Supporting Scalable Bayesian Networks using Configurable Discretizer Actuators

Isaac Osunmakinde, *SMIEEE* and Antoine Bagula
Department of Computer Science, Faculty of Sciences, University of Cape Town,
18 University Avenue, Rhodes Gift, 7707 Rondebosch, Cape Town, South Africa
{segun, bagula}@cs.uct.ac.za

Abstract. We propose a generalized model with configurable discretizer actuators as a solution to the problem of the discretization of massive numerical datasets. Our solution is based on a concurrent distribution of the actuators and uses dynamic memory management schemes to provide a complete scalable basis for the optimization strategy. This prevents the limited memory from halting while minimizing the discretization time and adapting new observations without re-scanning the entire old data. Using different discretization algorithms on publicly available massive datasets, we conducted a number of experiments which showed that using our discretizer actuators with the hellinger's algorithm results in better performance compared to using conventional discretization algorithms implemented in the Hugin and Weka in terms of memory and computational resources. By showing that massive numerical datasets can be discretized within limited memory and time, these results suggest the integration of our configurable actuators into the learning process to reduce the computational complexity of modeling Bayesian networks to a minimum acceptable level.

Keywords: Intelligent Systems, Massive datasets, Bayesian Networks, Discretization, Scalability.

1 Introduction

Bayesian network models often formulate the core reasoning component of some intelligent systems because of their suitability in handling complex problems [1]. Researchers and practitioners have stressed that learning such models from environments captured as massive datasets is computationally intensive [2] [3]. In practice, it is convenient to say that massive datasets are relatively defined based on the capacity of the machine used for learning and are also dependent on users' execution urgency. For example 50,000 records of dataset may be massive for a machine and moderate or small for another. The intensity on the datasets implies that too much of computational time is expended and limited memory space may crash during these operations. This affects business and research deliveries, and may hinder the growing usage of Bayesian networks in industries that keep massive datasets to build intelligent systems. From our practical knowledge, improving the performance of discretization is obviously a sound basis for optimizing Bayesian networks' learning. Intelligent system engineers do not want to wait too long to make the reasoning component ready for use.

Most of the existing conventional algorithms load entire massive datasets onto the limited memory for discretizations and a column is processed one at a time. Time is expended for loading, discretizing and probably saving back, which could otherwise have been minimized. Carrying out the discretization process on massive datasets whose size is more than the available allocated memory may currently not be practically feasible. Achieving this requires scalability of discretization methods which is very challenging.

A number of fairly recent studies have developed good conventional algorithms to discretize datasets but they fall short in considering the scalability of their approaches [4] [5]. Among the rationales in this scalability research is studying how massive are the datasets used in the existing discretization approaches. For examples, Li *et al.*[4] suggested feature selection heuristics for discretizing bio-medical data where they evaluated it on a notable lung-cancer dataset of 10,000 records. Lee's supervised algorithm [6] is similar to Li *et al.* but he used the entropy of intervals for discretization. Also, Lee used a maximum of 3,163 records of hypothyroid dataset to evaluate his work. Out of the 16 datasets used by Dougherty *et al.* [5], the maximum size is Australian dataset with 6,650 records.

The computational times and memory usages of the methods described above are not known though these are two important parameters upon which the scalability of discretizing massive datasets for learning networks depends. In an attempt to address scalability of discretizations for the core component of intelligent systems, the available open source network learning applications (e.g. Weka [7] and Hugin [8]) force users to discretize all numeric values of the attributes present in the datasets. However, certain numeric attributes in real life are not necessarily required to be discretized. In this research, we proposed configurable discretizer agent actuators which dynamically scale limited memory and improve computational time efficiency. In a number of comparative evaluations, the actuators outperform the conventional discretization approaches in speed and memory management respectively. Our major contributions are:

- The development of a new generalized configurable discretizer actuators and its system model to optimize the core intelligent system component through discretization processes.
- The evaluation of this configurable actuator on publicly massive datasets using different discretization algorithms implemented in Weka and Hugin systems.

The rest of this paper is arranged as follows: in section 2, we introduce the background of Bayesian networks, discretization algorithms, dynamic memory management scheme and the agent architecture as the theoretical foundations of our configurable discretizer actuators. Section 3 presents the system model and the configuration of the discretizer agent actuators. Section 4 presents the experimental evaluations of the discretization time and memory scalability using publicly available datasets from UCI [9] (University of California Irvire) repository used by intelligent systems researchers. We conclude the paper in section 5.

2 Theoretical Background

2.1 Bayesian Network Models

A Bayesian belief network is formally defined as a directed acyclic graph (DAG) represented as G = {X(G), A(G)}, where X(G) = {X_1,…,X_n}, vertices (variables) of the graph G and $A(G) \subseteq X(G) \times X(G)$, set of arcs of G. The network requires discrete random values such that if there exists random variables X_1, . . ., X_n with each having a set of some values x_1, . . ., x_n then, their joint probability density distribution is defined in equation 1;

$$pr(X_1,..., X_n) = \prod_{i=0}^{n} pr(X_i \mid \pi(X_i)) \quad (1)$$

where $\pi(X_i)$ represents a set of probabilistic parent(s) of child X_i [1]. A parent variable otherwise refers to as *cause* has a dependency with a child variable known as *effect*. Every variable X with a combination of parent(s) values on the graph G captures probabilistic knowledge as conditional probability table (CPT). A variable without a parent encodes a marginal probability. Learning the suitable networks from massive datasets is computationally intensive as stated above.

2.2 Discretization Algorithms

Discretization algorithms are techniques which are used as preprocessing key operations in learning Bayesian models [5] [6]. They classify numerical data into their corresponding interval values relatively to the patterns in the data attributes. Weka and the Hugin systems use discretization algorithms which are built around the simple binning and minimum description length (MDL) methods [1]. Simple binning include an equal-width method using an unsupervised discretization approach which divides attribute values into k equal sizes. The seed k is supplied by users while equal-width finds maximum and minimum attribute values and they are used to determine data intervals. The Hellinger-based algorithm uses interval entropy function E(.) as a justification for quality discretization to accommodate any datasets. The entropy of any interval between a and b is shown in equation 2 [6].

$$\mathrm{E}([a, b]) \equiv \sqrt{\left| \sum_{i} \left(\sqrt{pr(x_i)} - \sqrt{pr(x_i \mid \overline{ab})}\right)^2 \right|} \quad (2)$$

As a basis of the algorithm, the values x_i of the target attribute being discretized are sorted accordingly and they form a column of intervals. The probability distribution of x_i is represented as $pr(x_i)$. The scheme in the next subsection is therefore adopted to prevent the out-of-memory problems in the learning processes.

2.3 Dynamic Memory Management Scheme

The dynamic memory management scheme used in the Loci framework [10] is an economical solution which manages the memory by allocation and de-allocation of data structures based on the lifetime of data structures. Thus, in order to accommodate

discretization of large datasets within a limited memory, we extracted and interpreted the scheme from [10] as follows: (i) pre-allocation of memory to data structures, (ii) incorporate relevant memory management operations, (iii) invoke loop scheduling techniques, and (iv) recycle memory from data structures.

Pre-allocation with partitioning of the entire memory alone in scheme i does not benefit space saving until the others in the sequence are involved. Many parallel algorithms exploit scheme i as a trade-off to optimize speed but suffer from peak memory requirement. A possible relevant management operation in ii is the use of remote memory or secondary storage devices, for example. These management operations are generalized concepts of virtual memory. A virtual memory is a multilevel store which gives a large process an impression that it has more primary memory to itself, while it actually uses external disk devices as a supplement [11]. In iii, examples of loop scheduling techniques are multiple nested iterations, recursions, synchronizations, etc. Also in iv, at the end of every schedule or lifetime, memory is recovered from data structures after its execution. Thus, this scheme empowers a system to accommodate massive datasets within a limited memory without a halting problem. The next subsection also describes the basic agent architecture as fundamentals of our configurable actuators.

2.4 A Basic Agent Architecture

Among the classes of agents used in intelligent systems, the software agent as related to this work perceives from the components of *environments* through *sensors* and acting upon the environment through *actuators* [1]. According to Russell [1], a software agent can sense its environment using file content or network packets and also uses writing files or packets as actuators to act on the environment. From Russell's illustration, when environment is perceived, some forms of machine learning algorithms are used to interpret the *percepts*. They consequently generate the instructions required by the actuators to carry out actions on the environment.

The positions of the agent and the environment are often far apart which possess distributed properties. It illustrates that agents can be sent over a network to carry out specific tasks and can also provide services to other components on a given machine. It is deduced from here that agent actuators can be characterized with mobility as they include their required information in their description. Their independence influences the design of components for distributed agents which motivate the development of the configurable discretizer agent actuators in this study. Section 3 now describes the proposed configurable actuators.

3. The Generalized Configurable Discretizer Actuators

3.1 The System Model for the Actuators

Figure 1 depicts the system model that we used to accomplish complete scalable discretization. If either space or time is optimized, it is an incomplete scalability as a trade off is not beneficial to the networks used in intelligent systems. Our strategy combines the memory management scheme in subsection 2.3 and the architecture in subsection 2.4. In this strategy, an actuator is dedicated and sent to discretize values

of one or more attributes. For balancing purposes, a number of actuators, rather than all, are heuristically set by users and concurrently distributed at a time. Discretization time is faster as the actuators act on more than one attributes at a time. As the actuators complete discretization of some attributes in a pass, they are returned to the symmetric processors that reschedule them for subsequent attributes. With this, memory is continually and dynamically allocated which then recycles each time there is scheduling of actuators for discretization.

We now define the major components of Figure 1 as follows: discretizer *agent actuators* described in the previous paragraph, discretization *algorithm*, *massive environment* (or datasets), storage of previous discretized *parameters* and subsequent *observations* made after discretizing the massive datasets. The algorithm which resides on the limited memory of a machine generates tuples of intervals for the actuators to discretize values of the attributes remotely. We adopted the Hellinger-based algorithm in Lee's work [6] as a proof of concept since this research focuses on supporting the optimization of the core reasoning component of intelligent systems through scalable discretizations. The component of the massive dataset (or environment) is kept away from the limited memory and its attributes' values are acted upon concurrently in a secondary storage or across a network. This provides a competitive advantage in developing countries where discretization process can be accidentally suspended probably due to electricity power failure but modeling continues where the process stops.

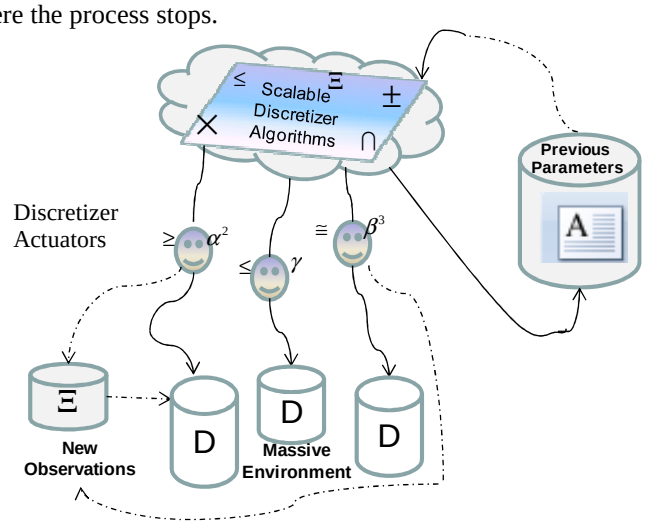

Figure 1: System Model for Discretizer Agent Actuators

Also, the previous parameters are used adaptively to discretize subsequent observations instead of re-scanning the entire old massive dataset. The last tuples of intervals if the data patterns remain the same, the data types for all attributes, etc are examples of previous parameters. The configuration used by the discretizer actuators is designed and described in the next subsection.

3.2 The Configurable Discretizer Actuators

We designed and configured these actuators as shown in Figure 2 with dynamic packets of information to act upon the environments. The content of the packet consists of the *control information* and the *environments*. The control information provides dynamic set of instructions that the actuators need to use to act upon the environments.

The constituents of the control information depicted in Figure 2 are as follows: *source-address* (e.g. agent-actuator-id), *destination* is any universal resource locator of the data (e.g. secondary storage or network machine address), *node-ids* (or attribute names) and *actions* (e.g. advance discretization scripts using the interval bins) taken by the actuators. The constituents retain their usual meanings as described. The environment acted upon is the schema table (or dataset) at various destinations. The configuration of the actuators can be expanded or modified as new functionalities are provided. Thus, our discretizer actuators are concurrently distributed because they are lightweight, mobile and independent which are suitable on single user machine and distributed architecture. Section Four brings our theory to practice.

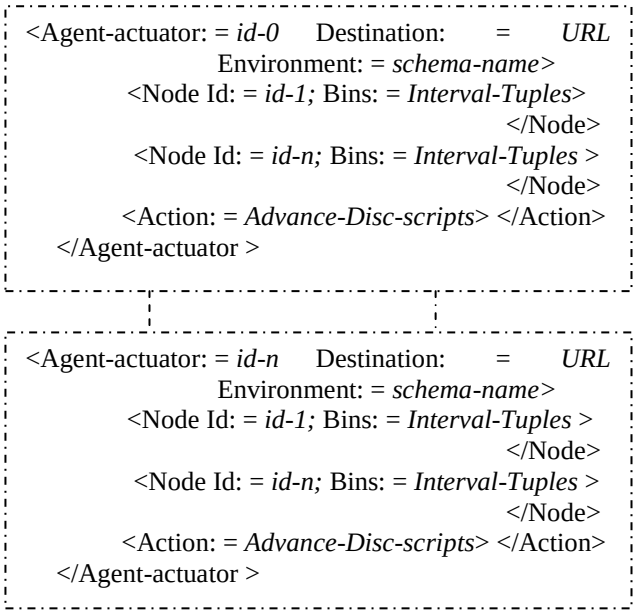

Figure 2: Configuration of the discretizer actuators convert numerical to discrete datasets

4. Experimental Evaluations

One of the objectives of our proposed discretizer actuators is to bring theory to practice with an emphasis on applications and practical work. The algorithms compared are Hellinger's algorithm using our actuators, Weka and Hugin algorithms. They are experimented on three public [9] massive datasets including (1) El-Nino, (2)

Census-Income (KDD) and (3) Pseudo periodic synthetic time series. The El-Nino data set contains oceanographic and surface meteorological readings. The Census-Income (KDD) contains weighted census dataset. Finally, the pseudo dataset is designed for testing indexing schemes in time series.

In practice, the major contributing factors that affect discretizations and modeling performances are the number of instances, columns and number of states (distinct values) in each column of the datasets. The three datasets have varying sizes with over 178,080, 200,000, and 100,000 instances respectively. They include 11, 9 and 10 numeric columns respectively. The pseudo dataset has the worst scenario because its number of instances is equal to the number of its distinct 100,000 states.

4.1 Experiment 1: Comparing Algorithms

Table 1: Comparing Configurable Actuators using Hellinger, Weka and Hugin algorithms

Data Sets	Methods	Number of Actuators	Speed (secs)	Mem-usage (MB)	Status
El-Nino (178,080)	Configurable actuators using Hellinger	1 2 3 4 5 11	264 148 105 71 69 33	17.3 17.6 17.8 17.9 18.0 18.2	Ready to Model
	Weka	1	201	59.0	Out of Memory
	Hugin	1	200	66.8	Towards Memory failure
Census-Income-KDD (200,000)	Configurable actuators using Hellinger	1 2 3 4 9	177 96 78 54 29	17.4 18.1 20.2 20.8 22.6	Ready to Model
	Weka	1	173	39.2	Out of Memory
	Hugin	1	176	39.6	Towards Memory failure
Pseudo (100,000)	Configurable actuators using Hellinger	1 2 3 4 5 10	174 104 79 66 64 54	23.4 23.9 25.1 26.5 26.9 29.0	Ready to Model
	Weka	1	169	51.2	Out of Memory
	Hugin	1	165	67.2	Out of Memory

The objective here is to find the impact of our configurable actuators on the algorithms. The results depicted by Table 1 are a summary of the average performance of the three algorithms on the three datasets in terms of speed and memory used by the configurable actuators. For each experiment, the speed includes the time to save back into the secondary memory other than leaving the results on the volatile RAM. In all the cases, the results revealed that our configurable actuators using Hellinger's algorithm discretized successfully and was ready to proceed to modelling while the other algorithms (Weka and Hugin) suffered from memory problems by exhibiting an "out of memory" or a "towards memory failure" states. Observe in Table 1 that the Hellinger algorithm performed tremendously better than the other algorithms when we consider the results provided by the highest (or best) number of actuators in each dataset. These results suggest that using our configurable discretizer agent actuators with the Hellinger's algorithm is an economically scalable solution which supports the optimization of Bayesian intelligent modeling.

4.2 Experiment 2: Comparing Execution Speed

From the results in Table 1, we specifically compared the discretization speeds of configurable actuators using Hellinger, Weka and Hugin on the El-Nino dataset stored remotely on a secondary storage. In the same vein with Weka and Hugin which use a processor, we discretized the massive datasets with one symmetric processor (or actuator). This set of experiments was successfully repeated by distributing and concurrently increasing the number of configurable actuators while recording the discretization time as shown in Figure 3. The results show that using the Hellinger's algorithm, an increase in the number of actuators makes the discretization process faster.

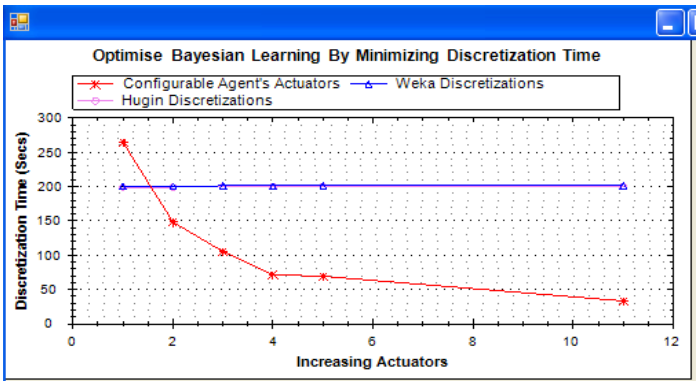

Figure 3: Increasing number of actuators on El-Nino dataset minimizes (or speeds up) discretization time better than Weka and Hugin discretizations.

In contrast, when looking at the Weka and Hugin discretizations in Figure 3, one can observe that varying the number of actuators did not improved the discretization time. By comparing the discretization time of the highest (best) number of actuators

used to the usual one processor of Weka and Hugin, within the allocated limited memory, our configurable actuator using Hellinger's algorithm is faster than Weka and Hugin by 83% in Figure 3. A similar performance pattern is revealed like Figure 3 when we adapted new observations to the previous discretization parameters (intervals used for the old datasets). By cross validation [1], 15% each of the datasets were selected at random as new observations and were discretized using the previous parameters. Minimization of the discretization time results was also recorded by increasing the actuators similarly to Figure 3.

4.3 Experiment 3: Comparing memory usage

The results described in experiments 1 and 2 above show that users who are not opportune to be in a networking environment or who cannot afford a suitable one, can safely discretize massive data on a machine with limited memory by distributing our configurable actuators. One can observe in Figure 4 from the Weka and Hugin discretizations that varying the number of actuators does not improve on memory usage because all the records are loaded onto the memory at a time. The details of occupied megabytes of memory can be seen in Table 1 which reveals halt states.

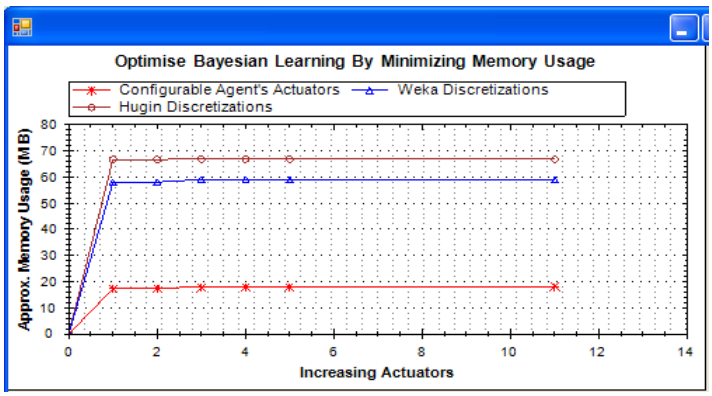

Figure 4: Concurrent distribution of actuators on El-Nino dataset minimizes memory usage better than Weka and Hugin discretizations.

From the results in Figures 7, our configurable actuators using Hellinger's algorithm successfully managed the same limited memory by concurrently exploiting secondary storage resources on remote locations (e.g. hard disk on a machine or on workstations). Though there are slight increases in memory usage as the number of actuators increases, one can observe in Figure 4 that our actuators reduce the memory usage to a minimum acceptable level. For example, this shows that the configurable actuators save 69.2% and 72.8% of the limited memory from crashing as compared to Weka and Hugin discretizations in Figure 4. This once again supports our claim that users cannot afford to trade off between *time* and *space* in real life Bayesian learning via discretization.

5. Concluding Remarks and Future Work

We have proposed in this paper the development of configurable actuators for the discretization of massive datasets as a supportive optimization solution to the computational problems arising in intelligent systems. Experimental results revealed that the use of the configurable actuator is an economically scalable solution to the problem which does not require purchasing expensive hardware. The results support the claim that using our configurable actuators with the Hellinger's algorithm leads to better memory usage and faster discretization of massive datasets compared to conventional algorithms such as Weka and Hugin discretizations.

This study shows that the configurable discretizer actuators can potentially become a more powerful scalable solution that puts an end to the computational problems raised by the learning of network models.

Acknowledgements. Our appreciation goes to the University of Cape Town and the Complex Adaptive Systems (Pty.) Ltd, for their financial supports.

References

1. Russell, S., Norvig, P.: Artificial Intelligence, A Modern Approach, 2nd Edition, Prentice Hall Series Inc. New Jersey 07458 (2003)
2. Chickering, D., Heckerman, D., Meek, C.: Large-Sample Learning of Bayesian Networks is NP-Hard. The Journal of Machine Learning Research, Volume 5, MIT Press, 1287 – 1330 (2004)
3. William, H., Haipeng, G., Benjamin, P., Julie, S.: A Permutation Genetic Algorithm For Variable Ordering In Learning Bayesian Networks From Data. Proceedings of Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers Inc, 383-390 (2002)
4. Li, J., Liu, H., Wong, L.: Mean-entropy Discretized Features are Effective for Classifying High-dimensional Biomedical data. Proceedings of the 3rd ACM SIGKDD Workshop on Data Mining in Bioinformatics, Washington, DC (2003)
5. Dougherty, J., Kohavi, R., Sahami, M.: Supervised and unsupervised discretization of continuous features. In 12th International Conference on Machine Learning (1995)
6. Lee, C.: A Hellinger-based discretization method for numeric attributes in classification learning, Knowledge-Based Systems, volume 20(4), pages 419-425 (2007)
7. Witten, I.H., Eibe, F.: Data Mining Practical Machine Learning Techniques and Tools, University of Waikato - WEKA, Morgan Kauffman (1999) DOI =http://www.cs.waikato.ac.nz/$\sim$ml/weka/
8. Olesen, K.G., Lauritzen, S.L., Jensen, F.V.: aHugin: A system creating adaptive causal probabilistic networks. In D. Dubois, M. P. Wellman, B. D'Ambrosio, and P. Smets, editors, *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, pages 223-229, Stanford, California, July 17-19. Morgan Kaufmann, San Mateo, California (1992), DOI = http://hugin.sourceforge.net/download/.
9. Newman, D., Hettich, S., Blake, C., Merz, C.: UCI Repository of Machine Learning Databases (University of California, Department of Information and Computer Science, Irvine,CA),(1998).DOI= http://www.ics.uci.edu/$\sim$mlearn/MLRepository.html
10. Zhang, Y., Luke, E.A.: Dynamic Memory Management in the Loci Framework, Computational Science. SpringerLink press, ISBN: 978-3-540-26043-1, Volume 3515, Pg 790 – 797 (2005).
11. Graham, R.M.: Principles of Systems Programming, John Wiley & sons Inc. New York (1975).