Approximate Query Answering in Complex Gaussian Mixture Models

Mattis Hartwig
Institute of Information Systems
University of Lübeck
Lübeck, Germany
hartwig@ifis.uni-luebeck.de

Marcel Gehrke
Institute of Information Systems
University of Lübeck
Lübeck, Germany
gehrke@ifis.uni-luebeck.de

Ralf Möller
Institute of Information Systems
University of Lübeck
Lübeck, Germany
moeller@ifis.uni-luebeck.de

***Abstract*—Gaussian mixture models are widely used in a diverse range of research fields. If the number of components and dimensions grow high, the computational costs for answering queries become unreasonably high for practical use. Therefore approximation approaches are necessary to make complex Gaussian mixture models more usable. The need for approximation approaches is also driven by the relatively recent representations that theoretically allow unlimited number of mixture components (e.g. nonparametric Bayesian networks or infinite mixture models). In this paper we introduce an approximate inference algorithm that splits the existing algorithm for query answering into two steps and uses the knowledge from the first step to reduce unnecessary calculations in the second step while maintaining a defined error bound. In highly complex mixture models we observed significant time savings even with low error bounds.**

***Index Terms*—Gaussian mixture models, query answering, conditional probability, inference, approximation**

I. INTRODUCTION

Gaussian Mixture models (GMMs) have been used very widely for modeling complex probabilistic distributions across diverse fields including agriculture, medicine, bioinformatics, etc. [1]. If the dimensions of the probability distribution and number of mixture components are large, the computational costs grow. In nonparametric approaches the number of components is not limited before learning the model [2] allowing the mixture to grow in its number of components dependent on the data that is fed into the model. Examples for these models are infinite GMMs [2] and nonparametric Bayesian networks [3], [4]. These nonparametric models have been used more widely recently and several use cases from different fields have been identified [3]. Existing query answering algorithms that work along the semantics of the query language just perform calculations for all mixture models separately which works very well for models with low dimensionality and low number of components [5]. But if the model complexity grows the computational costs increase rapidly.

In this paper we describe an approach to speed-up the query answering in GMMs with focus on calculating conditional probability distributions. The approach uses the fact that the posterior weights of mixture components can be calculated with relatively little costs and combines this with a modified version of a quick-select algorithm to identify the most important mixture components for the posterior distribution. The relatively costly step of calculating a posterior distribution for each mixture is only performed on the priority components, which significantly reduces the overall runtime. In situations with a high numbers of components and dimensions the approximate approach is up to 40 times faster while maintaining a low error bound.

The remainder of this paper has following structure. We begin by introducing GMMs and explain how highly complex mixture models can emerge. After defining the query language we explain how the queries can be answered in GMMs naively. Afterwards we introduce our approximate approach followed by a detailed complexity comparison and an evaluation. Our work is concluded by an evaluation and an outlook to future work.

II. PRELIMINARIES

This section introduces GMMs and explains how highly complex GMMs can emerge. For further information on mixture models or Gaussian distribution see for example [6].

A. Gaussian Mixture Models

The distribution of a single Gaussian random Variable X_i follows the probability density function:

$$f_{x_i}(x) = \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp\left(-\frac{1}{2\sigma^2}(x_i - \mu)^2\right) \tag{1}$$

where μ is the mean and σ^2 the standard deviation. We write more simply

$$X_i \sim N\left(\mu, \sigma^2\right) \tag{2}$$

The D-dimensional multivariate Gaussian distribution over the set of d random variables X follows the multivariate probability density function:

$$f(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right) \tag{3}$$

with μ being a D-dimensional mean vector and Σ being a $D \times D$-dimensional covariance matrix. We write more simply again

$$X \sim N(\mu, \Sigma) \tag{4}$$

Gaussian distributions are very easy to use for calculations because they are closed under multiplication and the integral can be calculated analytically. Both characteristics are useful for working with probabilities. Unfortunately only very few phenomena can be represented well enough with a pure Gaussian distribution. To keep some of the nice features of Gaussian distributions and gain flexibility in the representation, mixtures of Gaussians have been developed. A mixture of multivariate Gaussians can approximate any density function [7].

A mixture model M is defined as a set of K probability distributions$P(X|\theta_k)$ and K corresponding weights $w(k)$:

$$M = \left(\{P(X|\theta_k)\}_1^K, \{w(k)\}_1^K\right) \tag{5}$$

where θ_k includes the parameters of the corresponding probability distribution, $w(k) \geq 0$ and

$$\sum_{k=1}^{K} w(k) = 1. \tag{6}$$

Each pair of probability distribution and weight is called a mixture component or just component. A mixture model describes a joint probability distribution of the following form:

$$P(X) = \sum_{k=1}^{K} w(k) P(X|\theta_k) \tag{7}$$

Each component can be interpreted as an expert whose weight is a measure for his relative ”expertise” [8]. Since the weights sum-up to one the weights basically form a discrete probability distribution over the parameters. A mixture model is called a GMM if all distributions $P(X|\theta_k)$ are Gaussians (3). In a GMM each parameter set θ_k contains a μ_k and a σ_k.

B. Motivating High Complexity Mixtures

The complexity of GMMs is driven by the number of dimensions and the number of components. The number of components in a mixture model is often fixed but nonparametric models are used to make the number of components dependent on the amount of data that is fed into the model resulting in possibly very high number of components [2]. One example for high dimensions in models are medical diagnosis networks. The famous Computer-based Patient Case Simulation probabilistic model (CPCS-PM) has 422 dimensions and the Quick Medical Reference Bayesian Network (QMR-BN) even has 4040 dimesions [9], [10]. Some types of these network models can be reduced to GMMs, which makes our work applicable in these fields as well [11]. Example types that can be reduced to GMMs are Mixtures of Gaussian Bayesian networks or nonparametric Bayesian networks [4]. In nonparametric Bayesian networks both high number of dimensions and high number of components can emerge.

In general a high number of mixtures which can be interpreted as experts in their field allow for very specific ”expert opinions” in the model. Query answering in these mixtures can be interpreted as asking all experts based on a certain evidence. Since the knowledge of the experts depends on the evidence, for different evidences we should ask different experts (having a broken leg no one would ask a dentist). An approximate algorithm that does not wait for the full answer of every expert but can anticipate which expert's opinions are needed for the specific evidence is a contribution to make highly complex GMMs more usable.

III. Query Language

At the beginning of this section we describe the syntax and semantic of our query language. Afterwards we explain why a query answering algorithm that naively follows the semantic is not efficient.

A. Query Syntax

A query on a GMM M contains a set of queried variables $X^Q \subseteq X$ and a set of measured variables $X^E = x^E$ where $X^E \subseteq X \setminus X^Q$. The syntax for a query is on a GMM M is $P_M(X^Q|X^E = x^E)$. For brevity, we typically leave out the M and write x^E without the equation. The resulting probability distribution of an answered query is still represented by a GMM. It is called posterior or conditional probability distribution. If X^E is an empty set the result is called marginal distribution.

B. Query Semantic

The semantic behind the query is a sum of all conditional distributions of M:

$$P(X^Q|x^E) = \sum_{k=1}^{K} w(k|x^E) P(X^Q|x^E, \theta_k) \tag{8}$$

Calculating the conditional distribution $P(X^Q|x^E, \theta_k)$ of a multivariate Gaussian follows some basic rules, for a proof see [6].

$$X^Q|x^E \sim N(\mu_{Q|E}; \Sigma_{Q|E}) \tag{9}$$

where

$$\mu_{Q|E} = \mu_Q + \Sigma_{Q,E}\Sigma_{E,E}^{-1}(x^E - \mu_E) \tag{10}$$

$$\Sigma_{Q|E} = \Sigma_{Q,Q} - \Sigma_{Q,E}\Sigma_{E,E}^{-1}\Sigma_{E,Q} \tag{11}$$

According to the Bayes rule, to calculate the posterior weight $w(k|x^E)$ we calculate the likelihood given the observation multiplied with the prior and divided by the overall likelihood of observing x^E:

$$w(k|x^E) = \frac{w(k)P(x^E|k)}{P(x^E)} \tag{12}$$

The fact that not only the component posterior distributions change but also the weights of posterior mixture change can be a counter intuitive. The intuition behind is that each component has an area where it is knowledgeable. It is also called a mixture of experts [8]. The evidence therefore has an impact on the probability of a component to be an ”expert” for that case.

Algorithm 1 Recursive select top-L elements function

```
1:  procedure SELECTCOMPONENTS($list, errorBound, sC$)   ▷ sC list used to save the selected components
2:      $size \leftarrow list.size$
3:      $pivot \leftarrow$ PARTITION($list, 0, size - 1$)   ▷ partitions the list and returns the pivot element
4:      $partSum \leftarrow$SUM($list[pivot : size]$)   ▷ sums the right part of the partitioned list
5:      if $partSum < errorBound$ then
6:          ADD($sC, list[pivot : size]$)   ▷ adds the right part to the selected components
7:          $newBound \leftarrow errorBound - partSum$
8:          SELECTCOMPONENTS($list[0 : pivot], newBound, sC$)   ▷ recursive call on left partition with new bound
9:      else
10:         if $pivot \neq 0$ then
11:             SELECTCOMPONENTS($list[pivot : size], errorBound, sC$)   ▷ recursive call on right partition with old bound
12:         else
13:             ADD($sC, list[pivot : size]$)   ▷ adds the final component and terminates
```

C. Query Answering

As apparent in (11) the calculation of the conditional covariance matrix involves matrix inversion and matrix multiplication. Both computations are polynomial and thus are driving the complexity of the query answering algorithm [12]. A naive query answering along the semantic just repeats the inference steps for each mixture component. The result is that execution time grows linearly with the number of components and polynomially with the number of dimensions which is generating a need for faster approximate approaches. We will revisit the runtime complexity of the naive approach as a baseline in the complexity analysis in Section 5.

IV. APPROXIMATION APPROACH

The idea of our approximation approach to improve the query answering runtime is to divide the calculation in (8) into two steps. First, we calculate the posterior weights for all components based on (12). Second, we use these posterior weights to prioritize components based on an allowed error bound and only calculate the conditional distribution based on (9), (10) and (11) for the prioritized components. As an intuition we can think of our first step as an approximation of the expert knowledge given some evidence. In the second step we "only ask experts and wait for their answers if they contribute enough to the final answer". In this section we first detail out our concrete algorithm and show that a predefined error bound is met.

A. Detailed Algorithm

We are given a GMM with K components and D dimensions. The random variables are split into queried variables X^Q and measured variables X^E. A given error bound ϵ specifies the desired accuracy of the approximation.

1) Step 1: Calculate posterior weights: To calculate the posterior weights, the evidence x^E and the priors $w(k)$ are put in (12). This calculation is performed for all components and the result is an unsorted list that contains all posterior weights. Because of the conjugate nature of the weight distribution, all the posterior weights still sum-up to one, see (6).

2) Step 2: Component selection: The goal is to select a subset K_L of all mixture components such that the difference between the conditional probability distribution of of the subset K_L (the approximation) and the exact conditional probability distribution of the whole mixture (8) is smaller than the error bound ϵ in any point.

$$\sum_{k\in\{1,\ldots,K\}} w(k|x^E)P(X^Q|x^E,\theta_k) - \sum_{k\in K_L} w(k|x^E)P(X^Q|x^E,\theta_k) \leq \epsilon \quad (13)$$

To save runtime the size of subset should be as small as possible. Unfortunately a minimization of the size of K_L would involve the computation of the conditional probability distributions of all components which would result in the exact solution anyhow. To save runtime we developed a method that selects a subset K_L that fulfills (13) without spending too much computations. Our algorithm selects the top-L components whose sum of weights is at least $1 - \epsilon$. We detail-out our algorithm first and describe in the next section why the error bound condition (13) holds. A naive approach would sort the whole list and then would select the top-L components until their sum reaches $1 - \epsilon$ but would need on average $O(Klog(K))$ time. Our approach is similar to the Quick-Selection algorithm [13] and also has $O(K)$ average runtime (for pseudocode of the recursive procedure see Algorithm 1). One element of the unsorted list is selected as a pivot element that is used to partition the list. Afterwards the right (bigger) part is summed-up. If this sum is smaller than the needed accuracy, the complete right part is selected and only the left part needs to be checked further. For the left part the same partitioning is done again and the SelectionFunction in called recursively with a new accuracy decreased by the sum of the right part. If the sum of the right part is bigger, the left part can be ignored and the right part need to be checked further. The recursive process starts again but this time with an unchanged needed accuracy because no elements have been selected yet. The quick-select algorithm has an average runtime of $O(K)$ as long as the pivot is not systematically chosen badly because then each partition step reduces the list size by a fraction which results in linear runtime [13]. The number of element swaps in the partitioning process is smaller or equal to the number of components in the larger partition. The same is true for the additional summing-up that is needed by our selection algorithm. Since both have the same upper bound we keep the $O(K)$.

3) Step 3: Completing the posterior: In the last step the posterior probability distribution of each of the L selected mixture components is calculated based on the (8), (10) and (11). The new posterior mixture with L components represents the answer of the conditional probability query.

The next section discusses the error bound in more detail and explains why it holds.

B. Error Bound

We need to show that our approximation fulfills the defined error bound (13). We can shorten (13) by substracting all components in the selected set K_L which leaves us with components that are not selected:

$$\sum_{k\in\{1,\ldots,K\}\setminus K_L} w(k|x^E)P(X^Q|x^E,\theta_k) \leq \epsilon \tag{14}$$

We know that the sum of all the component weights is equal to one (6) and (12). The sum of the weights of the selected components is bigger than $1-\epsilon$ resulting in the sum of the ignored component weights is lower than ϵ:

$$\sum_{k\in\{1,\ldots,K\}\setminus K_L} w(k|x^E) \leq \epsilon \tag{15}$$

Because $P(X^Q|x^E,\theta_k)$ is a probability function, its value is always lower or equal to one resulting in

$$w(k|x^E)P(X^Q|x^E,\theta_k) \leq w(k|x^E). \tag{16}$$

Combining 14, (15) and (16) results in:

$$\sum_{k\in\{1,\ldots,K\}\setminus K_L} w(k|x^E)P(X^Q|x^E,\theta_k) \leq \sum_{k\in\{1,\ldots,K\}\setminus K_L} w(k|x^E) \leq \epsilon \tag{17}$$

This shows us that (13) and (14) are always true if we select the components accordingly to our algorithm which means that we stay in the defined error bound ϵ.

V. COMPLEXITY

We define

- D as the number of random variables X
- C as the number of measured variables X^E
- K as the number of mixture components
- L as the number of selected components by the approximation algorithm
- $\frac{L}{K}=\lambda$ as the component utilization factor ($\lambda=1$ means that all components need to be used in the posterior to meet the error bound)

In the naive implementation we have to calculate K times a posterior weight and K times a posterior probability. For the calculation of the posterior weights matrix two multiplications of two matrices with the dimensions $C\times C$ and an inversion of a $C\times C$ have to be performed (see (12) and (3)). The runtime of both operation grows polynomial and the function that describes all necessary operations is in $\Omega(C^2)$ and $O(C^3)$ resulting in $O(KC^3)$ complexity for calculating the posterior weights for all components [12]. To calculate the posterior probability distribution for each component based on (10) and (11) we need to perform a $C\times C$ matrix inversion and a matrix multiplication involving two $(D-C)\times C$ matrices (one being transposed) which results in a function in $O(K(D-C)^3)$ to describe all necessary operations. In a realistic setup it is often $D\gg C$, resulting in a domination by $O\left(K(D)^3\right)$.

In the approximate algorithm the first step is identical and therefore in the complexity of $O(KC^3)$. The second step generates additional runtime. As described in Section 4.1 selecting the top L components that meet our error bound can be done in $O(K)$. The function describing the last step is in $O(\lambda K(D-C)^3)$ because only L of the K components need to be used. Comparing the two approaches shows that we have runtime savings if:

$$K+\lambda K(D-C)^3 \leq K(D-C)^3 \tag{18}$$

$$1+\lambda(D-C)^3 \leq (D-C)^3 \tag{19}$$

$$\frac{1}{(D-C)^3}+\lambda \leq 1 \tag{20}$$

$$\lambda \leq 1-\frac{1}{(D-C)^3} \tag{21}$$

Equation (21) shows that the utilization factor λ can be nearly one for high $D-C$. This means that we even get a runtime improvement if we use most of the components and the number of dimensions is not too small. For complex mixture structures used in practice this will be always true. In the next section we look at the actual time savings of the approximation approach.

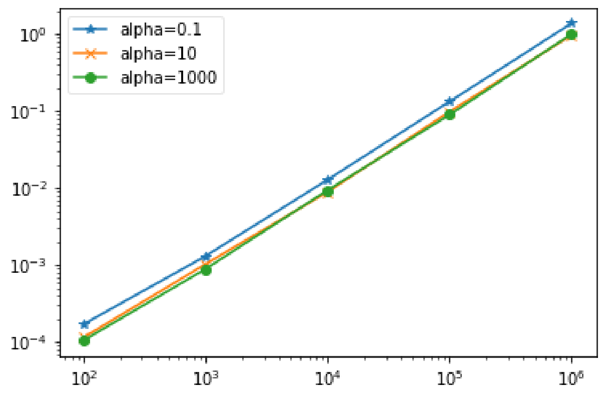

Fig. 1. Runtime [s] (y-axis) for different list sizes [#] (x-axis)

TABLE I
NAIVE ALGORITHM - AVERAGE RUNTIME OF 10 RUNS EACH IN SECONDS (ERROR BOUND CHOSEN TO BE 0.01)

Components	**Dimensions**			
	10	50	250	1250
10	0.003	0.003	0.008	0.369
50	0.014	0.013	0.035	1.778
250	0.064	0.066	0.166	8.847
1250	0.315	0.337	0.836	869.411

TABLE II
APPROX. ALGORITHM - AVERAGE RUNTIME OF 10 RUNS EACH IN SECONDS (ERROR BOUND CHOSEN TO BE 0.01)

Components	**Dimensions**			
	10	50	250	1250
10	0.002	0.002	0.003	0.066
50	0.007	0.007	0.010	0.120
250	0.032	0.033	0.047	0.619
1250	0.161	0.165	0.231	20.740

TABLE III
AVERAGE TIME SAVING FACTOR OF APPROX. VS NAIVE ALGORITHM (ERROR BOUND CHOSEN TO BE 0.01)

Components	**Dimensions**			
	10	50	250	1250
10	1.7	1.4	2.4	5.6
50	2.1	1.9	3.4	14.9
250	2.0	2.0	3.5	14.3
1250	2.0	2.0	3.6	41.9

TABLE IV
AVERAGE TIME SAVING FACTOR OF APPROX. VS NAIVE ALGORITHM (ERROR BOUND CHOSEN TO BE 0.05)

Components	**Dimensions**			
	10	50	250	1250
10	2.1	1.2	1.9	6.7
50	1.9	2.0	3.3	12.8
250	1.9	2.1	3.5	19.6
1250	1.9	2.1	3.7	34.6

TABLE V
AVERAGE TIME SAVING FACTOR OF APPROX. VS NAIVE ALGORITHM (ERROR BOUND CHOSEN TO BE 0.1)

Components	**Dimensions**			
	10	50	250	1250
10	1.5	1.6	2.5	7.3
50	1.9	2.0	3.5	13.9
250	1.9	2.1	3.7	18.0
1250	2.0	2.1	3.7	36.2

TABLE VI
AVERAGE NUMBER OF USED COMPONENTS (ERROR BOUND CHOSEN TO BE 0.1)

Components	**Dimensions**			
	10	50	250	1250
10	1.2	2	1	1.1
50	1.8	1.7	1.9	1.6
250	5.3	4.2	5.1	3.1
1250	15.6	16.7	16.8	19.3

TABLE VII
AVERAGE NUMBER OF USED COMPONENTS (ERROR BOUND CHOSEN TO BE 0.01)

Components	**Dimensions**			
	10	50	250	1250
10	1.1	2.2	2.3	1.2
50	2.8	2.6	2.4	2.8
250	7.8	7.1	7.4	9.1
1250	32	36.3	35.2	34

TABLE VIII
AVERAGE NUMBER OF USED COMPONENTS (ERROR BOUND CHOSEN TO BE 0.001)

Components	**Dimensions**			
	10	50	250	1250
10	2.2	2.2	2.8	2.4
50	2.5	2.5	2.5	3.3
250	9.4	10.8	11.8	12
1250	43.4	49.8	54.2	53.2

VI. EVALUATION

Two evaluations have been performed. Evaluation 1 focuses on the linear runtime of the top-L component selection and Evaluation 2 focuses on the time savings of the whole approximation algorithm compared to the naive implementation.

A. Evaluation 1: Top-L Selection

We evaluate the top-L algorithm with weight lists of different sizes. To get realistic test data, the lists are generated by a Dirichlet process, a process that is also often used to generate mixture models [14]. The evaluation is performed with different values for the α-parameter of the Dirichlet process. The α-parameter is responsible for the layout of the distribution. A high α-parameter results in a more peaky distribution which means that few components have high weights and others small weights. For each combination of number of components and α-parameter we perform 200 runs and take the average runtime.

Fig. 1 shows the linear behavior of our selection algorithm independently of the chosen α.

B. Evaluation 2: Time Savings

To evaluate the actual time savings compared to the naive implementation we check the time saving dependent on the number of mixture components and number of dimensions. We do this check for three different error bounds. Our starting mixture has a uniform weight distribution where the mean-vectors and covariance matrices are randomly generated.

Tables I–II show the average runtime measurements for the naive and approximate algorithm, respectively. Tables III–V show that the runtime saving factor ($\frac{\text{runtime naive}}{\text{runtime approx.}}$) increase with number of components and number of dimensions. In the high complexity case in our evaluation, the approximate algorithm performs up to 40 times than the naive implementation. That factor is high enough to make mixture models applicable in situations where they otherwise would lead to unreasonably high computational costs.

Tables III–V have different error bounds. With some irregularities (potentially connected to processes running in the background) we can see a tendency that a more relaxed error bound leads to more runtime savings.

C. Evaluation 3: Used components

The third evaluation investigates the number of used components in the approximate approach and should be tightly connected to the time savings because the run-time is driven by the number of components for with the posterior distribution needs to be calculated. Tables VI–VIII contain the average number of used components for different error bounds and show that for more restrictive error bounds the number of used components becomes higher. Logically the approach needs also more components if the underlying model has more components. Similar to the time-savings the number of dimensions also have a positive effect on the number of components but much less than the number of components. In general the number of used components for the restrictive error bound in Table VIII is around 5%.

VII. Conclusion

We contribute to the area of GMMs with a new approximate query answering algorithm for conditional probability distributions. In our theoretical complexity analysis we have proven that the additional costs of selecting the top-L mixture components are easily out-leveled by the time-savings of calculating less conditional probability distributions. Using randomly generated test mixtures we have shown that we can realize significant time savings in high complexity setups. These high complexity setups will become more frequent when concepts such as nonparametric Bayesian networks or infinite mixture models are more widely used. At the same time our approximate approach offers runtime savings that can make high complex GMMs applicable in situations where the calculation of the exact solution is currently unreasonable.

Looking forward we will transfer our approach to other mixture models (e.g. mixtures of Gaussian Processes) and we will further evaluate the algorithm in real applications.

References

[1] G. J. McLachlan, S. X. Lee, and S. I. Rathnayake, "Finite mixture models," *Annual review of statistics and its application*, vol. 6, pp. 355–378, 2019.

[2] C. E. Rasmussen, "The infinite Gaussian mixture model," in *Advances in neural information processing systems*, 2000, pp. 554–560.

[3] A. Hanea, O. M. Napoles, and D. Ababei, "Non-parametric Bayesian networks: Improving theory and reviewing applications," *Reliability Engineering & System Safety*, vol. 144, pp. 265–284, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832015002331

[4] K. Ickstadt, B. Bornkamp, M. Grzegorczyk, J. Wieczorek, M. R. Sheriff, H. E. Grecco, and E. Zamir, "Nonparametric Bayesian Networks," *Bayesian Stat*, vol. 9, p. 283, 2011.

[5] J. P. Petersen and O. Winther, "Gaussian Mixture Models for Analyzing Operational Ship Data," 2011.

[6] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[7] D. Alspach and H. Sorenson, "Nonlinear Bayesian estimation using Gaussian sum approximations," *IEEE Transactions on Automatic Control*, vol. 17, no. 4, pp. 439–448, aug 1972.

[8] I. C. Gormley and S. Frühwirth-Schnatter, "Mixtures of Experts Models," *arXiv preprint arXiv:1806.08200*, 2018.

[9] M. Pradhan, G. Provan, B. Middleton, and M. Henrion, "Knowledge engineering for large belief networks," in *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1994, pp. 484–490.

[10] M. A. Shwe, B. Middleton, D. E. Heckerman, M. Henrion, E. J. Horvitz, H. P. Lehmann, and G. F. Cooper, "Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base," *Methods of information in Medicine*, vol. 30, no. 04, pp. 241–255, 1991.

[11] R. D. Shachter and C. R. Kenley, "Gaussian Influence Diagrams," *Management Science*, vol. 35, no. 5, pp. 527–550, 1989. [Online]. Available: https://doi.org/10.1287/mnsc.35.5.527

[12] S. Boyd and L. Vandenberghe, *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares*. Cambridge university press, 2018.

[13] C. A. R. Hoare, "Algorithm 65: Find," *Commun. ACM*, vol. 4, no. 7, pp. 321–322, 1961. [Online]. Available: http://doi.acm.org/10.1145/366622.366647

[14] D. Görür and C. E. Rasmussen, "Dirichlet process gaussian mixture models: Choice of the base distribution," *Journal of Computer Science and Technology*, vol. 25, no. 4, pp. 653–664, 2010.