GPU-Accelerated Clique Tree Propagation for Pouch Latent Tree Models

Leonard K. M. Poon(✉)

The Education University of Hong Kong, Hong Kong SAR, China
kmpoon@eduhk.hk

Abstract. Pouch latent tree models (PLTMs) are a class of probabilistic graphical models that generalizes the Gaussian mixture models (GMMs). PLTMs produce multiple clusterings simultaneously and have been shown better than GMMs for cluster analysis in previous studies. However, due to the considerably higher number of possible structures, the training of PLTMs is more time-demanding than GMMs. This thus has limited the application of PLTMs on only small data sets. In this paper, we consider using GPUs to exploit two parallelism opportunities, namely data parallelism and element-wise parallelism, for PTLMs. We focus on clique tree propagation, since this exact inference procedure is a strenuous task and is recurrently called for each data sample and each model structure during PLTM training. Our experiments with real-world data sets show that the GPU-accelerated implementation procedure can achieve up to 52x speedup over the sequential implementation running on CPUs. The experiment results signify promising potential for further improvement on the full training of PLTMs with GPUs.

Keywords: GPU acceleration · Clique tree propagation
Pouch latent tree models · Parallel computing
Probabilistic graphical models

1 Introduction

Clustering [7,18] is a fundamental problem in machine learning. For soft clustering, the Gaussian mixture models (GMMs) are often used [23]. However, a GMM contains only one latent variable and can produce only a single clustering. This limitation may make GMMs not suitable for modern clustering applications, especially when the data sets contain many attributes and are multifaceted.

The *pouch latent tree models* (PLTMs) [28,29] have been proposed as a generalization of GMMs to allow multiple latent variables. They can produce clusterings on multiple facets and are more versatile for data of higher dimensions. They have been evaluated on several real-world data sets and have been shown better

Published by Springer Nature Switzerland AG 2018
F. Zhang et al. (Eds.): NPC 2018, LNCS 11276, pp. 90–102, 2018.
https://doi.org/10.1007/978-3-030-05677-3_8

than GMMs in terms of model quality and clustering performance [27,29]. However, the structure learning of PLTMs can be more time-demanding than GMMs due to the considerably larger number of possible model structures. For GMMs, the structure learning typically involves only the estimation of the number of the mixture components. In contrast, the structure learning of PLTMs involves determining the number of latent variables, the cardinalities of the latent variables, and the connections among the latent variables and the observed variables. The onerous structure training of PLTMs may pose a serious challenge for applying PTLMs on large data sets. Consequently, previous studies considered data with less than 100 attributes and 2000 samples [27,29]. Those sizes of data sets may be regarded as at most moderate in the Big Data Era.

In recent years, graphical processing units (GPUs) have become more prevalent in scientific computing. They have been demonstrated to achieve significant speedup in different artificial intelligence applications that involve high dimensional data, such as those in the fields of computer vision [16,33], constraint satisfaction [5,15], and clustering [1,4,25,31].

In this paper, we consider the possibility of using GPUs to accelerate the training of PLTMs. We focus on the clique tree propagation algorithm for performing exact inference during the training process. The inference task is used for computing the likelihood and marginal probabilities on a data set during training. It is the most strenuous one and needs to be called recurrently for each data sample and each model structure. To evaluate the performance of the GPU-accelerated inference procedure, we use it to compute likelihood of a given PLTM on a given data set. This computation requires running inference on each data sample and exhibits resemblances to the other computationally intensive steps for PLTM training. Our study thus constitutes an important first step for using GPUs to accelerate the whole training process of PLTMs to make it feasible for application on larger data sets.

The rest of the paper is organized as follows. First, we review PLTMs and the inference procedure in Sects. 2 and 3. Then, we describe the GPU-accelerated inference procedure in Sect. 4. Next, we evaluate the performance of the procedure in Sect. 5. After that, we discuss related work in Sect. 6 and conclude the paper in Sect. 7.

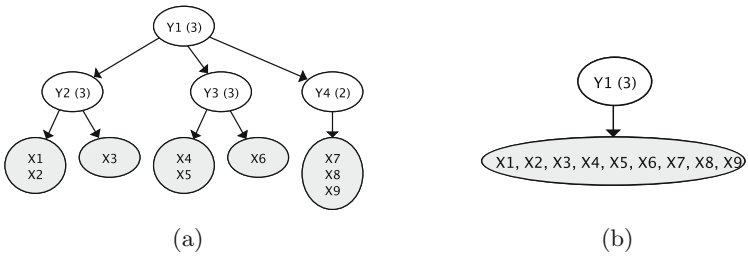

Fig. 1. (a) An example of PLTM. The observed variables are shown in shaded nodes. The numbers in parentheses show the cardinalities of the discrete variables. (b) A GMM depicted as a PLTM.

2 Pouch Latent Tree Models

A *pouch latent tree model* [28,29] is a tree-structured probabilistic graphical model. In the model, each internal node represents a latent variable, and each leaf node represents a set of observed variables. All the latent variables are discrete, whereas all the observed variables are continuous. A leaf node, also called *pouch node*, may contain a single observed variable or several of them. An example is shown in Fig. 1a. In the example model, X_1–X_9 are continuous observed variables and Y_1–Y_4 are discrete latent variables. For technical convenience, PLTMs are often treated as Bayesian networks [26].

Consider a PLTM with observed variables $\boldsymbol{X}$ and latent variables $\boldsymbol{Y}$. The dependency of a discrete latent variable Y on its parent $\mathit{\Pi}(Y)$ is characterized by a conditional discrete distribution $P(y|\pi(y))$. Let $\boldsymbol{W} \subseteq \boldsymbol{X}$ be the variables of a pouch node with a parent node $Y = \mathit{\Pi}(\boldsymbol{W})$. The models assume that, given a value y of Y, $\boldsymbol{W}$ follows the conditional Gaussian distribution $P(\boldsymbol{w}|y) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y)$, with mean vector $\boldsymbol{\mu}_y$ and covariance matrix $\boldsymbol{\Sigma}_y$. Denote the sets of pouch nodes and latent nodes by $\mathcal{W}$ and $\mathcal{Y}$, respsectively. The whole model defines a joint distribution over all observed variables $\boldsymbol{X}$ and latent variables $\boldsymbol{Y}$

$$P(\boldsymbol{x}, \boldsymbol{y}) = \prod_{\boldsymbol{W} \in \mathcal{W}} P(\boldsymbol{w}|\pi(\boldsymbol{W})) \prod_{Y \in \mathcal{Y}} P(y|\pi(Y)). \tag{1}$$

Given a model structure m, the parameters can be estimated by the EM-algorithm [13], which is well-known for estimating parameters of models with latent variables. When the model structure is unknown, a greedy search that aims to maximize a model selection score can be used [29].

The GMMs can be considered as a special case of the PLTMs. This is illustrated by the example GMM depicted in Fig. 1b. In the figure, all the observed variables X_1–X_9 in the GMM are drawn as a pouch node, which has a multivariate normal distribution conditional on its parent latent variable Y_1.

Similar to GMMs, PLTMs can be used for clustering. After training PLTMs on a given data set, the data can be partitioned using each of the latent variables Y. Each data point $\boldsymbol{d}$ can be classified to one of the states of Y by computing the posterior probability $P(y|\boldsymbol{d})$ based on the joint distribution defined by Eq. 1.

3 Clique Tree Propagation

Suppose the values of some variables $\boldsymbol{E} \subseteq \boldsymbol{X}$ are observed in a data sample. *Inference* refers to the computation of the posterior probability $P(\boldsymbol{q}|\boldsymbol{e})$, where $\boldsymbol{q}$ are the values of some variables $\boldsymbol{Q} \subseteq \boldsymbol{X} \cup \boldsymbol{Y}$.

Inference is a core computation task for PLTMs. It is used in the E-step of the EM-algorithm to estimate the values of the unobserved variables. It is also used to compute the cluster assignments after training PLTMs for cluster analysis. The inference task is time-demanding. It is a strenuous task and is recurrently called for each data sample and each model structure during PLTM training. Therefore, it is the first target of optimization for streamlining PLTM training.

Inference can be done on PLTMs similarly as the clique tree propagation (also known as belief propagation or junction tree algorithm) on conditional Gaussian Bayesian networks [20]. We describe the main steps of the inference algorithm and the numerical operations below. Readers are referred to [29] for more details on inference on PLTMs and to [11,12,20] for the general clique tree algorithm.

Construction of Clique Trees. Clique tree propagation requires converting the original model to a structure called clique tree $\mathcal{T}$ to organize the computation. Construction of clique trees is simple due to the tree structure of PLTMs. To construct $\mathcal{T}$, a clique C is added to $\mathcal{T}$ for each edge in M, such that $C = \boldsymbol{V} \cup \{\Pi(\boldsymbol{V})\}$ contains the variable(s) $\boldsymbol{V}$ of the child node and variable $\Pi(\boldsymbol{V})$ of its parent node. A separator node is added for discrete node in the PLTM. It is used to connect the two clique nodes containing the separator variable. The resulting clique tree contains two types of cliques: discrete cliques with at most two discrete variables and mixed cliques with a discrete variable and multiple continuous variables.

Propagation. After a clique tree is constructed, propagation can be carried out on it. The clique tree propagation consists of four main steps: initialization of cliques, incorporation of evidence, message passing, and normalization.

Step 1 initializes the clique tree with the model parameters. The mean vectors and the covariance matrices of a pouch node are copied to its corresponding mixed clique. Similarly, the conditional probability table of a discrete node is copied to the corresponding discrete clique. Note that the root node does not have a corresponding clique. Its marginal probability is multiplied to one of the cliques corresponding to its child variables.

Step 2 incorporates the evidence (observed values) in the potentials. For brevity, here we consider only the case where there is no missing value in the data. Consider a pouch node with variables $\boldsymbol{W}$ and with observed values $\boldsymbol{e}$. Furthermore, denote its parent variable by Y. This step involves computing the probability values $P(y|\,\boldsymbol{W} = \boldsymbol{e}) = \mathcal{N}\left(\boldsymbol{e}|\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y\right)$, where $\mathcal{N}\left(\cdot|\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y\right)$ denote the normal distribution conditional on the value of Y.

Step 3 performs a series of computations, each on a small part of the clique tree, as represented by the process of message passing. Since the clique tree does not contain any loop, exact inference can be performed by message passing in two phases. In the first phase, messages are passed from the leaf clique nodes to the clique corresponding to the root node of the PLTM. We denote that clique as *pivot*. In the second phase, messages are passed from the pivot along the opposite direction back to the leaf nodes.

For the mixed cliques, the messages to be sent from the mixed cliques have already been computed in step 2. The message passing between discrete cliques requires performing multiplication and division between potential tables of two variable and message of one variable. It also requires marginalizing out a variable to compute the message of one variable from a potential table with two variables.

After message passing is completed, the likelihood for a data sample can be determined from the pivot clique. The likelihood is equal to the sum of the potential entries of the pivot clique.

Step 4 normalizes the clique potentials by multiplying each entry by a particular constant. It converts the potential values to proper probability values. This entails dividing each entry of the potential tables by the likelihood value.

Complexity. Let n be the number of nodes in a PLTM, c be the maximum cardinality of a discrete variable, and p be the maximum number of variables in a pouch node. The time complexity of the inference is dominated by the steps related to message passing and incorporation of evidence on continuous variables. The message passing step requires $O(nc^2)$ time, since each clique has at most two discrete variables due to the tree structure. Incorporation of evidence requires $O(ncp^3)$ time. Although the tree structure of PLTMs allows tractable inference, with time complexity linear to the number of nodes, the inference can still needs much time as it has to be performed many times during PLTM training.

Table 1. Data units for performing inference for each data sample. p denotes the number of variables in a pouch node. c and c' denote the cardinalities of the variables of the node and its parent node, respectively, in the PLTM.

Node type in PLTM	Node type in clique tree	Data type	Number of entries
Continuous node	Mixed clique	Mean vector	$p \times c'$
		Covariance matrix	$p^2 \times c'$
Discrete node	Discrete clique	Potential table	$c \times c'$
All node	Separator	Message to parent	c'
		Message to children	c

4 Implementation for GPUs

In this section, we describe how to adapt the inference of PLTMs for running efficiently on GPUs. We refer to the CPU as host and the GPU as device below.

Data Representation. The original implementation of PLTMs[1] used the object-oriented approach to represent the data units as objects. This poses a challenging for GPU programming. Instead, we represent the data units as arrays for easy access by the GPU kernel functions. The data units required for performing inference are shown in Table 1. Each node in a PLTM has a corresponding clique

[1] https://github.com/kmpoon/pltm-east.

node and a separator node in the clique tree.[2] The third column describes the type of data associated with each clique node. The fourth column indicates the number of entries for each type of data.

Device Memory. The data units in Table 1 are used to store interim computation results during inference. Therefore, a clique tree (with cliques and separators) has to be allocated for each data sample. The data units are allocated at the beginning of likelihood computation so that they are available for the inference on multiple data samples in parallel. The memory is allocated in a single batch to minimize the number of API calls. We also allocate an array on the device for storing the likelihood results computed during inference.

Host-Device Memory Transfer. In the first step of inference, the clique tree needs to be initialized by the parameters of the model. We perform initialization on the host and then transfer the array data representing the initialized clique tree to the device. We transfer only the data corresponding to one instance of the clique tree. The data is then copied to the arrays representing the other instances of clique trees for all data samples on the device. This process saves the amount of data needed to be transferred between host and device. It also simplifies the way to transfer the model parameters to the device as the parameter values are now contained in an array rather than in objects. Besides, the data units representing the clique tree, the data matrix is also transferred to device in a single batch at the beginning to minimize the number of transfers.

Parallelism. Most computation of clique tree propagation for PLTMs is done during the two steps for incorporating evidence and message passing. Such computation involves the calculating multiple entries in a target potential or message. Those entries of a potential or message can be calculated in parallel. We refer to this parallelism as element-wise parallelism [36].

However, due to the tree structure of PLTMs, the number of entries of a potential or message is usually small compared to general Bayesian networks. The parallel computation of those entries may not be sufficient to utilize all GPU cores. For example, suppose the latent variables in a PLTM has at most 10 states. Then, the number of entries of a message is at most 10. This is much smaller than the number of cores on a GTX 1080 Ti GPU (3584).

To fully utilize the massive computation power in GPUs, we need to consider other parallelism opportunities. When the likelihood is being computed, the clique tree propagation is performed on each sample independently. Hence, they can be performed in parallel. This form of parallelism is referred to as data parallelism. As a comparison, the number of samples we used in the experiments can be at most 2310. With both data parallelism and element-wise parallelism, the GPU cores can be better utilized.

[2] An exception is that the root node sometimes does not have a separate clique node.

Host-Device Coordination. Compared with CPUs, GPUs can perform parallel computation very efficiently. However, they have fewer programming language support and programming on them can be tedious. Therefore, we program on the CPUs to mainly determine the sequence of computation and to delegate the highly parallelizable tasks to GPUs. For example, message passing is conducted one node at a time since the message computation needs to follow the aforementioned scheme of flow. We use the host to determine the sequence of the message passing and then invokes the kernel calls for computing the messages for each node. The kernel call is run with multiple threads on different data samples and different elements of messages in parallel.

Device-Host Memory Transfer. The computed likelihood values of the data samples are stored in an array on the device after the clique tree propagation. The array is transferred to the host and the values are then multiplied together on the host to obtain the final likelihood value on the whole data set.

It is worthwhile to note that the data structure storing the interim results do not need to be transferred back and forth between the host and the device. This reduces the transfer cost. The same situation also applies for the EM-algorithm. Only the final parameter estimates need to be transferred back to the host. The intermediate values can be kept in the device memory during the different iterations of EM steps. This show one resemblance between the inference procedure and the full PLTM training.

Implementation. We implemented the inference method based on the CUDA framework [30]. We used the Scala language for host programming and the JCuda package[3] as Java bindings for CUDA. The Java ecosystem was used to reduce the coding effort since the original implementation was written in Java.

Table 2. Descriptions of real-world data sets from the UCI repository used in the experiments.

Data set	#Attributes	#Classes	#Samples
`glass`	9	6	214
`image`	18	7	2310
`ionosphere`	33	2	351
`vehicle`	18	4	846
`wdbc`	30	2	569
`wine`	13	3	178
`yeast`	8	10	1484
`zernike`	47	10	2000

[3] http://www.jcuda.org.

5 Experiments

To evaluate the performance of the GPU-accelerated inference method, we use it to compute likelihood of a given PLTM on a given data set. We ran it using real-world data sets in our experiments. We used the same models and data sets in [29] to estimate the actual improvement in practice. Table 2 show the properties of the data sets used. The EAST algorithm [29] were used to train PLTMs on those data sets.

Three different implementations of the clique tree propagation were used in the experiments. The first implementation runs sequentially on CPUs. It serves as a baseline for comparison. The second one performs inference on different data samples in parallel on CPUs. The last one uses the GPUs for acceleration as described previously. The experiments were conducted on a Linux computer with a Xeon E3-1245 v5 CPU and a GeForce GTX 1080 Ti GPU. The CPU has fours cores (eight threads) running at a base frequency of 3.5 GHz. The GPU has 28 streaming multiprocessors with 3584 CUDA cores running with a maximum clock rate of 1.6 GHz.

Table 3. Average elapsed wall time in milliseconds (ms) for computing the likelihood of PLTMs on real-world data sets using different implementations of clique tree propagation, including the sequential version running on CPUs, the parallel version running on CPUs, and the accelerated version running on GPUs. The speedups over the baseline sequential version are shown in parentheses.

	Average running time (ms)		
	CPU-sequential	CPU-parallel	GPU
`glass`	8.62	3.89 (2x)	1.42 (6x)
`ionosphere`	48.37	14.67 (3x)	2.76 (18x)
`image`	1421.30	290.14 (5x)	27.52 (52x)
`vehicle`	169.90	41.54 (4x)	4.89 (35x)
`wdbc`	143.04	37.77 (4x)	3.28 (44x)
`wine`	7.54	2.94 (3x)	1.44 (5x)
`yeast`	50.74	15.06 (3x)	2.07 (25x)
`zernike`	2655.91	576.26 (5x)	97.75 (27x)

We measure the performance of the three implementations using the elapsed wall time for likelihood computation. We report the time averaged over 100 repetitions. The experiments first ran 20 repetitions at the beginning to allow the just-in-time compiler of the Java Runtime to come into force. Those repetitions were not included for time reporting.

Table 3 reports the average running time for one likelihood computation in milliseconds. The results show that the GPU acceleration could achieve 5x to 52x speedups over the baseline sequential version. It also attained 2x to 12x

speedups over the parallel version running on CPUs. The higher speedups over the sequential version were obtained on data sets with larger number of samples (e.g. `image`, `vehicle`, `wdbc`, `yeast`, and `zernike`). This can be explained by the fact that the larger number of samples provide better opportunity for exploiting data parallelism by GPUs.

The considerable speedups shown above has demonstrated that the GPUs can be effective in accelerating the inference task. Since the other computationally intensive tasks in the PLTM training procedure show similar parallelism opportunities as the likelihood computation task, our results signify the promising potential for further improvement on the full training of PLTMs with GPUs.

Table 4. Proportion of GPU activities and overall running time used for the incorporation of evidence routine.

	% of GPU activities	% of Overall time
`glass`	64.81%	3.02%
`ionosphere`	82.46%	8.07%
`image`	98.94%	86.13%
`vehicle`	97.10%	46.26%
`wdbc`	84.92%	15.74%
`wine`	71.70%	3.32%
`yeast`	32.71%	1.49%
`zernike`	99.60%	94.25%

To identify possible bottlenecks of the current GPU implementation, we used the tool `nvprof` provided in the CUDA Toolkits to profile different kernel calls. Table 4 lists the proportion of GPU activities and overall running time spent on the incorporation of evidence routine. We see that this task constituted most of the GPU activities except on the `yeast` data set. It even accounted for 86% and 94% of the overall running time on `image` and `zernike`, respectively. The two data sets happened to take the longest running time.

To understand this phenomenon, recall that the incorporation of evidence routine for a pouch node has a time complexity linear to the cardinality of its parent variable and cubic to the number of the variables in the pouch node. We compare the model properties on four data sets with similar number of attributes in Table 5. We see that the models for `image` and `zernike` both have a large pouch node with 10 and 16 variables, respectively. The problem is exacerbated by the high cardinality of the parent variables of those two pouch nodes. The PLTM for `wdbc` also has a large pouch node with 10 variables. However, that node has a smaller parent cardinality and the model has small pouch size on average. Hence, the incorporation of evidence routine was less significant on `wdbc`. Future study may consider how to tackle this bottleneck to make PLTMs more efficient on larger data sets.

Table 5. Properties of PLTMs on four data sets. The average and maximum number of variables in a pouch node are listed in the second and third columns, and the cardinality of the parent variable of the pouch node with maximum size in the fourth column.

	Average pouch size	Maximum pouch size	Parent cardinality
`ionosphere`	4.6	7	3
`image`	3.6	10	11
`wdbc`	3.0	10	5
`zernike`	7.8	16	6

6 Related Work

Several works used GPUs to speed up the EM-algorithm for parameter estimations [3,19,22]. They exploited the innate data parallelism due to the independent computation for different data samples. However, they considered only GMMs. The inference procedure on GMMs is simpler than PLTMs.

Some works used GPUs for belief propagation on Bayesian networks. Element-wise parallelism and arithmetic parallelism was exploited for inference [36] and a statistical model was further proposed for optimizing the GPU parameters [37]. Another work formulated the inference procedure in terms of operations on sparse matrices [6]. Existing matrix packages utilizing GPU computation (e.g. `PyTorch`) were then used to run the inference. Some studies used better memory layout and better scheduling among memory transfers and works to improve the inference performance on GPUs [5,15]. Some works studied belief propagation on Markov Random Fields used for stereo processing on GPUs [14,17,33]. They considered an approximate inference method that requires passing messages to the same nodes multiple times.

The above methods on inference usually achieve significant speedup only when the potential tables have a large number of entries. However, due to the tree structure in PLTMs, the parallelism exploited by those methods may not be as effective on PLTMs. Besides, our work considered data parallelism that were not available as those methods ran inference on only a single data sample.

PLTMs were proposed as a generalization of the latent tree models (LTMs). The LTMs [34] have discrete observed variables, in contrast to the continuous observed variables in PLTMs. The LTMs have found numerous applications such as density estimation, multidimensional clustering, spectral clustering, and topic modeling [24,35]. Attempts have been made to speed up the training of LTMs. Spectral methods [2] and Progressive EM [9] have been proposed for faster parameter estimation. Heuristics were proposed to guide the structure learning [10,21,32] and Stepwise EM was used to reduce the number of samples involved in computation [8]. Those attempts did not utilize any parallelism for GPUs. On the other hand, their acceleration techniques can possibly be combined with our proposed method to achieve higher speedups.

7 Conclusion

In this paper, we show how to use GPUs to accelerate the clique tree propagation algorithm for PLTMs. We use the likelihood computation task to evaluate the performance of the inference procedure. The experiment results demonstrate that substantial speedups (up to 52x) can be achieved. As the other computationally intensive tasks in the PLTM training procedure show similar parallelism opportunities as the likelihood computation task, our results signify promising potential for further improvement on the full training of PLTMs with GPUs. The GPU acceleration techniques discussed in this paper can be crucial in applying PTLMs on massive data sets.

Acknowledgement. Research on this article was supported by the Education University of Hong Kong under grant RG70/2017-1018R, the Top-up Fund of Dean's Research Fund, and the Small Research Grant of the Department of Mathematics and Information Technology.

References

1. Al-Ayyoub, M., Abu-Dalo, A.M., Jararweh, Y., Jarrah, M., Sa'd, M.A.: A GPU-based implementations of the fuzzy C-means algorithms for medical image segmentation. J. Supercomput. **71**(8), 3149–3162 (2015)
2. Anandkumar, A., Chaudhuri, K., Hsu, D., Kakade, S.M., Song, L., Zhang, T.: Spectral methods for learning multivariate latent tree structure. In: Advances in Neural Information Processing Systems, vol. 24, pp. 2025–2033 (2012)
3. Araújo, G.F., Macedo, H.T., Chella, M.T., Estombelo Montesco, C.A., Medeiros, M.V.O.: Parallel implementation of expectation-maximisation algorithm for the training of Gaussian mixture models. J. Comput. Sci. **10**(10), 2124–2134 (2014)
4. Arefin, A.S., Riveros, C., Berretta, R., Moscato, P.: kNN-MST-agglomerative: a fast and scalable graph-based data clustering approach on GPU. In: 7th International Conference on Computer Science Education, pp. 585–590 (2012)
5. Bistaffa, F., Bombieri, N., Farinelli, A.: An efficient approach for accelerating bucket elimination on GPUs. IEEE Trans. Cybern. **47**(11), 3967–3979 (2017)
6. Bixler, R.M.: Sparse matrix belief propagation. Master's thesis, Virginia Tech (2018)
7. Bouveyrona, C., Brunet-Saumard, C.: Model-based clustering of high-dimensional data: a review. Comput. Stat. Data Anal. **71**, 52–78 (2014)
8. Chen, P., Zhang, N.L., Liu, T., Poon, L.K.M., Chen, Z., Khawar, F.: Latent tree models for hierarchical topic detection. Artif. Intell. **250**, 105–124 (2017)
9. Chen, P., Zhang, N.L., Poon, L.K.M., Chen, Z.: Progressive EM for latent tree models and hierarchical topic detection. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 1498–1504 (2016)
10. Choi, M.J., Tan, V.Y.F., Anandkumar, A., Willsky, A.S.: Learning latent tree graphical models. J. Mach. Learn. Res. **12**, 1771–1812 (2011)
11. Cowell, R.G., Dawid, A.P., Lauritzen, S.L., Spiegelhalter, D.J.: Probabilistic Networks and Expert Systems. Springer, New York (1999). https://doi.org/10.1007/b97670

12. Darwiche, A.: Modeling and Reasoning with Bayesian Networks. Cambridge University Press, Cambridge (2009)
13. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. R. Stat. Soc. Ser. B (Methodol.) **39**(1), 1–38 (1977)
14. Eslami, H., Kasampalis, T., Kotsifakou, M.: A GPU implementation of tiled belief propagation on markov random fields. In: Eleventh ACM/IEEE International Conference on Formal Methods and Models for Codesign, pp. 143–146 (2013)
15. Fioretto, F., Pontelli, E., Yeoh, W., Dechter, R.: Accelerating exact and approximate inference for (distributed) discrete optimization with GPUs. Constraints **23**(1), 1–43 (2018)
16. Grauer-Gray, S., Kambhamettu, C., Palaniappan, K.: GPU implementation of belief propagation using CUDA for cloud tracking and reconstruction. In: IAPR Workshop on Pattern Recognition in Remote Sensing, pp. 1–4 (2008)
17. Grauer-Gray, S., Cavazos, J.: Optimizing and auto-tuning belief propagation on the GPU. In: Cooper, K., Mellor-Crummey, J., Sarkar, V. (eds.) LCPC 2010. LNCS, vol. 6548, pp. 121–135. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19595-2_9
18. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Comput. Surv. **31**(3), 264–323 (1999)
19. Kumar, N.S.L.P., Satoor, S., Buck, I.: Fast parallel expectation maximization for Gaussian mixture models on GPUs using CUDA. In: 11th IEEE International Conference on High Performance Computing and Communications, pp. 103–109 (2009)
20. Lauritzen, S.L., Jensen, F.: Stable local computation with conditional Gaussian distributions. Stat. Comput. **11**, 191–203 (2001)
21. Liu, T.F., Zhang, N.L., Chen, P., Liu, A.H., Poon, L.K.M., Wang, Y.: Greedy learning of latent tree models for multidimensional clustering. Mach. Learn. **98**(1–2), 301–330 (2015)
22. Machlica, L., Vanek, J., Zajic, Z.: Fast estimation of Gaussian mixture model parameters on GPU using CUDA. In: 12th International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 167–172 (2011)
23. McLachlan, G.J., Peel, D.: Finite Mixture Models. Wiley, New York (2000)
24. Mourad, R., Sinoquet, C., Zhang, N.L., Liu, T., Leray, P.: A survey on latent tree models and applications. J. Artif. Intell. Res. **47**(1), 157–203 (2013)
25. Pangborn, A.D.: Scalable data clustering using GPUs. Ph.D. thesis, Rochester Institute of Technology (2010)
26. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers, San Mateo (1988)
27. Poon, L.K.M.: Clustering with multidimensional mixture models: analysis on world development indicators. In: Cong, F., Leung, A., Wei, Q. (eds.) ISNN 2017. LNCS, vol. 10261, pp. 153–160. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59072-1_19
28. Poon, L.K.M., Zhang, N.L., Chen, T., Wang, Y.: Variable selection in model-based clustering: to do or to facilitate. In: Proceedings of the 27th International Conference on Machine Learning, pp. 887–894 (2010)
29. Poon, L.K.M., Zhang, N.L., Liu, T., Liu, A.H.: Model-based clustering of high-dimensional data: variable selection versus facet determination. Int. J. Approx. Reason. **54**(1), 196–215 (2013)
30. Sanders, J., Kandrot, E.: CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional, Boston (2010)

31. Shalom, S.A.A., Dash, M., Tue, M.: Graphics hardware based efficient and scalable fuzzy c-means clustering. In: Proceedings of the 7th Australasian Data Mining Conference, vol. 87, pp. 179–186 (2008)
32. Wang, Y., Zhang, N.L., Chen, T.: Latent tree models and approximate inference in Bayesian networks. J. Artif. Intell. Res. **32**, 879–900 (2008)
33. Xu, Y., Chen, H., Klette, R., Liu, J., Vaudrey, T.: Belief propagation implementation using CUDA on an NVIDIA GTX 280. In: Nicholson, A., Li, X. (eds.) AI 2009. LNCS (LNAI), vol. 5866, pp. 180–189. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10439-8_19
34. Zhang, N.L.: Hierarchical latent class models for cluster analysis. J. Mach. Learn. Res. **5**, 697–723 (2004)
35. Zhang, N.L., Poon, L.K.M.: Latent tree analysis. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, pp. 4891–4897 (2017)
36. Zheng, L., Mengshoel, O.: Exploring multiple dimensions of parallelism in junction tree message passing. In: Proceedings of the 2013 UAI Application Workshops: Big Data meet Complex Models and Models for Spatial, Temporal and Network Data (2013)
37. Zheng, L., Mengshoel, O.: Optimizing parallel belief propagation in junction treesusing regression. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 757–765 (2013)