Simulator Prototyping Through Graphical Dependency Modeling

Kumar Dheenadayalan, V.N. Muralidhara, and G. Srinivasaraghavan

International Institute of Information Technology, Bangalore, India
d.kumar@iiitb.org, {murali,gsr}@iiitb.ac.in

Abstract. Given a complex system, simulating the behavior of the system under various conditions and inputs is a common requirement in different domains. Designing a simulator by non-domain expert is not an easy task. However, the vast amount of performance related parameters that can be monitored on any physical system can help in probabilistic modeling of the system. Accurate modeling of a complex system requires the identification of dependency among the individual components. These dependencies can be deduced by probabilistic analysis of the performance parameters generated by the physical system. We propose constrained Hill Climbing for structure learning of a Bayesian Network to learn the dependency among internal components of the system. Dependency graph, thus generated will act as a basis for developing a prototype simulator. Multivariate Adaptive Regression Splines are compared with the proposed constrained hill climbing to deduce the mathematical relation between the interdependent components. Two prototype simulators are designed, one for a complex large-scale storage system and another for a group of computer servers. Behavior of both the physical systems is tested on workload traces. The normalized mean absolute error for simulation of storage systems and server was 4.6% & 3.75% respectively. Results indicate that the proposed simulation prototyping can be a useful and unique way of understanding complex systems without the help of a domain expert.

Keywords: Probabilistic graphical model · Bayesian network · Regression · Multivariate adaptive regression splines · Simulator

1 Introduction

Simulation and modeling of a physical system is a discipline in itself and widely used to analyze the behavior of a physical system. Modeling defines the interaction between components of a system and simulation provides insights on the implications of interaction between the components. Models help in simplifying the reality and represent interactions into a form that users can comprehend. Research in simulation and modeling has spread across various domains and the methodologies can vary in each research domain. There is no standard mechanism for users to extract a model and build a quick prototype simulator for a

physical system. Consider a novice user, trying to understand the behavior of a complex physical system for different inputs and configurations. Running a physical system for different type of inputs and combination of configurations can be taxing on the physical system and a costly, tedious and time consuming process for the users. Users of a physical system typically depend on a simulator to test its performance. In certain cases where a simulator does not exist, choices are limited and users are forced to go through multiple runs on the system. To facilitate the understanding of a system and its behavior in various scenarios, learning algorithms can be effectively used to build a quick prototype simulator.

Let us assume that the physical system, $\mathcal{S}$ has a management/monitoring interface that monitors all the components within a system. For example, a computer system, where performances of various components like hard disk, network, CPU etc. can be monitored (disk_read_rate, disk_latency, network_packets_sent etc.). Collecting performance parameters $\mathcal{P}$, of all the components over a period can act as a valuable dataset $\mathcal{D}$ for non-domain experts to extract a high level model of $\mathcal{S}$. In this paper we propose a generic scheme to develop a prototype simulator. The proposed scheme is evaluated on two different datasets, i.e., performance parameters for two different physical systems. Firstly, a group of computer servers are considered for modeling and simulation. Secondly, a multi-component storage system (NetApp filer) is used. Both the physical systems have their own way of extracting performance parameters relevant to the embedded components. Number of hardware, software and virtual components in the systems enables complex interactions. which need to be decoded for developing a simulator. The performance parameters of various components are collected over multiple days to build $\mathcal{D}$. Process of extracting a model from $\mathcal{D}$ is called the *modeling phase*. It builds the dependency between various performance parameters of $\mathcal{S}$. We propose the learning of dependency structure of a Bayesian Network using a score based algorithms, constrained by parameters of input traces. Constraining the structure based on input traces helps in optimizing the same for use in simulation. Constrained Hill Climbing is used as the heuristics to learn the Bayesian Network $\mathcal{B}$ to produce a Directed Acyclic Graph (DAG).

Acyclic property of the DAG helps in deducing the values of all the performance parameters through probabilistic analysis or through regression schemes during the *Simulation phase*. Topological ordering of nodes are extracted to identify the best path for simulation. Using the topological order of nodes, both Bayesian framework and a Multivariate Adaptive Regression Splines (MARS) based multi-model regression scheme are developed and their simulating capabilities are compared. It must be noted that all the performance parameters are treated as response variable with very few predictor variables (workload parameters) making this a complex error prone task. We propose Normalized Mean Absolute Error (NMAE) as a measure to estimate regression error. This is because, the systems being modeled and simulated tend to have parameters with large range of values. The NMAE rates for the two case studies discussed turn out to be 6% and 4.9%.

In the next section we discuss the past literature followed by an overview and analysis of the proposed scheme for prototype development in Section 3. Evaluation of the proposed method along with implementation details are discussed in Section 4 followed by the conclusion in Section 5.

2 Literature Survey

Modeling and Simulation exists in almost all the fields like aeronautics, medicine, physics, transportation, environmental studies, astronautics, climatology to name a few [7,16,17,19]. Past literature on applying learning algorithms for simulation appear in two diverse fields. One of the literature talks about applying clustering algorithm to guide architecture simulation [8]. Structured sequences of a few recurring behaviors are identified during program execution using clustering algorithms. These are uniquely sampled to create a complete representation of program's execution. This enables evaluation of computer architecture without having to evaluate the entire program. Another application of using learning algorithm for simulation is in the field of manufacturing [21]. A generic simulation system is developed for the manufacturing industry using Neural network based adaptive simulation system to independently model the interdependencies of production processes. Modeling and Simulation phases proposed in our work are also generic and simulation phase can be adapted to any domain.

2.1 Bayesian Network

A Bayesian network is a graphical model representing probabilistic relationships among a set of variables [10]. Bayesian Networks are used to model uncertain domain knowledge and find probable configurations of variables [11]. Features of the dataset are a set of random variables $X = \{X_1, ...X_n\}$, represented as nodes in a network. Learning the structure identifies the dependency among random variables and this can handle missing data in a probabilistic sense. Direct dependencies between variables is represented by a set of directed arcs between pairs of nodes, $\{X_i \rightarrow X_j\}$. Learning the structure of Bayesian Network from $\mathcal{D}$ is a way of modeling the domain that might have many inherent uncertainties. Identifying the best Bayesian Network structure from the Network space, $\mathcal{B} \in \mathcal{B}_n$, is quantified by a score function. Scoring function is a measure of goodness-of-fit of the network [11]. Best network structure is the one, which maximizes the scoring function for a given $\mathcal{D}$.

Application of Bayesian Network was initially limited to the field of medicine [4,9]. Later, Bayesian Networks have been used to model diverse systems and environments and the areas of application have continuously been growing. Diagnosis of physical systems like printers [18] and taking actions based on the diagnosis [1] have also been achieved using Bayesian Network. The level of uncertainties in the ecosystems and other components within the earths' environment is high. The risks of oil spill and the environmental impacts of human activities is modeled using Bayesian Network [12]. Applications as diverse as Software Fault

Prediction to dataset compression has been achieved through this learning algorithm [5]. Bayesian Network learns the dependencies of features from the data that makes it one of the most popular tool for modeling a physical environment.

3 Simulation Scheme

Simulation scheme proposed in this paper broadly consists of two steps.

- Construct the Constrained Bayesian Network from data and fit a Conditional Probability Table[CPT].
- Use the dependencies to build a multi-model regression.

A good model represents the dynamics of a system and allows exploring its behavior systematically. The key for simulation is the amount of information that is available to model a real world physical system. The information extracted from a physical system that is available in the form of a dataset is described here. A single data instance of performance parameters $\mathcal{D}_t$ consists of all the monitorable components and related parameters. If $\mathcal{C}_i$ is the i^{th} component of $\mathcal{S}$, then $\mathcal{P}_{\mathcal{C}_i}$ is a row vector of performance parameters related to the component $\mathcal{C}_i$.

$$\mathcal{D}_t = (\mathcal{P}_{t_{\mathcal{C}_1}}, . \ldots\ldots, \mathcal{P}_{t_{\mathcal{C}_m}})$$

Let $\mathcal{T}$ be the features of workload trace, i.e. the trace of inputs to the system. *Spec* is a vector of specifications of the physical system being modeled. The system to be simulated will be monitored to record two sets of information. $\mathcal{T}_t$ is the input trace recorded on the system at time t and the response of the system to input trace are recorded in $\mathcal{D}_t$. The workload traces here are the actual inputs to the live system. We build the dataset for our simulation as a combination of workload traces, system specifications and response of the system.

$$\mathcal{D} = \begin{bmatrix} \mathcal{T}_1 & Spec & \mathcal{P}_{1_{\mathcal{C}_1}} & \ldots & \mathcal{P}_{t_{\mathcal{C}_m}} \\ \mathcal{T}_2 & Spec & \mathcal{P}_{2_{\mathcal{C}_1}} & \ldots & \mathcal{P}_{t_{\mathcal{C}_m}} \\ \vdots & \vdots & & \ddots & \vdots \\ \mathcal{T}_n & Spec & \mathcal{P}_{n_{\mathcal{C}_1}} & \ldots & \mathcal{P}_{t_{\mathcal{C}_m}} \end{bmatrix}$$

System specifications are also recorded, as certain environments might contain multiple physical systems with various specifications. Model should be robust enough to simulate the inputs on physical systems with various specifications. The need for recording and combining the workload traces with performance parameters is explained in the next section.

3.1 Dependency Modeling

In this section, we describe Structure learning of a Bayesian Network constrained by traces for a given $\mathcal{D}$. Learning the most probable posterior probability of a Bayesian Network from data is known to be NP-hard [3]. Optimization strategy

is applied through search-and-score approach that consists of a scoring function and a search strategy. Search Strategy involves heuristics to efficiently search the network structure space and identify the best network for the given data. The notion of best network is measured by the scoring function, which is a measure of goodness-of-fit for the data. One such scoring function is the Bayesian Information Criterion (BIC) defined as

$$\mathcal{BIC}score(\mathcal{B}) = loglikelihood(\mathcal{B}) - (log(n)/2) \times log(N)$$

n ← Number of rows in $\mathcal{D}$
N ← Number of free parameters
The number of free parameters is defined as the sum of logically independent parameters of each node given its parents [2].

Hill climbing is a popular heuristic to search the network space locally until a local maxima is found [20]. Strategies like random perturbation of the network structure for a fixed set of iterations are used to prevent getting stuck at local maxima. Behavior of a physical system is a consequence of the input provided to the system. Hence, we want to induce this logic as a constraint in our structure learning algorithm. The reason for combining the workload traces with performance parameters was to make sure the structure is built with the above logic. All variables from $\mathcal{T}$ and *Spec* will be applied as constraints forcing them to be treated as a root node in the Bayesian Network. This constraint is imposed in the Hill climbing algorithm described in Algorithm 1.

Consider an example with two features related to input traces (T_1, T_2), one feature related to specification (S_1) and 3 performance parameters (P_1, P_2, P_3). The effect of constructing constrained and unconstrained networks using hill climbing is shown in Figure 1. An Unconstrained network has unwanted dependencies deriving input trace from parameters $(\{P_3 \rightarrow T_2\}, \{P_1 \rightarrow T_2\})$ whereas constrained network fixes these situations. Inputs affect the performance parameters and not the opposite. Although input traces can sometime be derived from performance parameters, they are not of much use for the simulation task considered here. As a result of these constraints, this workload traces to be simulated along with the *Spec* would be enough to simulate an entire system.

The hill climbing algorithm is initialized with an empty structure to induce sparsity. Similar to standard hill climbing, addition, deletion and reversal of node operations are applied to the graph $\mathcal{G}$. During this exploratory process, constraints are evaluated to ensure that addition of a node or reversal of an edge does not create a dependency from a performance parameter to a workload parameter. As a result, parameters of workload traces will most likely end up having no parents. $updateNetwork()$ function updates the network $\mathcal{B}$ every time a better structure is identified. Once the model of the physical system is available, we process the network by utilizing the identified dependencies in the simulation process.

Algorithm 1. Learn Bayesian Network Structure

procedure LEARNSTRUCTURE($\mathcal{D}, \mathcal{G}, \mathcal{B}, trainIndex, constraintFeatures$)
  $trainData \leftarrow \mathcal{D}[trainIndex,]$
  $\mathcal{BIC}score \leftarrow -\infty$
  while $\mathcal{BIC}score \geq max\mathcal{BIC}score$ **do**
    for all $\{node_i, node_j\} \in features(\mathcal{D})$ **do**
      if $acyclic(\mathcal{G} \cup \{node_i \rightarrow node_j\})\&\&\{node_j \cap constraintFeatures\} == \emptyset$ **then**
        $\mathcal{G} \leftarrow \{\mathcal{G} \cup \{node_i \rightarrow node_j\}\}$
        $update(\mathcal{D}, \mathcal{G}, \mathcal{B}, trainData)$
      end if
      if $acyclic(\mathcal{G} - \{node_i \rightarrow node_j\})$ **then**
        $\mathcal{G} \leftarrow \{\mathcal{G} - \{node_i \rightarrow node_j\}\}$
        $update(\mathcal{D}, \mathcal{G}, \mathcal{B}, trainData)$
      end if
      if $acyclic(\{\mathcal{G} - \{node_i \rightarrow node_j\}\} \cup \{node_j \rightarrow node_i\})\&\&\{node_i \cap constraintFeatures\} == \emptyset$ **then**
        $\mathcal{G} \leftarrow \{\{\mathcal{G} - \{node_i \rightarrow node_j\}\} \cup \{node_j \rightarrow node_i\}\}$
        $update(\mathcal{D}, \mathcal{G}, \mathcal{B}, trainData)$
      end if
    end for
  end while
end procedure

Algorithm 2. Update Bayesian Network Structure

procedure UPDATE($\mathcal{D}, \mathcal{G}, \mathcal{B}, trainData, \mathcal{BIC}score$)
  if $\mathcal{BIC}score \leq \mathcal{BIC}(\mathcal{G}, trainData)$ **then**
    $\mathcal{BIC}score \leftarrow \mathcal{BIC}(\mathcal{G}, trainData)$
    $\mathcal{B} \leftarrow updateNetwork()$
  end if
end procedure

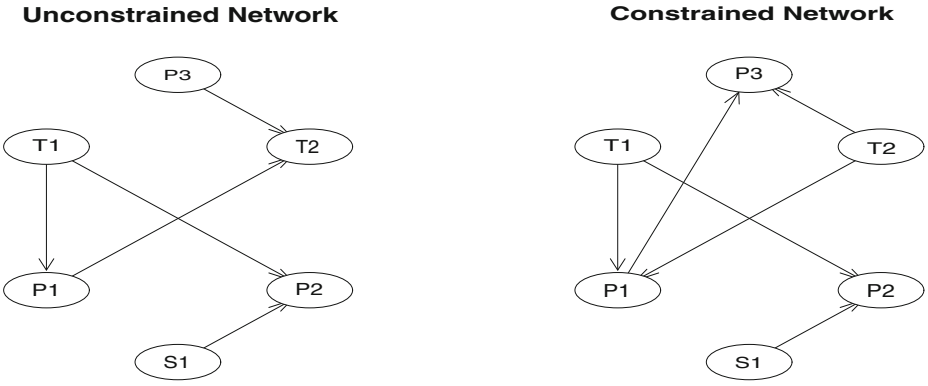

Fig. 1. Variation in Network Structure with input traces and specification used as constraints.

3.2 Node Value Prediction

Simulation phase in the proposed scheme consists of two steps:

- Topologically order the network structure and predict the responses of node values to input trace.
- Use Bayesian inference or a multi-model regression for predicting node values during simulation.

Lack of an edge is true in the physical system if the Bayesian network follows the Markov property. Verifying the existence of Markov property depends on domain knowledge, which we assume is unavailable. Experts can verify and nullify any unwanted dependency among nodes by parameterization of conditional probability table. Using constrained hill climbing, we use the basic knowledge of always treating performance parameters as dependent variables.

As there is lack of any other domain knowledge, we try an alternative approach by using MARS [6] to build a multi-model regression scheme. MARS algorithm uses variable importance estimation that can identify the best subset of features required in building the regression model. This may compensate for the lack of domain knowledge and indirectly eliminate any unrealistic dependency. Each non-root node in the modeled network is treated as a response variable and their parents as predictor variables. Using this strategy, at most set of $n-1$ different regression datasets for non-root nodes are extracted from $\mathcal{D}$. Each of the $n-1$ datasets will generate unique models for prediction of $n-1$ different nodes.

Another possible approach for extracting $n-1$ datasets would be to use non-root node as a response variable and the Markov Blanket as predictor variables. Markov blanket of a node consists of the node's parents, its children, and its children's parents. It is useful in finding optimal feature subsets [15]. Low prediction errors for nodes with high Markov Blanket is necessary. If prediction errors for such nodes are high, then the rate of propagation of these errors across network might also be high.

Inputs to physical systems are not random in a lot of domains. Fixed patterns or a time series are observed in the inputs to a physical system (e.g., Inputs to computer systems do show such behavior as shown in [8]). Hence, regression approach is suited to capture the response of nodes, which may also have a trend. The network built will have input traces as root nodes and the simulation responses (i.e., the value of performance parameters) will depend on the kind of input workload traces that are fed during the simulation process.

Algorithm 3 shows the steps followed in building the multi-model regression scheme during simulation phase. Regression for isolated nodes with no parents and children are built by considering the constraints as their predictor variables. Once the multi-model regression is built, past workload traces for which system behavior has to be evaluated is used as input to the Algorithm 4. Node values are predicted in topological order to avoid encountering missing data during the simulation process. Alternate approach of predicting the node values using classical Bayesian Network involves building of Condition Probability Table that

Algorithm 3. Extract Rules

procedure EXTRACTSIMULATIONRULES($\mathcal{B}, \mathcal{D}, trainIndex, constraintFeatures$)
  $trainData \leftarrow \mathcal{D}[trainIndex,]$
  for all node in $\mathcal{B}$ **do**
    $node.parents \leftarrow getParents(node)$
    if $\mathcal{N}(node.parents) == 0$ **then**
      $node.parents \leftarrow constraintFeatures$
    end if
    $responseVar \leftarrow node$
    $rules[node] \leftarrow fitModel(trainData, node.Parents, responseVar)$
  end for
  return $rules$
end procedure

best fits the network structure. We compare the prediction capabilities of using Constrained Bayesian Network(CBN) and Constrained Bayesian Network with MARS(CBNM) on real world data and the results are discussed in Section 4

Algorithm 4. Simulator Response

procedure SIMULATERESPONSES($\mathcal{B}, orderedRules, testworkloadtrace, cpt$)
  $response_{MARS}[] \leftarrow zeroes()$
  $response_{BN}[] \leftarrow zeroes()$
  $orderedRules \leftarrow topologicalSort(\mathcal{B})$
  for all rule in $orderedRules$ **do**
    $response_{MARS} \leftarrow predict(rule, node, testworkloadtrace, response)$
    $response_{BN} \leftarrow predict(cpt, node, testworkloadtrace, response)$
  end for
  return $response$
end procedure

3.3 Simulation Efficiency

The simulation efficiency is measured by comparing the predicted performance parameters with actual values of the system for an input trace. Three measures are used to evaluate the simulation efficiency as shown in Table 1.

In all the three error measures e_t is the prediction error at time t. As the range of values for each parameter within the dataset can vary, we introduce the normalized mean absolute error. RMSE and MAE may be dominated by error instances, which are too far from true values. Such scenarios are common in cases where range of values for a regression algorithm is high. Hence, NMAE is used to give better measure the overall simulation ability in our experiments. We report the percentage NMAE in this paper, as NMAE $\times$ 100

Table 1. Error Measure for Simulation

Error Measure	Formula
Mean Absolute Error (MAE)	$\frac{1}{n}\sum_{t=1}^{n}\lvert e_t\rvert$
Normalized Mean Absolute Error (NMAE)	$\frac{1}{n}\sum_{t=1}^{n}\frac{\lvert e_t\rvert}{\max(Y)-\min(Y)}$
Root Mean Square Error (RMSE)	$\sqrt{\frac{1}{n}\sum_{t=1}^{n}e_t^2}$

4 Results

4.1 Data Description

Two environments are simulated to evaluate the CBN and CBNM. First environment consists of four servers with different specifications (*Spec*). A set of applications are executed on this clustered setup and their performance parameters ($\mathcal{P}_{t_{\mathcal{C}_m}}$) related to the components of each server are recorded. Real world applications with traces ($\mathcal{T}$) are also recorded. Performance parameters and the workload traces are measured at specified time interval on each server. The monitored components ($\mathcal{C}_m$) in this experiment include processor, RAM, memory manager, network and I/O devices. Each component has variable number of performance parameters recorded in $\mathcal{P}_{t_{\mathcal{C}_m}}$ for the m_{th} component at time t. *Spec* vector consists of specifications related to processor, network card, RAM, graphics card and hard drive. Test application traces $\mathcal{T}$ are provided to simulate the performance parameters and error measures are evaluated in the test phase. Simulation for this dataset will be referred to as SIM1.

The second environment considered for simulation is a NetApp storage system also called a filer Each component within the storage architecture can produce useful information about its performance through multiple performance parameters called counters. More details about the components and their related parameters are available in [13,14]. The workload trace contains information about the type of I/O operation, and its destination for performing the operation. $\mathcal{D}$ consists of performance parameters, features of workload trace and specification of the filer. Simulation related to this dataset will be referred to as SIM2. Bayesian Networks constructed might turn out to have a different structure in each iteration. Hence, the average error rates for each node over 10 iterations are presented.

4.2 Analysis

We first discuss the results of SIM1 as it involves variation in the physical system (servers) and the *Spec* vector has a role in the modeling phase. There are 30 performance parameters, 8 specification parameters and 5 workload trace parameters. Given a dataset with 90,000 instances and 43 features, 13 non-performance

related parameters are treated as constraints to build the Bayesian Network. 80,000 instances are used as input for building the network. Application traces available in rest of the 10,000 instances are used as input to Algorithm 4. The built network has 30 non-root nodes and 13 root nodes resulting in 30 different regression schemes have to be built using MARS.

Algorithm 3 is applied to the Bayesian Network to build CPT and multi-model regression for SIM1. Any new application trace that needs to be simulated will be provided as a matrix with each column representing the workload trace features. The entire trace of application execution is fed to Algorithm 4, which predicts the responses of each performance parameter for both CBN and CBNM. Figures 2 shows the plot for three different simulation efficiency measures defined in the previous section. Lower the values in all the three subplots, better is the simulation efficiency. Range of values in the y-axis should be observed carefully as it explains the need for normalization. An example of how NMAE is able to estimate errors more accurately is seen in the plots. Though predictions for node index 4 have very high MAE and RMSE, the true estimate of prediction is seen in NMAE. As the range of values for the parameter indexed at 4 is very high, RMSE and MAE will be dominated by few bad predictions. In certain cases, even though the error might be perceived to be high, the relative error with respect to the range of values might be low. Hence, RMSE and MAE can over estimate the prediction errors, which can lead to wrong conclusions. In a majority of parameter predictions, CBN has better simulation results compared to CBNM.

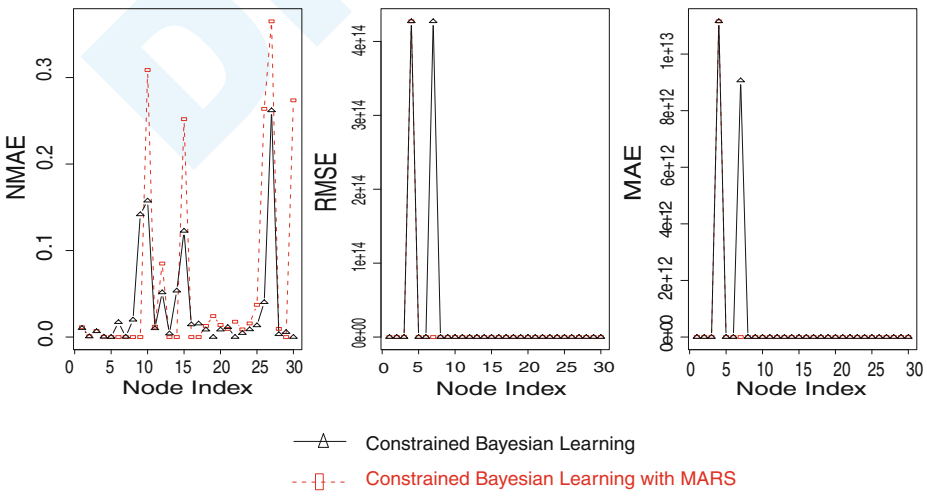

Fig. 2. Normalized MAE, RMSE and MAE for estimating simulation efficiency of server.

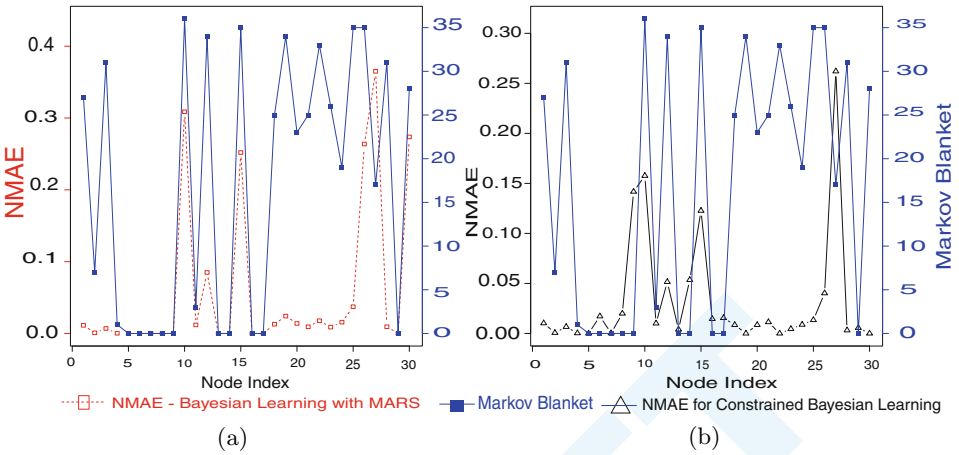

Fig. 3. Variation NMAE with the size of Markov Blanket for Server simulation.

The average percentage NMAE over all the 30 predictions was observed at 6% for CBNM and 3.75 CBN%. A plot comparing NMAE and Markov Blanket for each node is shown in Figure 3. Nodes with larger Markov Blanket should ideally have higher prediction accuracy (lower NMAE) as they influence large number of nodes. CBN has better values of NMAE compared to CBNM for most of the nodes in general and nodes with higher Markov Blanket in particular. The same idea can be extended to the number of children of a node as shown in Figure 4. Higher the number of children, lower should be the NMAE. This pattern is followed to a large extent, which can be the reason for higher simulation efficiency as indicated by the average NMAE.

The second simulation in our experiment was performed on a storage filer. Storage filer capable of handling TeraBytes of data was used for monitoring and data collection. There were a totally of 9,000 instances in the dataset with 88 performance parameters, 5 workload parameters and 6 *Spec* parameters. As there was only one filer used for data collection, *Spec* did not play a major role in the simulation. 8,000 instances are used in modeling and simulation phase with rest of the 1,000 instances being used for testing simulation efficiency. Figures 5 shows the plots related to evaluation of simulation efficiency. The average percentage NMAE for prediction of all node values was around 12.9% for CBNM and 4.6% for CBN. Both the simulations have shown a better simulation efficiency for CBN, as embedding multi-regression models for each node has failed to learn the set of parameters identified by the constrained network structure.

The average Markov Blanket in both the simulations were close to half of the total number of nodes in the network indicating high interdependence among components. As seen in SIM1, the nodes with higher number of children or with a larger Markov Blanket should be analyzed to see if their NMAE values are low. Figure 6 & 7 show certain nodes with higher children or larger Markov Blanket have higher values of NMAE, which can lead to error propagation.

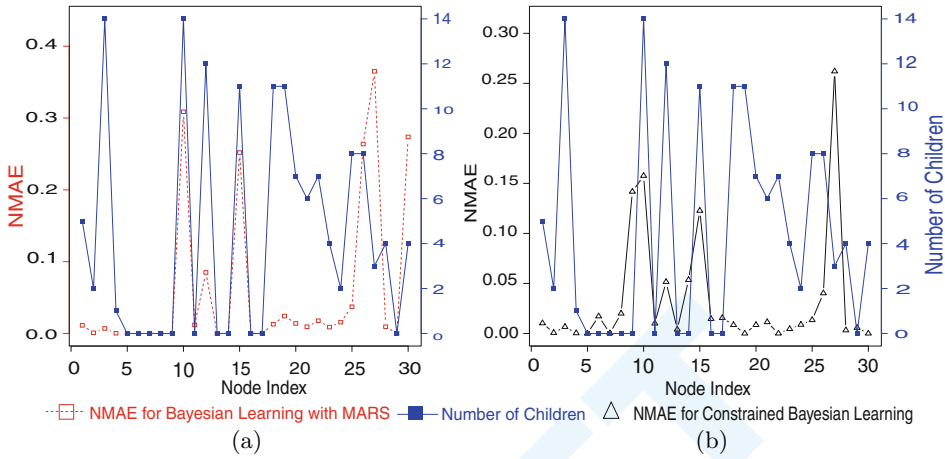

Fig. 4. Variation NMAE with the size of Children for Server simulation.

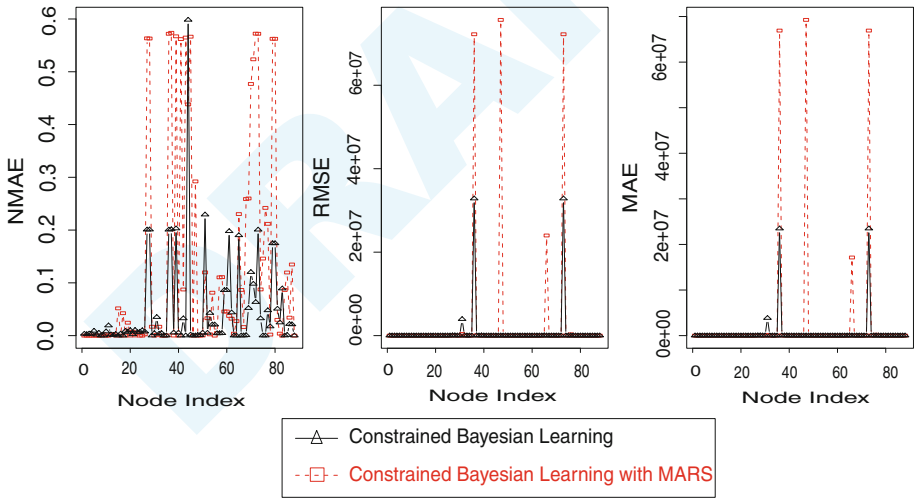

Fig. 5. Normalized MAE, RMSE and MAE for estimating simulation efficiency of Filer.

Also, node index obtained from topological ordering are a function of the depth of a node in the DAG. Lower the node index, higher is the probability of error propagation. Hence, nodes with lower index should be carefully analyzed during model construction to see how well the structure truly represents reality. Fig 6(a) & 6(b) clearly shows that majority of nodes with large Markov Blanket have lower NMAE values for CBN when compared to CBNM leading lesser error propagation.

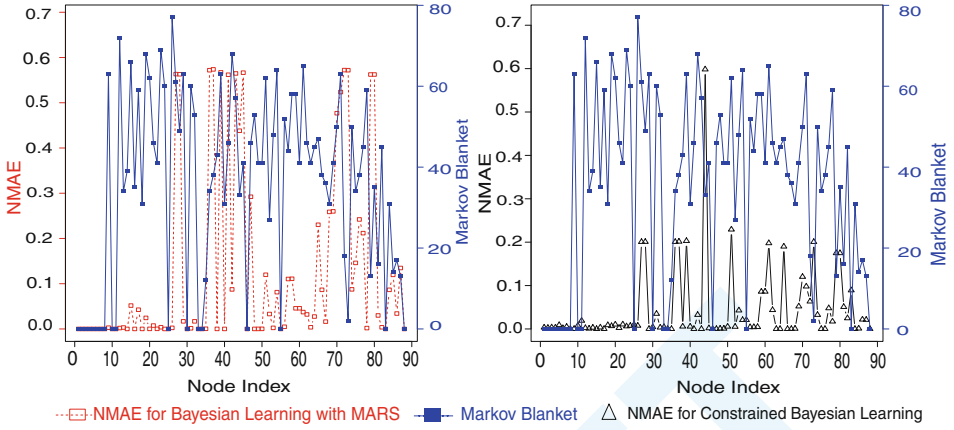

Fig. 6. Variation NMAE with the size of Markov Blanket for Filer simulation.

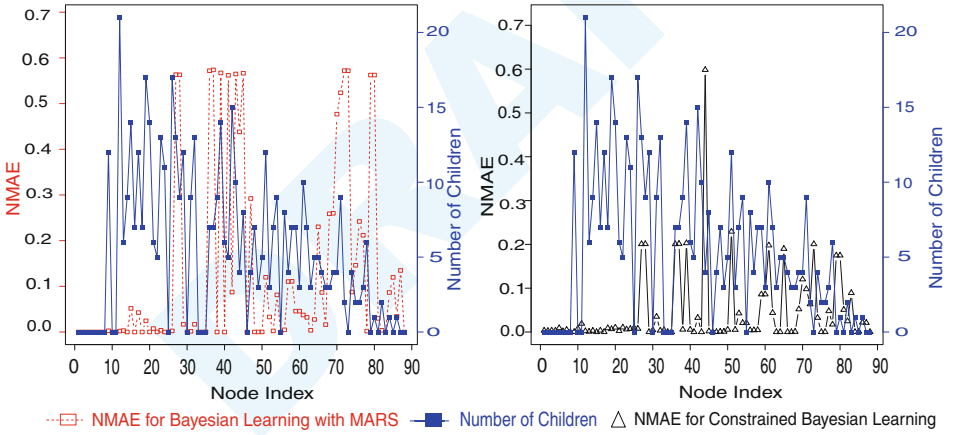

Fig. 7. Variation NMAE with the size of Children for Filer simulation.

5 Conclusion

Simulation provides major advantage in designing, developing, analyzing and verifying the performance of a physical system. A generic framework for non-domain experts to simulate system behavior was presented in this paper. Learning structure of Bayesian Network using Constrained Hill climbing algorithm has shown promising results in simulating the system performance. Workload traces can be directly provided to the Simulation process that can simulate multiple component behavior. Only assumption in the proposed scheme is the presence of a management interface to collect parameters. Lack of any other assumption helps in extending this scheme to other domains for simulation. Topological ordering of the nodes

ensures that there are no missing information while predicting the node values. Exploring regression scheme during simulation phase can help in not just simulating but also scaling the simulation process. Future work can focus on increasing simulation accuracy, especially nodes with higher number of children or large Markov Blanket. Integration of prototype simulator with a discrete event system will be attempted in our future work.

References

1. Breese, J.S., Heckerman, D.: Decision-Theoretic Troubleshooting: A Framework for Repair and Experiment. CoRR abs/1302.3563 (2013)
2. Chickering, D.M., Geiger, D., Heckerman, D.: Learning Bayesian networks: search methods and experimental results. In: Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics, pp. 112–128, January 1995
3. Chickering, D.M.: Learning Bayesian networks is NP-complete. In: Learning from Data: Artificial Intelligence and Statistics V, pp. 121–130. Springer-Verlag (1996)
4. Daly, R., Shen, Q., Aitken, S.: Review: Learning bayesian networks: Approaches and issues. Knowl. Eng. Rev. **26**(2), 99–157 (2011)
5. Davies, S., Moore, A.: Bayesian networks for lossless dataset compression. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 1999, pp. 387–391. ACM, New York (1999)
6. Friedman, J.H.: Multivariate adaptive regression splines. The Annals of Statistics **19**(1), 1–67 (1991)
7. Gerathewohl, S.J.: United States: Fidelity of simulation and transfer of training [electronic resource] : a review of the problem / by Gerathewohl, S.J., Dept. of Transportation, Federal Aviation Administration, Office of Aviation Medicine [Washington, D.C.] (1969)
8. Hamerly, G., Perelman, E., Lau, J., Calder, B., Sherwood, T.: Using machine learning to guide architecture simulation. J. Mach. Learn. Res. **7**, 343–378 (2006).
9. Heckerman, D.E., Horvitz, E.J., Nathwani, B.N.: Toward normative expert systems: Part I. The Pathfinder project. Methods of Information in Medicine **31**(2), 90–105 (1992)
10. Heckerman, D.: A tutorial on learning with bayesian networks. Tech. Rep. MSR-TR-95-06, Microsoft Research, March 1995
11. Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. CoRR abs/1302.6815 (2013)
12. Helle, I.: Assessing oil spill risks in the northern baltic sea with bayesian network applications. abs/1302.3563 (2015)
13. Inc, N.: Report fields and performance counters (2013). https://library.netapp.com/ecmdocs/ECMP1222473/html/GUID-33405732-F0C7-4FD3-8AB8-39A5B76ACC77.html
14. Inc, N.A.: Netapp Unified Storage Performance Management Using Open Interfaces, March 2010
15. Koller, D., Sahami, M.: Toward optimal feature selection. In: 13th International Conference on Machine Learning, pp. 284–292 (1995)
16. Lozano, A.C., Li, H., Niculescu-Mizil, A., Liu, Y., Perlich, C., Hosking, J., Abe, N.: Spatial-temporal causal modeling for climate change attribution. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2009, pp. 587–596. ACM, New York (2009)

17. Olstam, J.J., Lundgren, J., Adlers, M., Matstoms, P.: A framework for simulation of surrounding vehicles in driving simulators. ACM Trans. Model. Comput. Simul. **18**(3), 9:1–9:24 (2008)
18. Skaanning, C., Jensen, F.V., Kjærulff, U.B.: Printer troubleshooting using Bayesian networks. In: Logananthara, R., Palm, G., Ali, M. (eds.) IEA/AIE 2000. LNCS (LNAI), vol. 1821, pp. 367–380. Springer, Heidelberg (2000)
19. Sterbenz, J., Cetinkaya, E., Hameed, M., Jabbar, A., Rohrer, J.: Modelling and analysis of network resilience. In: 2011 Third International Conference on Communication Systems and Networks (COMSNETS), pp. 1–10, January 2011
20. Tsamardinos, I., Brown, L.E., Aliferis, C.F.: The max-min hill-climbing bayesian network structure learning algorithm. Machine Learning **65**(1), 31–78 (2006)
21. Westkamper, E.: Simulation based on learning methods. Journal of Intelligent Manufacturing **9**(4), 331–338