate the quality of the learned structure. Overall, the experimental results demonstrate that the proposed PSAGA shows better performance than the alternatives in terms of computational time and accuracy.

Index Terms—Bayesian networks, structure learning, heuristic search algorithm, parallel structure learning, memoization, simulated annealing with a greedy algorithm

1 INTRODUCTION

Bayesian networks (BNs) are graphical models of probability over a set of variables. BNs have been widely used for probabilistic inference in many applications such as fault detection in manufacturing [1], medical decision support [2], [3], video classification [4], speech recognition [5], adaptive robot control [6], and gene expression in bioinformatics [7], [8]. The learning of BNs consists of two phases: structure learning and parameter learning. Structure learning specifies a set of relations between variables, and parameter learning quantifies the strengths of these relations. In structure learning, identifying the optimal solution with the constraint that the estimated structure must be a directed acyclic graph (DAG) is an NP-hard problem because of the intractable search space of network structures [9].

To address this problem, various greedy search approaches such as tabu search [10], K2 [11], particle swarm optimization [12], simulated annealing (SA) [13], ant colony optimization [14], hill climbing [15], and population-based algorithms, including the genetic algorithm [16] and water cycle algorithm [17], have been considered to significantly reduce the computational time of the search process. Although computationally efficient, these algorithms are prone to converge to local minima, especially in a large data set.

Another approach involves decreasing the search space. Heuristic pruning approaches based on the sparse candidate algorithm, which reduce the search space by preselecting a restricted number of parent candidates for each variable, achieve faster structure learning [18]. However, these are limited because the variables that are not selected as parent candidates are excluded from the search.

Further, there are sets of hybrid approaches combining greedy search with heuristic reduction in the search space [19], [20], [21]. These algorithms have been shown to be computationally less expensive than the sparse candidate algorithm; however, the internally utilized constraint-based methods are sensitive to the significance level used in conducting the hypothesis testing [11].

More recently, several approaches have been proposed for parallel structure learning to resolve the inefficiency of pure sequential algorithms for structure learning. Some studies have utilized distributed computing based on MapReduce to conduct parallel searches [22]. Others have proposed parallel methods for a single local search algorithm because of the complexity of parallelizing algorithms [23]. Existing sequential and parallel approaches for identifying a network structure tend to make the algorithms less exhaustive to achieve computational efficiency. However, the quality of their solutions can be significantly enhanced using a more exhaustive search, although this requires additional computational resources. Thus, it is necessary to improve both the search efficiency and effectiveness for structure learning.

In this paper, we propose a hybrid search algorithm that combines a more exhaustive greedy search and a...
more efficient parallel search. First, we use simulated annealing with a greedy algorithm (SAGA), a variant of SA, which is a widely used metaheuristic for the structure learning of BNs [24]. The proposed model is an extension of the existing SAGA, restarting from a randomly initialized structure for a more exhaustive search. Second, we employ a parallelization approach to simultaneously run multiple instances of SAGA. Further, to improve the parallel processing effect, we use shared-memory communications for search instances to avoid needless reoperations in the parallel search.

The main contributions of this study can be summarized as follows:

1. Regarding local search, we employ SAGA to improve the performance of the original SA by reducing the probability of convergence into local optimal solutions. It is a more exhaustive search method for structure learning because it intensively explores most of the search space. We revise SAGA to share intermediate subsolutions in synchronous parallel processing.

2. We propose parallel simulated annealing with a greedy algorithm (PSAGA), which is a parallel version of SAGA with a memoization technique. We conduct parallel local searches with independent multiple local search instances and share intermediate subsolutions between instances to improve the parallel processing effect. PSAGA is developed as a multithread-based optimized structure learning algorithm.

3. To demonstrate the effectiveness and efficiency of the proposed algorithm, we compare it with the existing variants of SA-based algorithms. The experimental results demonstrate that the proposed PSAGA produces better solutions than the alternatives.

The rest of the paper is organized as follows. Section 2 presents preliminaries and definitions. Section 3 describes the proposed PSAGA algorithm. Section 4 presents a simulation study to compare the performance of the proposed algorithm and existing SA-based algorithms under various scenarios. Section 5 includes concluding remarks.

2 PRELIMINARIES

Let $X$ be a set of random variables $X_i$ from data set $D$. A BN is represented by a pair $(S, \theta)$, where $S$ and $\theta$ indicate the DAG (the learned structure) and the conditional probability distributions over $X_i$, respectively. The number of possible relations is $n(n-1)/2$, where $n$ is the size of the dimension. Given $S$ and $\theta$, the joint probability can be represented as follows:

$$P(X_1, X_2, ..., X_n) = P(X_1)P(X_2|X_1)... P(X_n|X_1, X_2, ..., X_{n-1}) = \prod_{i=1}^{n} P(X_i|Pa_i),$$

(1)

where $Pa_i$ denotes the set of parents of $X_i$. The factorization follows the Markov property of BNs, which states that every $X_i$ depends directly only on $Pa_i$ [25]. For any relation, edge-connecting variables in $S$ can be added, reversed, or deleted in learning to obtain a better structure.

The possible changes in this procedure are $O(n^2)$. To reduce the learning time and space complexity, we can factorize a reduced conditional probability $Pa_i$ for each $X_i$ more efficiently by eliminating unnecessary probabilities. Here, we summarize the key definitions for efficient structure learning of BNs [26], [27]. Upper-case letters $(X,Y)$ denote variables, and lower-case letters $(x,y)$ denote their values. $Pa_i$ and $Ch_i$ indicate the set of parents of $X_i$ and set of children of $X_i$, respectively.

**Definition 1. Conditional independence.** $X_1$ and $X_2$ are conditionally independent given $X_3$, denoted as $X_1 \perp X_2 \mid X_3$ where $X_3 \subseteq \cup\{X_1, X_2\}$.

$$X_1 \text{ is conditionally independent of } X_2 \text{ given } X_3 \text{ if } P(X_1 = x, X_2 = y)P(X_3 = z) = P(X_1 = x|X_3 = z)P(X_2 = y|X_3 = z).$$

**Definition 2. Local Markov condition.** A variable in a BN is independent of its non-descendant variables given its parents.

The local Markov condition holds that a BN structure $G$ entails a set of conditional independence assumptions. It provides compact representation of the joint probability distribution $P$ in a BN. It represents a full joint probability as products of local conditional probabilities.

**Definition 3. Faithfulness condition.** $G$ and $P$ are faithful to each other if and only if conditional independencies that are true in $P$ are entailed by $G$.

$G$ is faithful to $P$ for a variable set $\cup$ if every conditional independence relation in $P$ is encoded by $G$ and the Markov condition. The faithfulness condition holds that $G$ represents all conditional independencies. It enables us to recover $G$ from $P$.

**Definition 4. Equivalence.** Two network structures, $G$ and $G'$, are equivalent if the set of distributions that can be represented by one of the DAGs is identical to the set of distributions that can be represented by the other.

The equivalence definition implies that two equivalents $G$ and $G'$ are statistically indistinguishable in BNs.

**Definition 5. $v$-structure.** Three variables $X_1$, $X_2$, and $X_3$ form a $v$-structure if $X_2$ has two incoming edges from $X_1$ and $X_3$, forming $X_1 \rightarrow X_2 \leftarrow X_3$ while $X_1$ is not adjacent to $X_3$.

Two equivalents $G$ and $G'$ share a set of $v$-structures as the same skeleton and partial structure.

**Definition 6. Markov blanket.** A Markov blanket of a target variable $X_T$ is the minimal set of variables conditioned on which all other variables are independent of $X_T$.

The Markov blanket (MB) is a minimal union of variables that consist of $X_T$’s direct parents, children, and its children’s other parents. It can be denoted as $MB(X_T) = Pa_T \cup Ch_T \cup \bigcup Y \not\subseteq Pa_T Pa_T$.

In this study, we used a score-based approach that identifies $S$ in such a way that the result is optimal for a scoring metric. We used the Bayesian Dirichlet equivalence (BDe) score [28] as the measure of fit for the equivalence class [29]. The BDe score is calculated as follows:
$P(S, D) = P(S)P(D | S) = P(S)\prod_{i=1}^{n} \prod_{j=1}^{\Gamma} \frac{\Gamma(N_{ij})}{\Gamma(N'_{ij} + \Gamma)} \prod_{k=1}^{r_i} \frac{\Gamma(N'_{ijk} + N_{ijk})}{\Gamma(N'_{ijk})},$

where $P(S)$ is the prior probability of $S$, $\Gamma$ denotes the gamma function, $q_i$ ($1 \leq i \leq n$) is the number of states of $P_{A_i}$, and $r_i$ is the cardinality of $X_i$. $N_{ijk}$ is the number of occurrences of the $k^{th}$ ($1 \leq k \leq r_i$) state of the $i^{th}$ variable with the $j^{th}$ ($1 \leq j \leq q_i$) state of its parents. $n$ is $\sum_{k=1}^{r_i} N_{ijk}$, and $N'$ denotes the parameter of the Dirichlet prior distribution and $N'_{ij}$ is $\sum_{k=1}^{r_i} N'_{ijk}$. Learning $S$ can be considered an optimization problem in which the metric BDe of $G$ should be maximized with a heuristic search algorithm.

The aim of structure learning from $D$ is to determine $S$. We use SA, which uses iterative improvements in the exploration to determine an approximate global optimum [30]. SA starts at an initial temperature $T_i$ and explores a new structure with any change in the given structure. SA accepts a new solution based on the temperature of the system with the following acceptance probability function:

$$P(S, S', T) = \left(1 + \exp \left(\frac{f(S') - f(S)}{T}\right)\right)^{-1},$$

where $P$ is the probability function for acceptance, $S$ and $S'$ indicate the current and new solutions, respectively, $T$ is the temperature, $f$ is the fitness function, and $f(S') - f(S)$ denotes the energy increment between the new and current solutions. In the search procedure, $T$ is consistently decreased to ensure convergence to an acceptable solution. SA convergence is proved in [31].

SAGA is a hybrid search method that combines SA with a greedy search. SAGA thoroughly identifies the optimal structure by exploring most of the search space by undergoing a two-stage search procedure: first, with SA and then with greedy search, SAGA guarantees near optimal solutions within a fixed time without reducing the search space. SAGA provides a more exhaustive search to improve the performance of SA.

However, it has two drawbacks for identifying the optimal structure, especially in large datasets: (1) its performance does not improve with the passage of experiment time because of the limited exploration performance for convergence [32]; (2) Because of SA’s recursive nature, it is difficult to make SA parallel. Further, in structure learning of BNs, high computational costs are required to calculate the fitness score for a solution, but in the fitness function of SA, $f$ is difficult to parallelize [33]. In addition, to improve the performance of an asynchronous parallel search, it should share subsolutions between search instances. The details of the proposed algorithm to address these issues are discussed in Section 3.

3 PARALLEL SIMULATED ANNEALING WITH A GREEDY ALGORITHM

The objective of this study is to provide a hybrid algorithm combining a more exhaustive search with a more efficient parallel search to improve the performance of structure learning. Fig. 1 presents an overview of PSAGA utilizing multithread computing.

To parallelize SAGA for structure learning, we use a multithread approach to intuitively accelerate the search operations. Each thread independently performs a SAGA instance based on a Markov chain. Using multiple Markov chains is the most straightforward approach for parallelizing SAGA [34]. This is because SA, the default algorithm for SAGA, is intrinsically sequential, and thus, it is difficult to parallelize without modifying its iterative operation [35].

Fig. 2 illustrates an example of the overall procedure of parallelized structure learning. $S'_j$ is the learned network structure found at the $j^{th}$ iteration of the $i^{th}$ thread. A single thread launches a local search $r$ times, and subsequently it restarts the greedy search algorithm $k$ times. When all threads finish, we select the best solution $S^*$ between solutions.
PSAGA shares subsolutions among the search instances discovered during the search process. This is because asynchronous parallel processing causes reoperations, which reduce the effectiveness of parallel processing. Hence, to address this problem, we use memoization for the operations incurring high computational cost. Memoization is an optimization technique used to accelerate the search by storing and reusing subsolutions.

In score-based structure learning, scoring metrics entail high computational cost. The BDe metric can be decomposed into subsolutions according to the MB, which is a useful concept for conditional independence. It is possible to identify the MB of a variable from a data set $D$ under the faithfulness assumption defined in Section 2. We limit the MB to the base boundary to determine subsolutions for calculating the cost. The BDe metric maintains score-equivalent conditions if Markov-equivalent structures have the same score. Here, memoization makes the parallelized search more time-efficient by storing the cost combination value $v_t$ for the MB of $X_i$ in the lookup table $Q$ and reusing it to avoid recalculation between search instances.

The pseudocode for PSAGA is provided in Algorithm 1. We first initialize global variables and instantiate multiple local search SAGA instances (lines 6–11). Multiple instances are executed asynchronously and share intermediate subsolutions through the lookup table (lines 12–15). To avoid latency in execution due to frequent creation and destruction of threads, we use a thread pool in line 14. The threads in the pool execute tasks from the task queue $T_Q$, and then place these tasks into a completed task queue $T_Q_{comp}$ after completion of the search instances. Finally, we identify the best solution at the end of the process.
We now explain the base local search algorithm in detail. The pseudocode for the algorithm is provided in Algorithm 2. The algorithm is a single-thread operation that iteratively explores the search space to identify a new structure with any change in the given structure. An annealing operation is performed \( R \) times (lines 14–25). In particular, we use the lookup table \( Q \) to evaluate a network structure. At the end of each iteration, we employ SAGA in such a way that a greedy operation is performed \( K \) times with a randomized initial structure to identify the better solution (lines 9–12 and 27–29). Finally, the SAGA algorithm returns the best solution \( S_{\text{best}}^{tld} \) to the P-SAGA algorithm.

We use big O to analyze the computational complexity of the proposed method. Given a set of random variables \( X_i \) of size \( n \), suppose that \( r \) candidates are selected in the MB of a target variable \( X_r \). The proposed algorithm checks \( r \) possible changes at every step. Thus, the search space complexity is \( O(2^r) \), where \( 2r \) candidate arcs should be examined. The total time of the proposed method is \( O(n \times K \times R \times 2^2) \), where \( K \) and \( R \) are the maximum number of restart and local search operations, respectively. The time complexity is restricted by \( r \). Regarding the space complexity, we can suppose that \( K \) and \( R \) are constant and the average number of \( r \) is much smaller than \( n \), for example \( r = \log(n) \). Hence, the space complexity is \( O(n \times 2^{2\log(n)}) \).

<table>
<thead>
<tr>
<th>Algorithm 2. Local Search Algorithm Using SAGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: procedure LocalSearch ((tld, S_{\text{best}}^{tld}, Q))</td>
</tr>
<tr>
<td>2: // ( tld ): the instance id</td>
</tr>
<tr>
<td>3: // ( K, R ): the maximum number of the restart and local search operations, respectively</td>
</tr>
<tr>
<td>4: // ( T_{\text{min}} ): the given temperature threshold</td>
</tr>
<tr>
<td>5: // ( S_{\text{best}}^{tld} ): the solution of the ( tld ) instance</td>
</tr>
<tr>
<td>6: // ( Q ): the lookup table to share intermediate subsolutions</td>
</tr>
<tr>
<td>7: // ( f(S, Q) ): the function to estimate the BDe metric of ( S ) with the lookup table ( Q )</td>
</tr>
<tr>
<td>8:</td>
</tr>
<tr>
<td>9: ▶ Initialize</td>
</tr>
<tr>
<td>10: Instance_id ← ( tld, S_{\text{best}}^{tld} ) ← ϕ</td>
</tr>
<tr>
<td>11: for ( k = 1 ) to ( K ) do // perform the restarts of the search process ( K ) times</td>
</tr>
<tr>
<td>12: ( T ← T^1, S ← (\text{isNull}(S)) ? S^1 : S // initialize temperature and structure</td>
</tr>
<tr>
<td>13: do</td>
</tr>
<tr>
<td>14: ▶ Explore a new structure with any change of the given structure</td>
</tr>
<tr>
<td>15: for ( r = 1 ) to ( R ) do // consider ( R ) trials</td>
</tr>
<tr>
<td>16: ( S^r ← \text{computeNeighbour}(S) // ) pick a random neighbour structure</td>
</tr>
<tr>
<td>17: ( T ← \text{computeTemperature}(S^r) ) // calculate the temperature</td>
</tr>
<tr>
<td>18: ( \Delta E = f(S^r, Q) - f(S, Q) / ) calculate the energy increment</td>
</tr>
<tr>
<td>19: ( P = (1 + \exp(\Delta E/T))^{-1} / ) calculate the probability for acceptance</td>
</tr>
<tr>
<td>20: if ( \Delta E &lt; 0 ) or ( P &gt; \text{random(0,1)} )</td>
</tr>
<tr>
<td>21: ( S ← S^r // ) accept the ( r )th trial</td>
</tr>
<tr>
<td>22: ▶ Identify the better solution</td>
</tr>
<tr>
<td>23: if ( f(S, Q) ≥ f(S_{\text{best}}^{tld}, Q) )</td>
</tr>
<tr>
<td>24: ( S_{\text{best}}^{tld} ← S // ) update the best structure</td>
</tr>
<tr>
<td>25: end for</td>
</tr>
<tr>
<td>26: while((T &gt; T_{\text{min}}))</td>
</tr>
<tr>
<td>27: ▶ Extract a randomly initialized structure for the restart phase</td>
</tr>
<tr>
<td>28: ( S ← \text{initializeSolution}(S_{\text{best}}^{tld}) // ) randomly generate an initial solution from ( S_{\text{best}}^{tld} )</td>
</tr>
<tr>
<td>29: end for</td>
</tr>
<tr>
<td>30: return ( S_{\text{best}}^{tld} )</td>
</tr>
<tr>
<td>31: end procedure</td>
</tr>
</tbody>
</table>

### 4 Simulation

We evaluated the proposed algorithm on well-known benchmark data sets from the BN repository [36]: Child, Insurance, Mildew, Alarm, Hailfinder, Pigs, and Link. Table 1 summarizes the characteristics of the true networks for benchmark data sets.

Simulations were run on a desktop computer with Windows 10, an Intel® i7-6700 (3.40 GHz) processor with four cores and eight threads, and 32 GB of RAM. We used the Java SE Development Kit 8u121 to develop all test programs and implemented multithreading for parallel processing. Parallel execution can be obtained by specifying the number of threads. Moreover, we can manage threads...
by waiting and notifying signals for interthread communications. For the implementation of the memoization mechanism, we used a synchronized block to prevent concurrent destructive access to the shared memory.

### TABLE 1
**BENCHMARK DATA SETS OF BAYESIAN NETWORKS**

*The true networks of all seven data sets are known, and they are publicly available (http://www.bnlearn.com/bnrepository).*

<table>
<thead>
<tr>
<th>Benchmark data sets D</th>
<th>Nodes N</th>
<th>Edges E</th>
<th>Parameters θ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child</td>
<td>20</td>
<td>25</td>
<td>230</td>
</tr>
<tr>
<td>Insurance</td>
<td>27</td>
<td>52</td>
<td>984</td>
</tr>
<tr>
<td>Mildew</td>
<td>35</td>
<td>46</td>
<td>540,150</td>
</tr>
<tr>
<td>Alarm</td>
<td>37</td>
<td>46</td>
<td>509</td>
</tr>
<tr>
<td>Hailfinder</td>
<td>56</td>
<td>66</td>
<td>2,656</td>
</tr>
<tr>
<td>Pigs</td>
<td>441</td>
<td>592</td>
<td>5,618</td>
</tr>
<tr>
<td>Link</td>
<td>724</td>
<td>1,125</td>
<td>14,211</td>
</tr>
</tbody>
</table>

#### 4.1 Impact of Parallel Search with Memoization on Computational Time Performance and Search-Space Size

To evaluate the computational time performance of the proposed parallel search, we conducted experiments on the benchmark data sets by increasing the number of launched threads from one to 15. Note that one thread case is equivalent to a sequential search, which is the baseline. For each data set, we randomly generated 50,000 and 100,000 samples.

The average computational time with a confidence interval of 95% was obtained by conducting 10 replications of each data set. The computational time was calculated based on the elapsed time of the local search process, which includes the time spent managing threads and synchronizing the shared memory. The speed-up factor was calculated by dividing the multithreaded processing time by the sequential processing time (baseline).

![Average run times with confidence interval (95%) and average speedup factors by increasing the number of threads for Child data set with 50,000 and 100,000 samples](image)
Fig. 4. Average run times with confidence interval (95%) and average speedup factors by increasing the number of threads for Insurance data set with 50,000 and 100,000 samples.

Fig. 5. Average run times with confidence interval (95%) and average speedup factors by increasing the number of threads for Mildew data set with 50,000 and 100,000 samples.

Fig. 6. Average run times with confidence interval (95%) and average speedup factors by increasing the number of threads for Alarm data set with 50,000 and 100,000 samples.
Figs. 3–9 show the average run time (left y-axis) and speed-up factor (right y-axis) for the benchmark data sets. The left and right figures present the simulation results using 50,000 and 100,000 samples, respectively. Figs. 3–9 clearly show that the average run time decreased for all seven data sets with increasing number of threads. Further, the results of the average speed-up factor show that the proposed parallel search could achieve four- or five-times faster processing speed by using a sufficiently large number of threads. It is worth noting that the performance was not significantly improved when the number of threads used was more than 10. These results demonstrate that the proposed parallel approach significantly improves computational time performance over the sequential method.

TABLE 2

| SIZE OF THE SEARCHED SPACE TO EXAMINE PARALLEL SEARCH WITH MEMOIZATION |
|---|---|---|
| K | No memoization | Separate memoization | Shared memoization |
|---|---|---|

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2899096, IEEE Transactions on Knowledge and Data Engineering
Table 2 presents the experimental results on the size of the searched space to examine the usefulness of the parallel search with memoization. The size of the searched space, which is the number of the MB, is important because the MB and its score can be stored in the lookup table for the memoization technique and reused to avoid recalculation subsolutions. Three experimental settings were used for comparison. The first was based on PSAGA without memoization, which is a pure asynchronous parallel version. The second was based on PSAGA with non-shared memoization (only to avoid repeating works on an instance). The third setting was based on the parallel search with shared memoization. Note that the third setting reflects the method proposed in this study. We set the number of launched threads to 10 based on the previous experiments. We varied the maximum number of restart operations $K$ from 50 to 10,000 with the timeout termination condition of $t_{max} = 15$ minutes for SA because of time efficiency. In addition, we used an alarm data set with 5,000 samples. The average with a confidence interval of 95% was obtained by conducting 10 replications of each setting.

As can be observed in Table 2, the parallel search with shared memoization outperformed the other two cases in terms of the searched space. Moreover, its performance significantly exceeded that of the others when the restart and local search operations $K$ and $R$ increased, respectively. It is noteworthy that the larger search space can lead to better solutions.

### 4.2 Evaluation of Structural Accuracy

With regard to the experiments to compare performance accuracy, the proposed method PSAGA was compared to SA, simulated annealing (SR), SAGA, and parallel SR (PSR) algorithms. We used six different sample sizes for training (500, 700, 1,000, 3,000, 5,000, and 10,000) to examine the impact of sample size on performance. For each sample size, we conducted 10 replications with different random seeds to draw statistically meaningful conclusions.

The proposed and comparative methods have six key parameters: start temperature $T_1$, temperature threshold $T_{min}$, fixed decay rate $\alpha$ ($0 < \alpha < 1$) with the exponential temperature schedule $T_i = \alpha \cdot T_{i-1}$, maximum number of the restart and local search operations $K$ and $R$, respectively, and number of multiple search instances $T$. Kirkpatrick et al. [34] showed that the performance of SA highly depends on its optimization parameters. Burke et al. [37] proposed a guideline to determine the SA parameters in practice. We used a grid search to determine the best parameter values of $T_1$, $T_{min}$, and $\alpha$ for each method and each data set. That is, we varied the different values of $T_1$, $T_{min}$, and $\alpha$ and found the one that maximized the metric BDe of the BN structure $G$. As a result, we noticed that these parameters did not significantly vary for the different techniques. Because these parameters determine a base search performance of SA to identify an approximate global optimum, we set the same values for these parameters in all methods. In addition, we conducted a sensitivity analysis to identify the influence of $K$, $R$, and $T$, as described in Section 4.1. Regarding $K$ and $R$, we used sufficiently large values for each data set to ensure a fair comparison because they determine the termination condition for convergence. We set $T$ to 10 based on the experimental results in Figs. 3–9.

Table 3 lists the parameter values used in our simulation. We provided the same time-to-run to ensure fair comparisons. Regarding SR and PSR, we set the removal rate $\rho$ of the reannealing technique to randomly extract the sub-structure from the learned structure. As for the parallel search, PSR and PSAGA used the memoization technique to accelerate their search process.

<table>
<thead>
<tr>
<th>TABLE 3</th>
<th>PARAMETER SETTINGS FOR THE PROPOSED AND COMPARATIVE METHODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark</td>
<td>$T_1$</td>
</tr>
<tr>
<td>Child</td>
<td>2,000</td>
</tr>
<tr>
<td>Insurance</td>
<td>2,500</td>
</tr>
<tr>
<td>Mildew</td>
<td>4,000</td>
</tr>
<tr>
<td>Alarm</td>
<td></td>
</tr>
<tr>
<td>Hailfinder</td>
<td></td>
</tr>
<tr>
<td>Pigs</td>
<td></td>
</tr>
<tr>
<td>Link</td>
<td></td>
</tr>
</tbody>
</table>

To examine the performance accuracy, we compared the learned network structures with true networks in terms of the following measures: precision, recall, Euclidean distance, BDe score, global structure, and structural hamming distance. The precision is the number of true-positive edges divided by the number of true edges. The recall is the number of true-positive edges in the output divided by the number of edges in the output. The Euclidean distance, which is a balanced measure of the difference from a true network (perfect precision and recall), is calculated as $\sqrt{(1 - \text{precision})^2 + (1 - \text{recall})^2}$, as proposed by Pena et al.

---

The global structure measure $\mathcal{G}$ [38] describes the capability of an algorithm to learn the true network, including the correct orientations of edges as follows:

$$\mathcal{G} = \frac{\sum \text{Well oriented Arcs}}{\sum \text{Well oriented Arcs} + \sum \text{Wrong oriented Arcs}} + \sum \text{Added Arc} + \sum \text{Missing Arc}.$$  

(4)

The structural hamming distance $\mathcal{H}$ is the sum of the additional edges, missing edges, and incorrect edge directions [19], [40].

Fig. 10. Average precision for simulations with different sample sizes (higher is better)

![Fig. 10. Average precision for simulations with different sample sizes (higher is better)](image)

Fig. 11. Average recall for simulations with different sample sizes (higher is better)

![Fig. 11. Average recall for simulations with different sample sizes (higher is better)](image)

Fig. 12. Average Euclidean distance for simulations with different sample sizes (lower is better)

![Fig. 12. Average Euclidean distance for simulations with different sample sizes (lower is better)](image)

Figs. 10–12 present mean values aggregated over the seven experimental datasets. Figs. 10 and 11 show that in most cases, the proposed PSAGA yielded the best results in terms of both average precision and average recall. With a sample size of 10,000, the average precision and recall gaps of PSAGA compared with those of SA were approximately 17.51% and 6.03%, respectively. In terms of Euclidean distance, the solutions derived from PSAGA were lower than those of comparative algorithms with different sample sizes, as shown in Fig. 12. Comparing the SA and PSAGA solutions, the improvement rates in the Euclidean distance with the different sample sizes were of 2.34%, 3.93%, 4.67%, 2.4%, 4.3%, and 4.69%. Moreover, we observed that PSAGA consistently improved in the three measures considered in this experiment.

Fig. 13. Improvement rate with confidence interval (95%) in BDe metric $M(\text{BDe}_{\text{type}}/\text{BDe}_{\text{SA}})$ with different sample sizes (lower is better)

![Fig. 13. Improvement rate with confidence interval (95%) in BDe metric $M(\text{BDe}_{\text{type}}/\text{BDe}_{\text{SA}})$ with different sample sizes (lower is better)](image)

Fig. 14. Improvement rate with confidence interval (95%) in global structure $M(\mathcal{G}_{\text{type}}/\mathcal{G}_{\text{SA}})$ with different sample sizes (higher is better)

![Fig. 14. Improvement rate with confidence interval (95%) in global structure $M(\mathcal{G}_{\text{type}}/\mathcal{G}_{\text{SA}})$ with different sample sizes (higher is better)](image)

Fig. 15. Improvement rate with confidence interval (95%) in hamming distance $M(\mathcal{H}_{\text{type}}/\mathcal{H}_{\text{SA}})$ with different sample sizes (lower is better)

![Fig. 15. Improvement rate with confidence interval (95%) in hamming distance $M(\mathcal{H}_{\text{type}}/\mathcal{H}_{\text{SA}})$ with different sample sizes (lower is better)](image)

Figs. 13–15 also show the learning performance of the algorithms with similarity to the true network. Regarding the improvement rate compared with the solutions of SA, SR, and SAGA, the proposed PSAGA identified better solutions when the size of the data set increased (Fig. 13). Moreover, PSAGA outperformed the alternatives for all sizes of data sets on the BDe scoring metric. As shown in Figs. 14 and 15, the resulting BNs with PSAGA for all data sets more closely resembled the true network than the alternatives. The higher the value of the global structure measure was, the greater the resemblance. The lower the value of the structural hamming distance measure was, the lesser the possibility of incorrect edges.
5 Conclusion

In this study, we proposed a parallel hybrid search algorithm PSAGA for structure learning of BNs. The proposed PSAGA promotes the accuracy of the network structure by combining a more exhaustive greedy search and a more efficient parallel search. Its improved performance is attributed to its capacity to overcome the drawbacks of existing variant algorithms for structure learning with extremely long runtime in large data sets. The experimental results on well-known benchmark data sets confirmed that the proposed PSAGA showed superior performance by improving accuracy. There are several directions for extending the present study. One future work will focus on the use of graphics processing units to accelerate the performance of PSAGA. Furthermore, more effort is required to obtain better search efficiency and to improve the quality of solutions for structure learning.

Acknowledgment

This research was supported by Samsung Electronics, Co., Ltd., Brain Korea PLUS, Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Science, ICT, and Future Planning (NRF-2016R1A2B1008994), and the Ministry of Trade, Industry & Energy under the Industrial Technology Innovation Program (R1623371).

References

[36] Bnlearn, Bayesian Network Repository, an R package for Bayesian network learning and inference (http://www.bnlearn.com/bnrepository/) (access date 24 May 2018)

Sangmin Lee received an M.S. degree in Industrial and Systems Engineering in 2007 from the Korea Advanced Institute of Science & Technology. He has worked as a software engineer at Samsung Electronics. He is currently a Ph.D. candidate in the Department of Industrial Management Engineering, Korea University, Seoul, Korea. His research interests include Bayesian networks, metaheuristic algorithms, and predictive analytics for manufacturing.

Seoung Bum Kim is a professor in the School of Industrial Management Engineering, Korea University, since 2009. He received an M.S. degree in Industrial and Systems Engineering in 2001, an M.S. degree in Statistics in 2004, and a Ph.D. degree in Industrial and Systems Engineering in 2005 from the Georgia Institute of Technology. His research interests include statistical and machine learning modeling for various problems appearing in engineering and science.