Improving the Performance and Understanding of the Expectation Maximization Algorithm: Evolutionary and Visualization Methods

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Priya Krishnan Sundararajan

B.Tech., Information Technology, Anna University
M.S., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

August, 2016

Abstract

The Expectation Maximization (EM) algorithm is a method for learning the parameters of probabilistic graphical models when there is hidden or missing data. The goal of an EM algorithm is to estimate a set of parameters that maximizes the likelihood of the data. In spite of its success in practice, the EM algorithm has several limitations, including slow convergence, computational complexity, and inability to escape local maxima. Using multiple random starting points is a popular approach to mitigate the local maxima problem, but this method is time consuming. This work seeks to improve the understanding and performance of the EM algorithm.

We combine evolutionary algorithms, which make use of stochastic search, with the multiple random starting points strategy for the EM algorithm. First, we propose a genetic algorithm for expectation maximization (GAEM), where we combine the global search property of genetic algorithms (GAs) and the local search property of EM. We investigate how different choices of population sizes, crossover and mutation probabilities, and selection techniques affect the solution quality. We found that small population sizes are sufficient to produce high solution quality and considerable speed-up compared to the traditional EM algorithm.

Second, we develop an age-layered EM algorithm (ALEM), where we incorporate an age-layered population structure heuristic in which age is the number of iterations of an EM run. We focus on speeding up the EM algorithm for Bayesian networks. ALEM enables comparisons between similarly aged EM runs and discards less promising EM runs well before their convergence. Experimentally, we find that ALEM can significantly reduce the average number of iterations with no or minimal degradation in solution quality.

Finally, we introduce an intuitive graphical user interface (GUI) to visualize and analyze graphs including Bayesian networks. In particular, the user can perform multi-focus zooming wherein he or she can compare multiple nodes in an overview graphical window and study their parameters in detail windows. For EM learning, this GUI helps to understand the progress of the estimated probability parameters.

Acknowledgments

I would like to express my sincere gratitude to my advisor Professor Ole Mengshoel, who worked hard with me throughout my PhD giving his inspiration and guidance, without which this thesis can never be possible. During our weekly meetings, he would patiently listen, motivate and come up with new ideas whenever I was at stuck. I have always felt more confident after every weekly meeting. I have learnt and benefited from his perfection in writing research papers. I am grateful for the research freedom he gave me on developing my expertise.

I would also like to thank Dr. Ted Selker for his support and motivation. I am very grateful to my thesis committee members - Professor Ole J. Mengshoel (advisor and chair of the committee), Professor Jason Lohn and Professor Joy Zhang for introducing me to the field machine learning and evolutionary computing techniques, and Dr. Nimish Radia for giving me the industry exposure to understand the application of these techniques in various projects.

I would like to thank my colleagues and friends - Lu Zheng, Brian Ricks, Zheng Sun, Aniruddha Basak, Dongzhen Piao, and Bing Liu for the warm and friendly lab atmosphere. Special thanks to Irina Brinster who has always been there to willing to offer help. I would also like to thank Briana Johnson and Lydian Lee with whom I have enjoyed working on different projects.

This work would not have been possible without the support of my family. My parents have been very supportive and continuously inspired me in my research. They took care of my kid so that I can spare more time for research. I also like to thank my brother for his insightful comments during our discussions. And most of all, I would like to thank my husband for supporting me, putting up with me during the weekends when I had work, patiently listening to my work and for giving me the motivation to finish the thesis.

Finally, this work was supported by National Science Foundation grants CCF-0937044, ECCS-0931978, National Aeronautics and Space Administration grants NNX15AR29A, and Ericsson. I greatly appreciate the financial support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Probabilistic Graphical Models and Machine Learning

Many real world systems have uncertain behavior. A prediction model typically will be more accurate if it takes into account the uncertain behavior of these systems. Probabilistic graphical models (PGMs) are used for this purpose. PGMs combines probability theory and graph theory. Probability theory studies the uncertain events and their effects on each other. For example, a set of random variables can be used to model a weather prediction system. Let us consider two random variables, E represent a weather-related event and W represent the weather. We wish to reason about the probability of occurrence of rain ($E = rain$) given the cloudy weather ($W = cloudy$). Probabilistic theory helps to answer such questions: $P = P(E = rain \mid W = cloudy)$. This relation shows the posterior probability of event random variable E conditioned on weather random variable W. However, a real weather prediction system may be a complicated system with thousands or millions of random variables. Each random variable can have many states and associated probability values, which makes it difficult to store them efficiently and compactly. For example, if we have a system with 1000 binary random variables, we need to store 2^{1000} probability values to do reasoning in a naive way. Thus, we need an efficient way to represent these probability distributions compactly.

Graph theory, which induces conditional dependence and independence assumptions, is used to represent the random variables and their relationships compactly. PGMs are a graph-based representation of complex probabilistic distribution over a set of random variables. The nodes of the graph represent the random variables, the edges represent the probabilistic interactions between them and the probabilities associated with each node are stored. The number of probability values stored based on a graph structure is modest compared to the size of the represented probability distribution. A Bayesian network (BN) is a commonly used PGM in uncertainty reasoning of many real world problems such as electrical power system diagnosis [69], medical diagnosis [80] and image recognition [71].

A BN is a directed acyclic graph, whose nodes are random variables and whose edges often represent causal-effect relationships between the variables. Associated with each node is a set of conditional probability values that quantify the strength of causal relationship between the node and its parents. Estimation of parameters can be challenging. Fortunately, using complete data, learning parameters is reduced to finding maximum likelihood or Bayesian estimates, which are the estimates that maximize the probability of observing the data. These parameters will be similar to the true values when the amount of available data is large [28].

1.2 Expectation Maximization (EM)

In real world scenarios, the data can be incomplete *i.e.,* can have missing or hidden values. For example, hidden variables can arise due to privacy or security constraints and missing values can arise due to a system fault. Hidden variables can also arise when the training data is large and unlabeled. Labeling a training data is usually done by a person and this is a time consuming task. A combination of the naive Bayes classifier and the EM algorithm is used to incorporate the unlabeled trained data where the EM algorithm probabilistically labels unlabeled documents and estimates the parameters of a naive Bayes classifier [79]. When the dataset is incomplete, either with some hidden variables or missing values, expectation maximization (EM) [18] is widely used to estimate the parameters. EM alternates between an expectation step (E-step), in which it calculates an expectation of the likelihood by including the missing variables as if they were observed and a maximization (M-step), in which the maximum likelihood estimates of the parameters are found by maximizing the expected log likelihood found in the E-step .

1.3 Challenges Associated with EM

Despite its popularity, the EM algorithm has several severe limitations: the local optima problem; the computational complexity of the E-step and the M-step; the slow convergence problem and lack of understanding of EM progress. The EM algorithm can easily converge to a locally (but not globally) optimal solution, depending on the initialization of the algorithm. The most prevalent way to mitigate the local optima problem is the multiple starting point strategy [43, 68], wherein we initialize the EM algorithm from multiple random starting points. After these EM runs have completed, we choose the run that resulted in the highest data log likelihood and use that as a conservative estimate of the global maximum. The multiple starting points strategy allows us to search the parameter space extensively for globally optimal solutions, but can be extremely expensive, considering that we expend a significant amount of processing power on runs that are in no way guaranteed to be the global optimum or even close to it.

Stochastic algorithms have been combined with the EM algorithm to alleviate the local optima problem. The Genetic Algorithm (GA) was first introduced by Holland (1975), based on the principle of natural selection in the evolution of species. A GA framework was proposed for solving the local maxima problem in EM [46, 85], where the focus is on learning Gaussian mixture models from multivariate data. It is found that the genetic-based EM algorithm outperforms the baseline EM, as it achieves the same fitness scores while identifying the correct number of Gaussian components more often than the baseline EM.

A second problem with EM is the computational complexity in the E-step and the M-step. This issue has been addressed by many variants of the EM algorithm [17, 65]. The E-step cannot be performed in closed form in some cases, so stochastic approximation EM (SAEM) [17] is introduced. SAEM replaces the E-step of the EM algorithm by one iteration of a stochastic approximation procedure. The M-step can also become computationally unattractive. The Expectation Conditional Maximization (ECM) algorithm [65] is introduced to deal with this issue. ECM replaces each complex M-step with a sequence of simple conditional maximization (CM) steps in which each parameter is maximized individually, conditionally on the other parameters remaining fixed.

A third problem with EM is that it can take an excessive number of iterations for convergence, *i.e.,* slow convergence. A number of methods have been proposed to counter-act this slow convergence such as those utilizing conjugate gradient methods [45, 81, 93]. Most of these methods require a modification to the original EM algorithm thus making it more complex, very difficult to analyze and hence not popular in practice.

Finally, there is a lack of understanding of how the probability values of different random variables change during the progress of EM learning. In a multiple random starting point setup, we also may not be sure which EM runs are progressing well and which are not.

1.4 Summary of Thesis Contributions

In this thesis, we try to solve the local optima and the slow convergence problems of EM by the application of evolutionary techniques. We attempt to speed up the EM algorithm and also produce a higher quality solution. To tackle the lack of understanding problem, we provide an user interface integrated with a multi-focus zooming technique to analyze the change in probability values during EM learning. We focus on the problem of learning parameters for Bayesian networks in the presence of incomplete data.

- A Genetic Algorithm for Expectation Maximization (GAEM) that combines the monotonic property of EM with the stochastic property of genetic algorithms is developed. Powerful replacement mechanisms are investigated to counter-act the local maxima problem of EM. Replacement happens among a group of random candidate solutions, those which perform better, to form the parents for the next generation. The exploration of the search space is done using genetic operators such as mutation and crossover. We investigated the effects of genetic operators for different mutation and crossover probabilities. We studied the varying effects of population size on the local optima for different configurations of crossover, mutation and replacement techniques. We found that small population size always performs better than large population sizes thus saving heavy computations. Using finite Markov chain theory, we prove that GAEM converges to a global optimum. Our experiments show that GAEM significantly reduces the average number of iterations and speeds up EM on two Bayesian networks.
- We performed extensive experiments on Alarm and Carstarts Bayesian networks for varying degree of hidden variables and sample set configurations. We found that the average

number of iterations taken by all EM runs is significantly larger than the average number of iterations taken by successful EM runs. So, an age-layered EM algorithm (ALEM) [94] is developed to speed up EM. The key idea is to identify a poorly progressing EM run during an EM iteration and discard it from the population. The age-layered strategy is influenced by the age-layered population structure (ALPS) paradigm, originally implemented for genetic algorithms [40]. From our experiments, we found that ALEM manages to significantly decrease the average number of iterations required on four Bayesian networks, but at the same time still achieves the global optimum, or gets very close, in all instances.

- An intuitive user interface, used to visualize BNs is developed. The user interface consists of a network window and a detail window. The network window shows the graphical view of the BN and the detail window shows the probabilities associated with the nodes in the BN. A multi-focus technique [100, 101] is developed, which allows analysts to zoom in on multiple nodes and analyze them simultaneously. The visualization tool is used for problem solving tasks with a BN. It also provided useful analysis regarding the behavior of the EM algorithm and the progress of GAEM for one generation.

1.5 Outline of Thesis

In this thesis, we discuss some of the technical background in Chapter 2. We describe a novel GA-based speed-up technique (GAEM) and describe experimental results in Chapter 3. Another novel speed-up technique, ALEM, based on an age-layered strategy, is discussed in Chapter 4. Both these techniques are used to speed-up the EM algorithm for parameter estimation in Bayesian networks. In Chapter 5, we present a user interface for visualizing BNs. It is integrated with a multi-focus zooming technique that allows a user to zoom in on multiple nodes simultaneously study them in detail. The technique is useful in various diagnostic and analysis tasks. It is also useful for understanding EM learning. We present conclusion and future work in Chapter 6.

Chapter 2

Technical Preliminaries

Speeding up EM algorithm for Bayesian network machine learning using stochastic methods require an understanding of Bayesian networks, the EM algorithm and stochastic methods. In this chapter, we give an introduction to Bayesian networks. We discuss the problem of learning the parameters from data using maximum likelihood estimation and the EM algorithm. Finally, we give an overview of a stochastic optimization technique, genetic algorithms, and describe hybrid genetic algorithms.

Consider an example scenario [92], where one day we walk in the lawn and find the grass is wet. It may be because it rained as the weather is cloudy. But the grass can also be wet when the sprinkler is on. Usually the gardener turn on the lawn sprinklers when it is not cloudy. Given that the grass is wet, we want to know what caused it, whether the rain or lawn sprinklers.

To answer such queries, we first need to specify the random variables of interest in this problem. Random variables are Cloudy (C), Rain (R), Sprinklers (S) and GrassWet (W). Each variable can take one of the two values: 1 or 0. Secondly, we need to come up with the probability values for each assignment of the random variables. For example, one such assignment can be $C = 1, R = 1, S = 0, W = 1$. If we are living in a rainy place, it is most likely that the grass gets wet due to rain. Then the probability of the above assignment will be close to 1. The probabilities of all such assignments is called the joint probability distribution. The joint probability distribution assigns probabilities to all possible events, as shown in the Table 2.1 below:

Suppose we want to know the probability that the grass is wet because of rain, $P(R = 1|W =$

C	R	S	W	$Probability$	C	R	S	W	$Probability$
1	1	1	1	0.3	0	1	1	1	0.02
1	1	1	0	0.1	0	1	1	0	0.01
1	1	0	1	0.08	0	1	0	1	0.04
1	1	0	0	0.02	0	1	0	0	0.05
1	0	1	1	0.04	0	0	1	1	0.02
1	0	1	0	0.16	0	0	1	0	0.04
1	0	0	1	0.07	0	0	0	1	0.01
1	0	0	0	0.01	0	0	0	0	0.03

Table 2.1: Joint probability distribution table over binary random variables C, R, S and W.

$1)$. Using the probability theory, we can compute the probability of $P(R = 1|W = 1)$ as shown below:

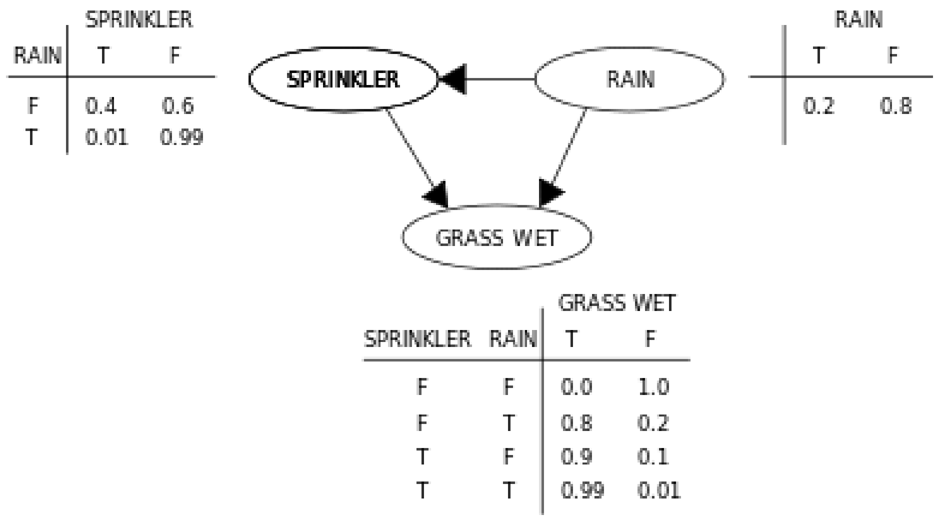
$$P(R = 1|W = 1) = \frac{\sum_{C,S} P(R = 1, W = 1, C, S)}{\sum_{C,S,R} P(W = 1, C, S, R)} \tag{2.1}$$

The values for the numerator and denominator can be obtained from the joint distribution table by marginalization over the remaining variables.

The size of the joint probability distribution table is exponential in the number of random variables involved in the problem. For our problem, we have 4 variables each of which can take two values, so we have 16 entries as shown in the above Table 2.1. It is also difficult to come up with the probability values for all such assignments in the table. We need a compact representation of the joint distribution.

2.1 Bayesian Networks

Bayesian networks often provide compact representations of the joint distributions. Formally, a BN is a directed acyclic graph whose vertices are random variables and the directed edges denote the dependency relationship among the random variables. Bayesian networks are well suited for systems where we need to make predictions under uncertainty [84]. We use upper case letters (X) to denote the random variables and lower case letters (x) denote the specific state of the random variable. Let θ denote the probability values (also called parameters). Let bold face



	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99

RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Figure 2.1: Sprinkler Bayesian Network; inspired by Stuart J. Russell and Peter Norvig [92]

upper case letters **X** denote the variable sets and bold face lower case letters **x** denote their set instantiations. We use **Pa** to denote the parent set of the random variable X.

Formally, a BN is defined as $\mathcal{B} = (\mathcal{G}, P)$, where $\mathcal{G}$ is a directed acyclic graph and P is the set of probability distributions. The graph is denoted by $\mathcal{G} = (\mathbf{X}, \mathbf{E})$ where $\mathbf{X} = \{X_1, X_2 \ldots X_n\}$ is the node set and $\mathbf{E}$ is the edge set. Each node X_i represents a random variable that can be discrete having a countable number of states or continuous. If there is an edge from X_i to X_j, then $(X_i, X_j) \in \mathbf{E}$. We call X_i as the parent of X_j. $\mathbf{Pa}(X_j)$ denotes the set of all parents of X_j. The key property of BNs is the *conditional independence* of the variables from any of their non-descendants, given the value of their parent variables, *i.e.,* given the value of $\mathbf{Pa}(X_j)$, X_j is conditionally independent of all its non-descendants. A BN factorizes a joint distribution $P(\mathbf{X})$ as shown below:

$$P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i \mid \mathbf{Pa}(X_i)) \tag{2.2}$$

Figure 2.1 shows sprinkler Bayesian network [47] in which the Cloudy variable (whether it is cloudy) has two casual links, one causal link to whether it rains, and another causal link to whether the lawn sprinklers are turned on (because a gardener who observes clouds is less likely to turn the sprinklers on). Both rain and sprinklers have an effect on whether the grass gets wet. Once we have specified the structure of the Bayesian network, we need to know the probability of every state for those variables. From the conditional independence assumption, it makes sense to specify probabilities in a Bayesian network by specifying the conditional probability of a node given its parents.

A conditional probability table (CPT) of a variable X_i is a probability distribution of X_i for each combination its of parents values. CPTs can be shown in tabular format next to each node in Figure 2.1. Such CPTs can be created by domain experts or machine learning techniques. The size of the CPT is only exponential in the number of parents, this makes inference (Section 2.2) and machine learning (Section 2.3) computationally feasible in Bayesian networks.

2.2 Inference in Bayesian Networks

Let $\mathbf{E} \subset \mathbf{X}$ be the variables which are under observation (we call them evidence variables), and $\mathbf{e}$ denote their observed values (evidence). A Bayesian network can help to solve different probabilistic queries. The process of solving the probabilistic queries is called inference. The inference algorithms assume that the nodes in $\mathbf{E}$ are clamped to values $\mathbf{e}$. Computation of *most probable explanation (MPE)* amounts to finding a most likely assignment ($\mathbf{y}$) to all of the non-evidence variables $\mathbf{Y} = \mathbf{X} - \mathbf{E}$, or MPE($\mathbf{Y}$). Computation of marginals (or beliefs) amounts to inferring the *most likely value (MLV)* over one query variable $Q \in \mathbf{Y}$. Computation of the *maximum a posteriori probability (MAP)* generalizes MPE computation and finds a most probable instantiation over some variables $\mathbf{Q} \subseteq \mathbf{Y}$, MAP($\mathbf{Q}, \mathbf{e}$). Different BN inference algorithms can be used to perform the above computations. Different BN inference algorithms [51], can be used to perform the above computations.

2.3 Parameter Estimation for Bayesian Networks

In order to be able to answer different queries, it is necessary that the conditional distribution of each random variable is fully specified. Often these conditional distributions include parameters which are unknown and need to be learned from data. Parameter estimation can be thought of as the process of estimating these distributions for the random variables. If the data is complete, maximum likelihood estimation is used. If the data is incomplete, expectation maximization algorithm (EM) [19] is widely used.

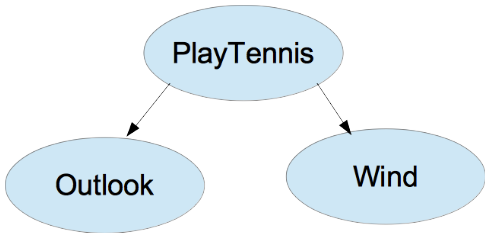


Figure 2.2: PlayTennis Bayesian network; inspired by Tom Mitchell [70]

2.3.1 Maximum Likelihood Estimation

In Maximum Likelihood Estimation(MLE), we assume that we know the structure of the Bayesian network and have complete data with no missing values. Consider a simple Bayesian network (shown in Figure 2.2), where the random variables are $X_1 = PlayTennis$, $X_2 = Outlook$, and $X_3 = Wind$. We denote the sample space as Ω. All the variables are binary, $\Omega(X_1) = \{yes, no\}$, $\Omega(X_2) = \{sunny, rain\}$ and $\Omega(X_3) = \{weak, strong\}$. The sample data is shown in Table 2.2. We denote the whole dataset as $\mathcal{D}$ and each data sample as d_i. Let M be the total number of samples in the dataset $\mathcal{D}$.

Example	$PlayTennis(X_1)$	$Outlook(X_2)$	$Wind(X_3)$
1	no	sunny	weak
2	yes	rain	weak
3	yes	sunny	strong
4	no	rain	strong
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 2.2: Sample Data For PlayTennis network

Our goal is to estimate the “best” set of parameters ($\boldsymbol{\theta_{best}}$) that fits the given data. $\boldsymbol{\theta_{best}}$ is called the MLE estimate. We use likelihood function ($L(\boldsymbol{\theta} : \mathcal{D})$) to measure the likelihood of data given the parameters. $X[m]$ denote the observed variable. For computational convenience, the MLE estimate is obtained by maximizing the log-likelihood function ($LL(\boldsymbol{\theta}|\mathcal{D})$).

The likelihood function is given by:

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_{m=1}^{M} P(X[m]|\boldsymbol{\theta}) \tag{2.3}$$

The log-likelihood function is given by:

$$LL(\boldsymbol{\theta} : \mathcal{D}) = \sum_{m=1}^{M} \log P(X[m]|\boldsymbol{\theta}) \tag{2.4}$$

MLE estimate is given by:

$$\boldsymbol{\theta_{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{m=1}^{M} \log P(X[m]|\boldsymbol{\theta}) \tag{2.5}$$

In the case of PlayTennis BN, the likelihood function is given by:

$$P(\mathcal{D}|\boldsymbol{\theta}) = \prod_{m=1}^{M} P(X_1[m])P(X_2[m]|X_1[m])P(X_3[m]|X_1[m]) \tag{2.6}$$

The log-likelihood function is given by:

$$\log P(\mathcal{D} \mid \boldsymbol{\theta}) = \sum_{m=1}^{M} (\log P(X_1[m]) + \log P(X_2[m]|X_1[m]) + \log P(X_3[m]|X_1[m]) \tag{2.7}$$

Suppose we are interested in finding $\theta_{X_3=weak|X_1=no}$. Differentiating the log-likelihood function (LL) and solving for $\theta_{x_3|x_1}$ gives the maximum likelihood estimate.

$$\begin{aligned}
\frac{\delta log P(\mathcal{D} \mid \boldsymbol{\theta})}{\delta\theta_{X_3|X_1}} &= \frac{\delta \sum_{m=1}^{M} \log P(X_3[m]|X_1[m])}{\delta\theta_{X_3|X_1}} \\
&= \log(\theta^{k}_{X_3|X_1}) + \log(\theta^{t}_{X_3|X_1}) \\
&= k\log(\theta_{X_3|X_1}) + t\log(\theta_{X_3|X_1}) \\
&\quad \text{where } k \text{ and } t \text{ are the times } X_1 = yes \text{ and } X_1 = no \text{ are seen in } \mathcal{D} \\
&= k\log((\theta_{X_3|X_1})) + t\log(1-\theta_{X_3|X_1}) \\
&\quad \text{where } X_3 \text{ is binary (} X_3\text{=weak or } X_3\text{=strong) in our example.}
\end{aligned} \tag{2.8}$$

$$\theta_{X_3|X_1} = \frac{k}{k+t} \tag{2.9}$$

From the above data, we can estimate the conditional probability distributions by counting the number of times a given combination of values are observed in the dataset. Eg. $\theta_{X_1=yes} = \frac{\#(X_1=yes)}{\#(X_1=yes)+\#(X_1=no)}$ and $\theta_{X_3=weak|X_1=no} = \frac{\#(X_3=weak)}{\#(X_1=no)}$. In general, $\theta x|u = \frac{\#(x,u)}{\#(u)}$, where $\#(x,u)$ is the number of times x and u appear in the dataset, and $\#(u)$ is the number of times u appears in the dataset.

When the data is incomplete, the problem of estimating the conditional distribution becomes harder. The data is incomplete in three different ways. First, missing at random (MAR), where the probability that a value is missing depends on observed values. For example, a medical test result is missing because a doctor was fairly sure of a diagnosis given earlier test results. Second, missing completely at random (MCAR), where the probability that a value is missing is independent of the observed values. For example, a sensor fails to record a value due to a power blip. We will be focussing on the second case (MCAR), we call it the hidden variables case. In this case, we assume the variable is never observed. Expectation Maximization algorithm [18] is typically used for estimating parameters with incomplete data. EM starts with some initial parameters $\boldsymbol{\theta}^{\mathbf{0}}$, called a random starting point, and successively improves on them via iteration.

2.3.2 Expectation Maximization Algorithm

The Expectation Maximization (EM)[18] family of algorithms is one of the most popular methods for learning the parameters of probabilistic models. It is widely used for maximum likelihood estimation in incomplete data models. Let $\mathbf{Y}$ be the set of observed variables and $\mathbf{Z}$ be the set of unobserved variables. We cannot use MLE to calculate $\boldsymbol{\theta_{ML}}$, where $\boldsymbol{\theta_{ML}} \leftarrow \arg\max_{\boldsymbol{\theta}} \log(P(\mathbf{Y}, \mathbf{Z}|\boldsymbol{\theta}))$. We lost its unimodality and its closed form representation because of the hidden variables $\mathbf{Z}$. EM is used to estimate the parameters using the expected log likelihood $E_{P(\mathbf{Z}|\mathbf{Y},\boldsymbol{\theta})}[\log(P(\mathbf{Y}, \mathbf{Z}|\boldsymbol{\theta}))]$.

The likelihood function is given by:

$$L(\boldsymbol{\theta} :< \mathcal{D}, \mathbf{Z} >) = \prod_{m=1}^{M} \sum_{Z[m] \in \mathbf{z}} P(Y[m], Z[m] = z|\boldsymbol{\theta}) \tag{2.10}$$

The log-likelihood function is given by:

$$LL(\boldsymbol{\theta} :< \mathcal{D}, \mathbf{Z} >) = \sum_{m=1}^{M} \log \sum_{Z[m] \in \mathbf{z}} P(Y[m], Z[m] = z|\boldsymbol{\theta}) \tag{2.11}$$

We use expected log likelihood due to the presence of hidden variables Z. Expected log likelihood function is given by:

$$E_{P(Z[m]|Y[m],\boldsymbol{\theta})}[LL(\boldsymbol{\theta} :< \mathcal{D}, \mathbf{Z}, \boldsymbol{\theta^t} >)] = \sum_{m=1}^{M} \sum_{Z[m] \in \mathbf{z}} P(Z[m]|Y[m], \boldsymbol{\theta^t}) \log P(Y[m], Z[m] = z|\boldsymbol{\theta}) \tag{2.12}$$

where $\boldsymbol{\theta}^{\boldsymbol{t}}$ are the parameters at iteration t and $\boldsymbol{\theta}$ are the new estimated parameters.

MLE estimate can be computed using the expected log likelihood function.

$$\boldsymbol{\theta}_{\boldsymbol{ML}}^{\boldsymbol{t+1}} = \arg\max_{\boldsymbol{\theta}} \sum_{m=1}^{M} \sum_{Z[m]\in \boldsymbol{z}} P(Z[m]|Y[m], \boldsymbol{\theta}^{\boldsymbol{t}}) \log P(Y[m], Z[m] = z|\boldsymbol{\theta}) \tag{2.13}$$

In the case of PlayTennis BN, suppose we are interested to know whether it is good to play tennis given the outlook and wind. But lets say that, we have collected data about outlook and wind for the past few days. There is no data about the playTennis random variable. The collected sample data is shown in Table 2.3. The column ”Count” represents the number of times the data is seen in the dataset. The Bayesian network is shown in Figure 2.2.

$PlayTennis(Z_1)$	$Outlook(Y_2)$	Wind (Y_3)	Count
?	sunny	weak	6
?	rain	strong	4
?	sunny	strong	1
?	rain	weak	1

Table 2.3: Sample incomplete data for PlayTennis network; PlayTennis is a hidden variable.

In the E-step, the EM algorithm assigns a probability distribution for the hidden variables given the observed data and the current model ($P(Y[m], Z[m] = z|\boldsymbol{\theta})$). This step involves a call to the inference algorithms. For a smaller BN (eg. PlayTennis BN), we can compute the joint distribution by hand. So the posterior distribution over the hidden variable involves a simple calculation as shown below for $P(Z_1 = yes|Y_2 = sunny, Y_3 = weak)$. Similarly, we find $P(Z_1 = yes|Y_2 = rain, Y_3 = weak)$,$P(Z_1 = yes|Y_2 = sunny, Y_3 = strong)$ and $P(Z_1 = yes|Y_2 = rain, Y_3 = strong)$.

$$\begin{aligned} P(Z_1 = yes|Y_2 = sunny, Y_3 = weak) &= \frac{P(Z_1 = yes, Y_2 = sunny, Y_3 = weak)}{P(Y_2 = sunny, Y_3 = weak)} \\ &= \frac{0.4 * 0.55 * 0.53}{(0.4 * 0.55 * 0.53) + (0.6 * 0.41 * 0.42)} = 0.53. \end{aligned}$$

In the M-step, the parameter estimation is done using the distribution over the hidden variables to generate expected counts for the different sample cases. The expected sufficient statistics is calculated as the sum of the probabilities over all the sample cases for a given value of the hidden variable. The expected sufficient statistics for the playTennis BN is shown below:

$$
\begin{aligned}
ESS(Z_i : D, \theta) &= \sum_{m=1}^{M} P(Z_i[m]|Y[m], \theta) \\
ESS_{\theta^t}(Z_1 = yes) &= \sum_{m=1}^{M} (P(Z_1 = yes) = 4.68 \\
ESS_{\theta^t}(Y_2 = sunny, Z_1 = yes) &= \sum_{m=1}^{M} (P(Z_1 = yes|Y_2 = sunny) = 2.54 \\
ESS_{\theta^t}(Y_3 = weak, Z_1 = yes) &= \sum_{m=1}^{M} (P(Z_1 = yes|Y_3 = weak) = 2.51 \\
ESS_{\theta^t}(Z_1 = no) &= \sum_{m=1}^{M} (P(Z_1 = no) = 7.31
\end{aligned}
$$

The new set of parameters are estimated using the MLE as shown below, where **X** denotes all the variables in the BN, **x** denotes its values and **u** denotes the values of parent variable.

$$
\theta^{t+1}_{x_i|\mathbf{u}} = \frac{ESS_{\theta^t}(x_i, \mathbf{u})}{ESS_{\theta^t}(\mathbf{u})}
$$

$$
\begin{aligned}
P(Z_1 = yes) &= \frac{ESS(Z_1 = yes)}{ESS(Z_1 = yes) + ESS(Z_1 = no)} = 0.39 \\
P(Y_2 = sunny|Z_1 = yes) &= \frac{ESS(Y_2 = sunny, Z_1 = yes)}{ESS(Z_1 = yes)} = 0.54 \\
P(Y_2 = sunny|Z_1 = no) &= \frac{ESS(Y_2 = sunny, Z_1 = no)}{ESS(Z_1 = no)} = 0.34 \\
P(Y_3 = weak|Z_1 = yes) &= \frac{ESS(Y_2 = weak, Z_1 = yes)}{ESS(Z_1 = yes)} = 0.54 \\
P(Y_3 = weak|Z_1 = no) &= \frac{ESS(Y_2 = weak, Z_1 = no)}{ESS(Z_1 = no)} = 0.34
\end{aligned}
$$

The expected log likelihood is calculated using the below equation:

$$\sum_{m=1}^{M} E_{P(\mathbf{Z}|(\mathbf{Y},\theta))}[LL(\theta|D)] = \sum_{i=1}^{N} \sum_{(y_i,ui)} \sum_{m=1}^{M} P(\mathbf{Z} = z|d[m],\theta) log\theta_{(y_i|ui)}$$

$$= \sum_{i=1}^{N} \sum_{(y_i,ui)} ESS_{\theta^t}(y_i, ui) log\theta_{(y_i|ui)}$$

$$\begin{aligned} LL = 4 \times \log((0.39 * 0.46 * 0.46) + (0.61 * 0.66 * 0.66)) + \\ \times \log((0.39 * 0.54 * 0.46) + (0.61 * 0.34 * 0.66)) + \\ \times \log((0.39 * 0.46 * 0.54) + (0.61 * 0.66 * 0.34) + \\ 6 \times \log((0.39 * 0.54 * 0.54) + (0.61 * 0.34 * 0.34)) \\ = -15.99 \end{aligned}$$

Table 2.4: PlayTennis BN: Posterior distribution for the first 7 iterations

Iteration	1	2	3	4	5	6	7
$P(Z_1 = yes \mid Y_2 = sunny, Y_3 = weak)$	0.53	0.74	0.87	0.95	0.98	0.99	0.99
$P(Z_1 = yes \mid Y_2 = rain, Y_3 = strong)$	0.29	0.16	0.09	0.04	0.02	0.01	0.01
$P(Z_1 = yes \mid Y_2 = sunny, Y_3 = strong)$	0.42	0.43	0.45	0.46	0.47	0.47	0.47
$P(Z_1 = yes \mid Y_2 = rain, Y_3 = weak)$	0.39	0.43	0.44	0.46	0.47	0.47	0.47

EM alternates between performing an E-step, which computes an expectation of the likelihood by including the incomplete data as if they were observed, and a M-step, which computes the maximum likelihood estimates of the parameters by maximizing the expected likelihood found in the E step. The parameters found in the M step are then used to begin another E step and the process is repeated until convergence. In our PlayTennis BN, the EM algorithm converges around 7 iterations. Table 2.4 shows the posterior probability distribution of the PlayTennis (Z_1) hidden variable. The learned parameters for the first 7 EM iterations of the PlayTennis BN are shown in Table 2.5. The CPTs for Y_2 and Y_3 are the same (Table 2.5), which also makes sense, since the data is completely symmetric for Y_2 and Y_3. When $z_1 = yes$, y_2 and y_3 are almost certainly $sunny$ and $weak$ respectively. When $z_1 = no$, y_2 and y_3 have a moderately high probability of being $rain$ and $strong$ respectively. The graph shown in Figure 2.3 shows the progress

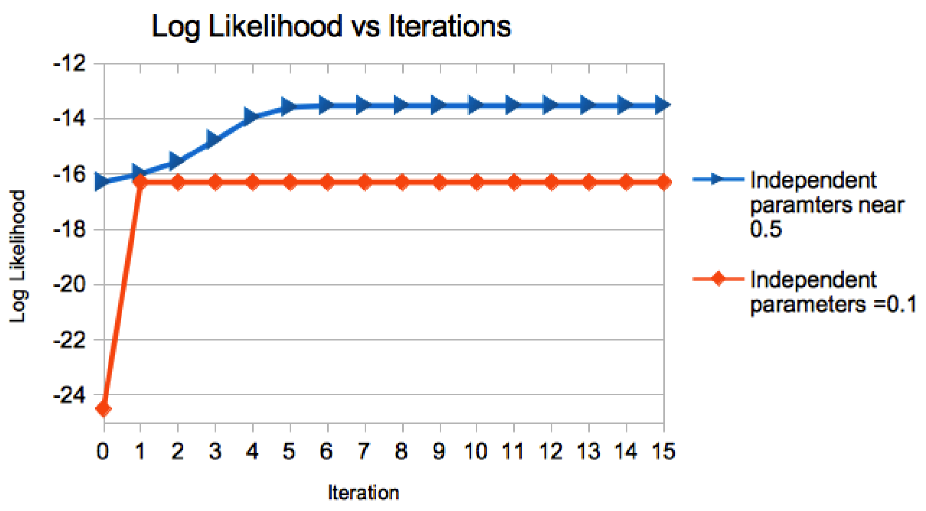


Figure 2.3: PlayTennis BN: There are two local optima, one is around $LL = -13.52$ when the independent parameters are set near 0.5. Other local optima is stuck at $LL = -16.3$ when the independent parameters are set to a same value 0.1.

of log likelihood of the observed data given the model as a function of the iteration, we can see that it increases monotonically. The EM algorithm takes larger increments in the log likelihood value , especially during the initial EM iterations [66].

Table 2.5: PlayTennis BN: Learned parameters for the first 7 iterations

Iteration	1	2	3	4	5	6	7
$P(Z_1 = yes)$	0.39	0.39	0.42	0.41	0.41	0.41	0.41
$P(Y_2 = sunny\|Z_1 = yes)$	0.54	0.61	0.24	0.80	0.86	0.89	0.89
$P(Y_2 = sunny\|Z_1 = no)$	0.34	0.29	0.55	0.15	0.10	0.09	0.08
$P(Y_3 = weak\|Z_1 = yes)$	0.54	0.61	0.24	0.80	0.86	0.89	0.89
$P(Y_3 = weak\|Z_1 = no)$	0.34	0.29	0.55	0.15	0.10	0.09	0.08

The EM algorithm has been widely used in many engineering fields, from estimating transition and emission probabilities in hidden Markov models in speech recognition (the Baum-Welch algorithm [87]) to microarray gene expression clustering in computational biology [20]. Despite its successful and widespread use, the EM algorithm has several limitations. It has a strong tendency to gravitate towards the locally optimal solution. Depending on how our training algorithm is initialized, one can settle on a set of parameters that are locally but not globally optimal. This phenomenon can in turn lead to other issues, for example poor generalization to unseen test data.

Several techniques have been proposed to allay this problem, for example initializing EM from multiple random starting points and selecting the highest likelihood out of all runs. The multiple starting point strategy is a single-point deterministic search. Though it can improve the performance of search but still are adversely affected by the existence of multiple local optima. So many EM runs have to be started which can be very expensive computationally. Consequently, many speed-up techniques have been proposed. These speed-up techniques might involve modification of the original algorithm, require tuning of a number of parameters, thus making the original algorithm more complex.

In this work, we focus on the application evolutionary techniques for speeding-up EM algorithm. In particular, we apply genetic algorithms for speeding up EM algorithm at the same achieving a better log likelihood. Genetic algorithm is chosen as it is a population based search in contrast to other stochastic methods such as tabu search or simulated annealing.

2.4 Genetic Algorithms

Genetic Algorithm (GA) was first introduced by Holland (1975) based on the principle of natural selection in the evolution of species. GAs are mainly used as function optimizers. It is based on a stochastic and population based search. The stochasticity is introduced by mutation and crossover operator. These operators are applied on each individual in the population. GAs are widely used as they are robust, simple to implement, and generic in nature which contributes its success in many applications.

In this section, first we present a simple genetic algorithm. We extend it to hybrid genetic algorithm and discuss its implementation in more detail.

2.4.1 The Simple Genetic Algorithm

A simple genetic algorithm is shown in Algorithm 2.4.1. In GA, a set of individuals which constitute a population are evolved toward better solutions. Each individual, denoted by ρ_i^t, is called as a chromosome or a solution. Traditionally, chromosomes have alphabet $\{0, 1\}$. The evolution usually starts with the set of randomly generated solutions, denoted by $\boldsymbol{\rho^t} = \{\rho_1^t, \rho_2^t, \ldots, \rho_n^t\}$ where $t = 0$ representing the initial population and n is the total number of individuals. After creating the initial population, each individual is evaluated and assigned a fitness value. The evaluation or objective function provide a measure of performance of the individual with respect to a particular set of parameters. Each individual is selected with selection probability proportional to it's fitness. Selected individuals ($\boldsymbol{\rho^{t'}}$) undergo mutation and crossover. Crossover happens between two individuals with crossover probability p_c and constitute population $\boldsymbol{\rho^{t''}}$. For example, $\boldsymbol{c_1} = (000111)$ and $\boldsymbol{c_2} = (111000)$ are two individuals. In a single point crossover, with the crossover point =3, the new set of offspring individuals after crossover are $\boldsymbol{c_3} = (111111)$ and $\boldsymbol{c_4} = (000000)$. These offspring individuals $\boldsymbol{c_3}$ and $\boldsymbol{c_4}$, then undergo mutation with mutation probability p_m and form population $\boldsymbol{\rho^{t'''}}$. For example, if mutation on $\boldsymbol{c_3}$ happens at the first position, then the mutated individual will look like, $\boldsymbol{c_3} = (011111)$. After crossover and mutation, the offspring individuals in the current population compete with the parents to be selected for the next generation. These selected set of individuals ($\boldsymbol{\rho^{t''''}}$) become parents for the next generation. The above steps, selection, crossover and mutation happen iteratively. The population in each iteration is called a generation. Such a GA is called a generational GA where the entire population is replaced in every generation.

When genetic algorithms are used for optimization, they are often not used as simple genetic algorithm. But they are viewed as a framework for population based search. The genetic algorithms used for optimization are mostly modified to improve their solutions in terms of solution quality and computational efficiency. A hybrid genetic algorithm is developed in which the ability of the standard genetic operators is augmented with a local search algorithm [23].

2.4.2 Genetic Algorithm with Local Search

Global search, for example as found in GAs, aims to ensure that every part of a search space is searched enough to provide a reliable estimate of the global optimum. A specific local search method, often tailored to a given problem, provides a refinement of the current solution which will often produce a better solution. A combination (GALS) of the two often produces better solutions than either of them alone. That is the reason why many optimization algorithms combine a domain specific local search strategy with a global search method to produce better solutions.

Algorithm 2.4.1: GA($maxgen$)

```
procedure MAIN($\boldsymbol{\rho^0}, maxgen$)
  $t \leftarrow 1$
  comment:
  Randomly generate individuals; evaluate
  $\boldsymbol{\rho^t} \leftarrow$ RANDOMIZE($\boldsymbol{\rho^0}$)
  $F(\boldsymbol{\rho^t}) \leftarrow$ DOEVALUATION($\boldsymbol{\rho^t}$)
  repeat
    $\boldsymbol{\rho^{t'}} \leftarrow$ SELECTION($\boldsymbol{\rho^t}$)
    $\boldsymbol{\rho^{t''}} \leftarrow$ DOCROSSOVER($\boldsymbol{\rho^{t'}}, p_c$)
    $\boldsymbol{\rho^{t'''}} \leftarrow$ DOMUTATION($\boldsymbol{\rho^{t''}}, p_m$)
    $F(\boldsymbol{\rho^{t'''}}) \leftarrow$ DOEVALUATION($\boldsymbol{\rho^{t'''}}$)
    $\boldsymbol{\rho^{t+1'}} \leftarrow$ REPLACEMENT($F(\boldsymbol{\rho^t}), F(\boldsymbol{\rho^{t'''}})$)
    $t \leftarrow t + 1$
  until $t \leq maxgen$
```

Algorithm 2.4.2: GALS($maxgen$)

```
procedure MAIN($\boldsymbol{\rho^0}, maxgen$)
  $t \leftarrow 1$
  comment:
  Randomly generate individuals; evaluate
  $\boldsymbol{\rho^t} \leftarrow$ RANDOMIZE($\boldsymbol{\rho^0}$)
  comment:
   Do local search; evaluate
  $\boldsymbol{\rho^{t'}} \leftarrow$ APPLY LOCAL SEARCH($\boldsymbol{\rho^t}$)
  $F(\boldsymbol{\rho^{t'}}) \leftarrow$ DOEVALUATION($\boldsymbol{\rho^{t'}}$)
  repeat
    $\boldsymbol{\rho^{t''}} \leftarrow$ SELECTION()$\rho^{t'}$
    $\boldsymbol{\rho^{t'''}} \leftarrow$ DOCROSSOVER($\boldsymbol{\rho^{t''}}, p_c$)
    $\boldsymbol{\rho^{t''''}} \leftarrow$ DOMUTATION($\boldsymbol{\rho^{t'''}}, p_m$)
    $F(\boldsymbol{\rho^{t''''}}) \leftarrow$ DOEVALUATION($\boldsymbol{\rho^{t'''}}$)
    $\boldsymbol{\rho^{t+1}} \leftarrow$ REPLACEMENT($F(\boldsymbol{\rho^{t'}}), F(\boldsymbol{\rho^{t''''}})$)
    $\boldsymbol{\rho^{t+1'}} \leftarrow$ APPLY LOCAL SEARCH($\boldsymbol{\rho^{t+1}}$)
    $t \leftarrow t + 1$
  until $t \leq maxgen$
```

Figure 2.4: A simple genetic algorithm is shown in Algorithm 2.4.1 [35]. A genetic algorithm with local search is shown in Algorithm 2.4.2 [52].

Hybrid genetic algorithms, which integrates GAs and local search are used in cloud computing [109], genetic engineering [42] and machine learning [53, 82].

Hybrid genetic algorithms are used in two different ways. The first approach is to use the local search after one complete run of a GA. Here the local search is used to fine tune the solutions produced from GA. For example, a reward based decision making procedure of when to use the GA and when to use the local search is discussed [62]. The other approach is to use the local search within every GA iteration. In this type of GALS hybrids, the local and global search can influence each other's behavior. A simple GALS framework is shown in Algorithm 2.4.2.

It can be seen from Algorithm 2.4.2 that the genetic operators are applied on the new set of individuals $\rho^{t'}$, produced by the local search. This type of learning where learned individuals of local search replaces the parent individuals is called *Lamarckian evolution*. The alternative to *Lamarackian evolution* is *Baldwinian evolution* where the learned individual of local search is discarded and only its fitness influences the search. We consider *Lamarckian evolution* as we are interested not only in the fitness of the individual but also on the genes (learned parameters) of the individual.

Hybrid genetic algorithms are not only seen as variants of GAs that have the local search component. The use of elitism for keeping the best solution seen so far in the replacement procedure is a departure from the traditional genetic algorithm. Hybrid GA acts as a steepest ascent algorithm by guiding the genetic algorithm in the direction of the best solution seen so far. This bias in the direction of exploration will help to arrive at a good solution quickly even with small populations.

In this thesis, we are focused on combining the monotonicity of EM algorithm along with the stochasticity of the GA algorithm based on *Lamarckian evolution*, to speed up the EM at the same time producing better optimal solutions.

Chapter 3

GAEM: Genetic Algorithm for Expectation Maximization

We develop a method called Genetic Algorithm for Expectation Maximization (GAEM) for learning parameters in Bayesian networks. The method is based on using the Expectation Maximization (EM) algorithm. This method aims to combine the global search property of a GA with the local search property of EM. We introduce GAEM, compare it with other related works in the area, study some of its properties both analytically and empirically. Experimentally, GAEM provides a significant speed-up since it tends to select more fit individuals, which converge faster, as parents for the next generation.

3.1 Introduction

In this chapter, we aim to address the local optima and the slow convergence problems of Expectation Maximization algorithm (EM) by combining it with a Genetic Algorithm (GA). GAs belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. A GA selects among a group of random candidate solutions, those which perform better, as the parents for the next generation. We focus on the problem of using the EM algorithm for parameter estimation in Bayesian networks (BNs). Our contributions are the following:

- We present and analyze our Genetic Algorithm for EM (GAEM) where we introduce a Genetic Algorithm framework for EM. The goal of GAEM is to reduce the number of iterations in a multiple random starting point EM setup by selecting more fit individuals as parents for the next generation.

- We classify BNs based on the hardness of their search landscapes. We perform experiments to study the role of varying population sizes, crossover and mutation probabilities, and different replacement techniques for different BN search landscapes.
- We propose a new replacement mechanism called the GAEM-ALEM replacement mechanism. In this mechanism, the child EM run is compared with its parent EM run during the EM local search. The fitness of an EM run is measured by the log likelihood value. If the child EM run has lesser log likelihood than its parent, then it is discarded and replaced by the parent EM run.
- We perform experiments to compare the impact of GAEM in terms of log likelihood, total number of iterations and CPU time on varying sample sizes and replacement mechanisms. We find that the GAEM's solution quality is always higher than the traditional EM and GAEM-ALEM replacement mechanism provides a speed up of 2 to 7 times resulting in a significant reduction in the CPU time.

We discuss related work in Section 3.2. We present our approach, Genetic EM algorithm (GAEM), in Section 3.3, where we combine the Genetic algorithm with EM in an attempt to speed up EM and overcome the local maxima problem. In Section 3.4, we derive the conditions on the parameters of GAEM which ensure its convergence to the global optimum. We perform experiments in Section 3.5 to demonstrate GAEM's ability to reduce the number of iterations while at the same time achieving better solution quality. We conclude with a summary of contributions and future work in Section 3.6.

3.2 Discussion of Related Work

The EM algorithm has three problems: local optima, computational cost of the E-step and the M-step, and slow convergence. Several variants of the EM algorithm have been proposed to address these problems, as discussed in Section 3.2.1 below before comparing with GAEM in Section 3.2.2.

3.2.1 Related Work

The most prevalent way to mitigate the *local optima* problem is the multiple restart strategy, in which the EM algorithm is initialized from n random starting points [13]. After these n EM runs have completed, the run that results in the highest log-likelihood is used as an estimate of the global optimum.

The GA was introduced by based on Darwin's principle of natural selection in the evolution of species [39]. Some of the challenges of stochastic EM implementation are reviewed and a GA method for solving the local maxima problem in EM is proposed [46]. GA and EM are combined for learning Gaussian mixture models [85]. Their algorithm escapes from local optima and identifies the number of Gaussian components more accurately than traditional EM. A random swap EM algorithm is proposed for learning Gaussian mixture models, where a randomly selected component is swapped to a new location in the feature space [108]. In summary, learning Gaussian mixture models from multivariate data was the main focus in the above works [46], [85] and [108]. They show that a GA-based EM algorithm outperforms traditional EM, as it achieves the same fitness score while identifying the correct number of Gaussian components more often.

The problem of the *computational cost* of the E-step and M-step has been addressed by many EM variants [17] [65]. For clustering of large datasets, there are incremental EM and lazy EM speed-up methods [102]. Both these methods are based on a partial E-step and provide encouraging results for Gaussian mixtures. In incremental EM, the EM algorithm cycles through the samples in blocks and updates the parameters incrementally. A method to determine the optimal block size is proposed. In lazy EM, at scheduled EM iterations, only samples that cause significant changes in the parameters are used by EM.

Several methods have been developed to address the problem of *slow convergence* of the EM algorithm. These methods use conjugate gradient, modified Newton Raphson techniques [44], or age-layered methods [94]. Most of these methods require a modification to the original EM algorithm. This makes the overall method more complex, more difficult to analyze, and hence not popular in practice. Age-layered EM has been extended to the MapReduce framework [88], [6] [5].

Other research, even though it does not study the EM algorithm directly, is also relevant to this work. A review of the application of evolutionary algorithms for solving NP-hard problems in BN inference and learning is done [56]. A GA for learning BN structures is developed [57]. This GA searches for the best ordering of the BN variables, where the score of an order is the score of a single high-scoring BN structure [15]. An evolutionary Markov chain Monte Carlo (EMCMC) method is proposed [75], which combines an evolutionary algorithm with a Markov chain Monte Carlo algorithm for learning BN structures from incomplete data. A hybrid algorithm called Population Markov Chain Monte Carlo (PopMCMC) is introduced [58]. Their experimental results show that incorporating information exchange increases the rate of improvement in solutions and the diversity in the population.

3.2.2 Comparison of GAEM to Related Work

We now compare GAEM to related methods. GAEM uses converged EM runs as parents for the next generation unlike other related works [46], [85]. This allows GAEM to explore close-to-optimal regions of the search space and helps the algorithm escape local optima. GAEM does not modify the EM algorithm ([44], [17], [102], [65]) or the training data ([24]). Instead, GAEM is as a wrapper around the EM algorithm and uses the full set of training data to learn the parameters. Certain related works deal with learning Gaussian mixture models using EM [46], [85], and [108]). Others are focused on Bayesian network structure learning [57], [58] and [75] . GAEM is focused on parameter learning in discrete Bayesian networks using the EM algorithm.

3.3 Genetic Algorithm for Expectation Maximization (GAEM)

The GAEM algorithm combines the monotonic improvement property of the EM algorithm with the stochastic property of a GA. A GA starts from a random set of individuals or parents. Two of the parents are selected to undergo crossover and mutation to create two offspring. The fitness of parents and offspring are evaluated through a fitness function. The individuals with the highest fitness will be chosen as parents for the next iteration. Each iteration of this process is called a generation. The entire set of generations is called a GA run.

The EM algorithm is a deterministic method, in that a particular starting point for an EM run will always converge to the same answer. The fitness of an EM run is measured in terms of its log-likelihood (LL), a measure of how well the parameters fit the training data. An EM run can escape from a local but non-global optimum with the help of GA stochasticity introduced by the crossover and mutation mechanisms of GAEM, see Figure 3.1. The input to the GAEM algorithm is a set of EM runs and the output is a converged EM run with the maximum LL among all EM runs seen in the GA run.

3.3.1 Notation and Definitions

Let $\boldsymbol{X} = \{X_1, X_2, ..., X_R\}$ denote the set of random variables in a Bayesian network (BN). Each random variable X_k is associated with a conditional probability table (CPT). An individual ρ^t consists of a vector of CPTs of random variables in a BN.

Definition 3.3.1. The CPT estimate of an individual i at generation t is denoted by Θ_i^t. An individual is defined as vector consisting of CPTs: $\rho_i^t = (\Theta_1^t, \Theta_2^t, .., \Theta_R^t)$.

A CPT is generated based on the constraint that the sum of probabilities for different states of the random variable should be equal to 1 for a parent instantiation. A CPT is given by: $\Theta_j^t = (\theta_1^t, \theta_2^t, ..)$ where $\theta_l^t \in [0, 1]$ denotes a probability value for a particular state given a parent instantiation.

3.3.2 GAEM: The Algorithm

We now discuss GAEM, see pseudocode in Figure 3.1. An individual ρ is created by initializing its random variables with randomly generated initial probabilities for the CPT, observing the sum constraint. Let n_p denote the population size, and $\boldsymbol{\rho}^t$ denote the bag of n_p individuals at the t-th generation: $\boldsymbol{\rho}^t = (\rho_1^t, \rho_2^t, .., \rho_{n_p}^t)$. Let n_g denote the number of generations. The number of iterations reached by the i-th EM run is given by $\eta(\rho_i^t)$. An EM run is converged when it has the reached the maximum number of iterations ω or when the relative difference in log-likelihood between successive iterations is less than ϵ. A learned individual $\rho_i^{t'}$ is produced when the EM algorithm (line 24 in Figure 3.1) is run on an individual ρ_i^t until convergence. For each GA generation, the learned individuals form a bag $\boldsymbol{\rho}^{t'} = (\rho_1^{t'}, \rho_2^{t'}, .., \rho_{n_p}^{t'})$. Crossover and mutation (lines 31, 32 and 34 in Figure 3.1) are applied to $\boldsymbol{\rho}^{t'}$ to create offspring $\boldsymbol{\rho}^{t'''}$.

Consider an example BN with $R = 5$ random variables. We now describe the crossover, mutation, and replacement mechanisms of GAEM for this BN.

Crossover The GA selects two learned individuals randomly from the learned individual set $\boldsymbol{\rho}^{t'}$ as parents. Let $\rho_a^{t'}$ and $\rho_b^{t'}$ be two individuals from the parent population: $\rho_a^{t'} = (\Theta_{a1}^{t'}, \Theta_{a2}^{t'}, \Theta_{a3}^{t'}, \Theta_{a4}^{t'}, \Theta_{a5}^{t'})$ and $\rho_b^{t'} = (\Theta_{b1}^{t'}, \Theta_{b2}^{t'}, \Theta_{b3}^{t'}, \Theta_{b4}^{t'}, \Theta_{b5}^{t'})$ We apply a single point crossover. Let $c \in \{1, 2, 3, 4\}$ be a random crossover point. Crossover happens with probability p_c. The resulting individuals

```
1:  procedure GAEM($\boldsymbol{\rho}^0, n_g, n_p, p_c, p_m, \alpha, \omega, \epsilon$)
2:      $t \leftarrow 1$                                   ▷ Initialize population
3:      for $i \leftarrow 1, n_p$ do
4:          $\boldsymbol{\rho}^t \leftarrow \boldsymbol{\rho}^t \cup \rho_i^t$
5:      end for
6:      $\boldsymbol{\rho}^{t'} \leftarrow$ POPEM($\boldsymbol{\rho}^t$)           ▷ Compute learned individuals
7:      repeat
8:          $\boldsymbol{\rho}^{t'''} \leftarrow$ GA($\boldsymbol{\rho}^{t'}$)
9:          if $\alpha$ = GAEM-ALEM & $t \geq 2$ then
10:             $\boldsymbol{\rho}^{t+1'} \leftarrow$ GAEM-ALEM($\boldsymbol{\rho}^{t'''}$)
11:         else
12:             $\boldsymbol{\rho}^{t''''} \leftarrow$ POPEM($\boldsymbol{\rho}^{t'''}$)
13:         end if
14:         if $t \geq 2$ then
15:             $\boldsymbol{\rho}^{t+1'} \leftarrow$ REPLACE($\boldsymbol{\rho}^{t'}, \boldsymbol{\rho}^{t''''}, \alpha$)
16:         end if
17:         $\rho^* \leftarrow$ SELECTBEST($\boldsymbol{\rho}^{t+1'}, \rho^*$)
18:         $t \leftarrow t + 1$
19:     until $t > n_g$
20:     return $\rho^*$
21: end procedure
22: procedure POPEM($\boldsymbol{\rho}^t$)
23:     for $i \leftarrow 1, n_p$ do
24:         $\rho_i^{t'} \leftarrow$ EM($\rho_i^t, \omega, \epsilon$)    ▷ EM till convergence
25:         $\boldsymbol{\rho}^{t'} \leftarrow \boldsymbol{\rho}^{t'} \cup \rho_i^{t'}$
26:     end for
27:     return $\boldsymbol{\rho}^{t'}$
28: end procedure
29: procedure GA($\boldsymbol{\rho}^t$)
30:     for $\rho_i^{t'}, \rho_j^{t'} \leftarrow \boldsymbol{\rho}^{t'}$ do
31:         $(\rho_i^{t''}, \rho_j^{t''}) \leftarrow$ CROSSOVER($\rho_i^{t'}, \rho_j^{t'}, p_c$)
32:         $\rho_i^{t'''} \leftarrow$ MUTATION($\rho_i^{t''}, p_m$)
33:         $\boldsymbol{\rho}^{t'''} \leftarrow \boldsymbol{\rho}^{t'''} \cup \rho_i^{t'''}$
34:         $\rho_j^{t'''} \leftarrow$ MUTATION($\rho_j^{t''}, p_m$)
35:         $\boldsymbol{\rho}^{t'''} \leftarrow \boldsymbol{\rho}^{t'''} \cup \rho_j^{t'''}$
36:     end for
37:     return $\boldsymbol{\rho}^{t'''}$
38: end procedure
39: procedure GAEM-ALEM($\boldsymbol{\rho}^{t'''}$)
40:     $\boldsymbol{\rho}^{t''''} \leftarrow$ STARTEM($\boldsymbol{\rho}^{t'''}$)
41:     while $|\boldsymbol{\rho}^{t''''}| \neq 0$ do
42:         for $\rho_i^{t''''} \leftarrow \boldsymbol{\rho}^{t''''}$ do
43:             if $\eta(\rho_i^{t''''}) > n$) then
44:                 STOPEM($\rho_i^{t''''}$)
45:                 if ( $f(\rho_i^{t''''}) < f(\rho_i^{t'})$ ) then
46:                     $\boldsymbol{\rho}^{t+1'} \leftarrow \boldsymbol{\rho}^{t+1'} \cup \rho_i^{t'}$
47:                     $\boldsymbol{\rho}^{t''''} \leftarrow \boldsymbol{\rho}^{t''''} - \rho_i^{t''''}$
48:                 else
49:                     RESUMEEM($\rho_i^{t''''}$))
50:                 end if
51:             else
52:                 if ($\kappa(\rho_i^{t''''}, \omega, \epsilon)$) then
53:                     $\boldsymbol{\rho}^{t+1'} \leftarrow \boldsymbol{\rho}^{t+1'} \cup \rho_i^{t''''}$
54:                     $\boldsymbol{\rho}^{t''''} \leftarrow \boldsymbol{\rho}^{t''''} - \rho_i^{t''''}$
55:                 end if
56:             end if
57:         end for
58:     end while
59:     return $\boldsymbol{\rho}^{t+1'}$
60: end procedure
```

Figure 3.1: The GAEM algorithm, with inputs: initial bag of EM runs ($\boldsymbol{\rho}^0$), number of generations (n_g), population size (n_p), crossover probability (p_c), mutation probability (p_m), iterations after which comparison is performed in the GAEM-ALEM method (n), the name of the replacement method (α), iteration threshold (ω), and LL tolerance (ϵ). We include pseudocode for GAEM-ALEM. Other replacement methods are discussed in Section 3.3.3. The procedure PopEM runs EM on a population until convergence; StartEM starts EM on a population; StopEM stops an EM run; and ResumeEM resumes an EM run. The procedure κ checks convergence of an EM individual. The output of GAEM is the EM run with the highest LL found.

($\rho_i^{t''}$ and $\rho_j^{t''}$) are the children (line 31 in Figure 3.1). For instance, if $c = 2$, then the children of $\rho_a^{t'}$ and $\rho_b^{t'}$ are: $\rho_a^{t''} = (\Theta_{a1}^{t''}, \Theta_{a2}^{t''}, \Theta_{b3}^{t''}, \Theta_{b4}^{t''}, \Theta_{b5}^{t''})$ and $\rho_b^{t''} = (\Theta_{b1}^{t''}, \Theta_{b2}^{t''}, \Theta_{a3}^{t''}, \Theta_{a4}^{t''}, \Theta_{a5}^{t''})$

Mutation The purpose of mutation is to introduce diversity. Mutation helps to avoid premature convergence and gives a broader exploration of the search space. Mutation alters the values of one or more CPTs in an individual (lines 32 and 34 in Figure 3.1). For example, mutation in individual $\rho_b^{t''}$ happens with probability p_m at the CPT $\Theta_i^{t''}$ level. Mutation replaces the random variable's CPT $\Theta_i^{t''} = (\theta_1^{t''}, \theta_2^{t''}, ..)$ with randomly chosen values. Specifically, $\Theta_{a4}^{t'''}$ is mutated by choosing random values satisfying the probability constraint that the state probabilities sum to 1 for a given parent instantiation. The mutated individual $\rho_b^{t'''} = (\Theta_{b1}^{t''}, \Theta_{b2}^{t''}, \Theta_{a3}^{t''}, \Theta_{a4}^{t'''}, \Theta_{a5}^{t''})$. If mutation is applied on all random variables, then the resulting EM individual will be similar to a new random starting point and the EM algorithm will take more number of iterations to converge. Hence, it is useful to apply mutation on only a few random variables.

Replacement The replacement mechanism picks individuals from two sets of learned individuals ($\boldsymbol{\rho}^{t'}$ and $\boldsymbol{\rho}^{t''''}$) to form parents for the next generation. The comparison is done using the fitness f of learned individuals (line 45 in Figure 3.1). We define fitness as $f(\rho_i)$ = LL(ρ_i). The learned individuals of the first generation ($t = 1$) are $\boldsymbol{\rho}^{1'} = (\rho_1^{1'}, \rho_2^{1'}, .., \rho_{n_p}^{1'})$. The set of learned individuals in $\boldsymbol{\rho}^{1'}$ undergo crossover and mutation as explained above and generate offspring $\boldsymbol{\rho}^{1'''} = (\rho_1^{1'''}, \rho_2^{1'''}, .., \rho_{n_p}^{1'''})$. The EM algorithm is run on each individual in $\boldsymbol{\rho}^{1'''}$ to generate a set of learned individuals $\boldsymbol{\rho}^{1''''}$. Now, the two sets of learned individuals ($\boldsymbol{\rho}^{1'}$ and $\boldsymbol{\rho}^{1''''}$) will undergo replacement and the parents for the third generation are selected (lines 9 and 14 in Figure 3.1).

3.3.3 Replacement Mechanisms

We studied four different replacement mechanisms: traditional replacement, deterministic replacement, probabilistic replacement and ALEM-based replacement. We now explain each of them.

Traditional Replacement (GAEM-TRAD) Using this method, we compare the fitness of each parent $\rho_i^{t'}$ with its child $\rho_i^{t''''}$ and select the individual that has higher fitness (during crossover, a child is tied to the parent from which it gets the first part). Using the above example, we first compare the fitness of $\rho_1^{1'}$ with $\rho_1^{1''''}$, then compare the fitness of $\rho_2^{1'}$ with $\rho_2^{1''''}$, and so on. If the parent $\rho_i^{1'}$ is best, then it replaces its corresponding child $\rho_i^{1''''}$, otherwise we pick the child. This method illustrates a competition within the family, where one parent competes with only one child.

Deterministic Replacement (GAEM-DETER) This method is based on deterministic crowding [63], where an offspring competes against its most similar parent. The distance $KL(\rho_a^{t'}, \rho_a^{t''''})$ is computed using KL divergence [54]. Under this replacement scheme, each offspring competes for survival with its most similar parent. The winner is the individual with the highest fitness. This crowding scheme can be used to efficiently preserve diversity in the population.

Probabilistic Replacement (GAEM-PC) Probabilistic crowding uses a non-deterministic rule to establish the winner of a competition between parent and child [67]. The probability that a parent ($\rho_a^{t'}$) replaces a child ($\rho_a^{t''''}$) is: $P_{\rho_a^{t'}} = \frac{f(\rho_a^{t'})}{f(\rho_a^{t'})+f(\rho_a^{t''''})}$. In this replacement mechanism, the fitter individuals do not always win over the weaker individuals, they win proportionally according to their fitness. This strategy helps to preserve diversity in the population.

ALEM based Replacement (GAEM-ALEM) This technique is inspired by the age-layered EM (ALEM) technique, where EM runs compete within age brackets [94]. Age is interpreted as the number of EM iterations. The assumption is that the probability of a "young" and poorly performing EM run (relative to its age bracket) turning into an "old" but strongly performing run (relative to its age bracket) is low. Such a "young" and poorly performing EM run can thus be discarded early to save CPU cycles for other, better performing EM runs.[1] The EM algorithm shows a rapid progress during the initial EM iterations, so traditional accelerating methods start the optimization with EM and then move to accelerating methods when we are close to a solution [103], [73]. We incorporated these ideas in the GAEM-ALEM() Pseudocode in Figure 3.1. After an initial n iterations in GAEM, the fitness of a child EM run is compared with the fitness of its parent EM run at each iteration (lines 43 and 45 in Figure 3.1) . If the parent has a higher fitness than the child, then parent replaces child. If the child has a higher fitness, then the child is allowed to iterate but the comparison check takes place at each EM iteration of the child. The final set of individuals ($\boldsymbol{\rho}^{t+1'}$) form the parents for the next generation (line 59 in Figure 3.1).

3.4 Analysis

Using finite Markov chain theory, it has been shown that the canonical GA converges to the global optimum when the best solution is maintained in the population [91]. We will prove the convergence of GAEM along similar lines as an earlier proof [53], namely by deriving conditions for the parameters so that the convergence to global optimum is feasible. Some definitions from matrix theory are given below:

Definition 3.4.1. A square matrix A is positive if its elements $a_{ij} > 0 \; \forall \, i, j \in \{1, 2, ..., m\}$. A square matrix A is primitive if there exists a $k \in \mathbb{N}$ such that A^k is positive. A square matrix is column allowable if there exists at least one positive entry in each column. If the elements of a

[1]In ALEM, EM runs with similar *age* (i.e., the number of iterations) are grouped together in layers. When an EM run's age exceeds upper limit of its current layer, it is moved up to the next layer if it can replace a poor performing individual in the target layer, otherwise it is discarded. The introduction of *age* and *layer* ensures a fair competition among similar EM runs. In GAEM-ALEM, a special case of age layering is used where the size of first layer is set to n iterations. The size of all other layers are small with 1 iteration. The number of layers is not a fixed value as in ALEM, but it is equal to the number of iterations taken by an EM run. Each child EM run is compared with it's parent EM run only and not with other EM runs as in the case of ALEM.

square matrix A, $a_{ij} \in [0,1]$ and $\sum_{j=1}^{m} a_{ij} = 1$, then it is a stochastic matrix.

In GAEM, the population of individuals at generation t is denoted by $\boldsymbol{\rho}^t$ (see Section 3.3.1). A CPT is generated based on the constraint that the sum of probabilities of the states of a random variable should be equal to 1 for a given parent instantiation. We define the state space S of this problem as the space of all possible random starting points satisfying the CPT constraint. The state space can be numbered from 1 to $|S|$. In GAEM, the process of generating a population can be denoted by a random variable Y, $Y(t) = {\boldsymbol{\rho}^t}_{t\geq 0}$ is dependent only on the previous population $Y(t-1)$ at generation $(t-1)$. Hence $Y(t) = {\boldsymbol{\rho}^t}_{t\geq 0}$ is a Markov chain and is denoted by:

$$\Pr(Y(t) = \boldsymbol{\rho}^t | Y(t-1) = \boldsymbol{\rho}^{t-1}, \ldots, Y(0) = \boldsymbol{\rho}^0) \;=\; Pr(Y(t) = \boldsymbol{\rho}^t | Y(t-1) = \boldsymbol{\rho}^{t-1}).$$

The transition probabilities p_{ij} are defined as the probability of generating population $\boldsymbol{\rho}^i$ from $\boldsymbol{\rho}^j$: $p_{ij}(t) \;=\; Pr(Y(t) = \boldsymbol{\rho}^i | Y(t-1) = \boldsymbol{\rho}^j)$. These transition probabilities are independent of the time instant, *i.e.*, $p_{ij}(s) = p_{ij}(t)$ for all $\boldsymbol{\rho}^i, \boldsymbol{\rho}^j \in S$ and for all $s, t \geq 1$. Therefore, $Y(t) = {\boldsymbol{\rho}^t}_{t\geq 0}$ is a time-homogeneous finite Markov chain. Let $P = \{p_{ij}\}$ be the transition matrix of the process $Y(t) = {\boldsymbol{\rho}^t}_{t\geq 0}$. The entries of the matrix P satisfy $p_{ij} \in [0,1]$ and $\sum_{j=1}^{|S|} p_{ij} = 1 \; \forall i \in S$. So P is a stochastic matrix. For GAEM to converge to a global maximum, it is required that P be a primitive matrix. We investigate the operators that make the matrix P primitive. The probabilistic changes of the individuals during GAEM operation can be captured by the transition matrix P, which can be decomposed into a product of stochastic matrices, $P = C * M * R * L$, where C, M, R, and L describe the intermediate transitions caused by crossover, mutation, replacement, and local search (EM learning) respectively. Each element of the matrices C, M, R and L represent the transition probability of generating a new individual $\rho_i^{t+1'}$ from a parent individual $\rho_i^{t'}$ on the application of crossover, mutation, replacement, and local search (EM) mechanisms respectively. Since every positive matrix is primitive, it is therefore enough to find the conditions which make C and M positive, and R and L column-allowable.

Proposition: Let C, M, R and L be stochastic matrices, where C and M are positive[2] and R and L are column allowable. Then the product $P = C * M * R * L$ is positive.

Crossover: An individual $\rho_i^{t''}$ can be obtained from the individual $\rho_i^{t'}$ on application of the crossover operator. Each state of S is mapped probabilistically to another state, so C is stochastic.

Mutation: The matrix M is positive if any individual $\rho_i^{t'''}$ can be obtained from an individual

[2]The crossover and mutation probabilities are assumed to be greater than zero ($p_c > 0$ and $p_m > 0$).

$\rho_i^{t''}$ on application of the mutation operator. The mutation is done by taking into account the constraint on CPT as discussed in Section 3.3.

Replacement: The matrix R is column allowable, *i.e.*, there is at least one positive entry in each column of the matrix. The replacement process does not alter the CPTs of the individual. This matrix takes into account the probability of survival of each individual in the current population, which depends on the fitness (log likelihood) value of the individual. So there is only one positive entry, r_{ij} when $i = j$. The survival probability can be bounded as shown: $r_{ii} \geq \frac{f(\rho_i^t)}{\sum_{i=1}^{N} f(\rho_i^t)} \geq 0,\ \forall i \in S$. Even though this bound changes with the generations, it is always strictly positive. Hence, R is column allowable.

Local search: The matrix L is column allowable, *i.e.*, there is at least one positive entry in the matrix. The local search is done by applying the EM algorithm, which is a deterministic procedure. When an individual ρ_i^t undergoes EM learning, it will always converge to the same new individual $\rho_i^{t'}$. So there is only one positive entry per column, $l_{ij} = 1$.

Theorem: Let $X(t) = \max\{f(\rho_i^t)|i = 1, \ldots, n_p\}$ be a sequence of random variables representing the best fitness within a population $\boldsymbol{\rho}^t$ at generation t. Let the matrices C and M be positive and R and L be column allowable as described above. Then,

$$\lim_{t\to\infty} \Pr(\{X(t) = f^*\}) = 1,$$

where $f^* = \max\{f(\rho_i^t)\,|\rho_i^t \in S\}$ is the global optimum.

Proof Sketch: It is proved that a canonical genetic algorithm, which maintains the best solution found so far, converges to the global optimum [91]. Under the hypothesis of the theorem, the transition matrix P of GAEM is primitive as discussed above. From the pseudocode in Figure 3.1, it can be seen that the best solution found in the generation is maintained.[3] Thus, the theorem follows from [91, Theorem 6]. $\square$

3.5 Experiments

Local search helps to fine tune the solution whereas global search is more exploratory. We start GA on a population of already converged EM individuals. The randomness in GA is introduced by its operators. This randomness helps to move an already converged EM individual to a new position in the search space. GA's helps in finding a better starting point for the local search

[3]The best solution is different for different replacement mechanisms.

(EM) in the converged region of the search space. We investigate the role of GA's operators such as mutation, crossover and replacement mechanisms.

We have done experiments to understand the role of GA's stochastic operators such as mutation, crossover and replacement mechanisms. We also study the role of population size in GAEM. Before we start with the experiments on GAEM, we do a search space analysis of four different BNs and classify them into hard and easy search spaces. All the experiments were performed for hard and easy search spaces.

In all sets of experiments, we set the maximum number of iterations that the EM algorithm can undergo a maximum of 1000 iterations, and the log likelihood difference tolerance (i.e. if the difference in log likelihood between two iterations is less than this tolerance, than we deem the EM run to have converged or terminated) to 0.00001.

3.5.1 Datasets and Evaluation

We performed experiments on four Bayesian Networks (BN) of different sizes and structures, the Carstarts with 18 nodes, Alarm with 37 nodes, Hepar2 with 70 nodes and Win95pts with 76 nodes. The Carstarts BN [36] is based on the various operations in a car. The more complex Alarm BN [9] represents a real life situation for monitoring a patient in intensive care unit. Hepar2 BN [83] is used for diagnosing liver disorders. The win95pts BN [41] is created for printer troubleshooting in Windows 95. We collected these BNs from the bnlearn repository. [4]

Through Gibbs sampling, we generated $n_s = 500$ samples with the 19, 7, 35 and 38 hidden variables for the alarm, carstarts, hepar2 and win95pts BNs respectively. For the alarm and carstarts BN, we chose these hidden variables based on our previous experiments where we found that with these hidden variables, EM took more number of average iterations to converge [94]. For win95pts and hepar2, we hid 50% of the random variables. If m represents the number of hidden variables in a hidden variable configuration, then for each configuration we randomly chose m variables to hide. We implemented GAEM by interfacing with the libDAI graphical models C++ library [72], which contains an EM implementation oriented towards parameter estimation in Bayesian networks, and the Boost multithreading library to process the EM runs in parallel.

[4]http://www.bnlearn.com/bnrepository/

For the GAEM experiments, we generated EM runs by randomly choosing starting points in the search space for each Bayesian network, in this case starting conditional probability distribution values, for each of the random variables in the network obeying the CPT constraint as described in Section 3.3. These EM runs are the initial individuals in the population. The number of EM runs varies based on the experiments. The GAEM experiment determines the parent individuals for the next generation based on crossover, mutation and replacement mechanisms described in Section 3.3.

3.5.2 Search Space Analysis

To better understand the structure of the BN parameter search spaces, we first study the spread of LL values for the traditional EM algorithm at convergence. In this experiment, we used eight BNs: Carstarts, Child, Sprinkler, Insurance, Win95pts, Alarm, Hepar2, and Hailfinder. For each BN, we executed $n_p = 200$ traditional EM runs and used a sample size of 500. Distances are calculated using d_i = LL* - LL$_i$, where LL* is the maximum LL (max_LL) seen among the 200 EM runs and LL$_i$ is the LL of the i-th EM run. On the left side of Figure 3.2 shows the spread of distances for the experimental BNs when $n_p = 200$.

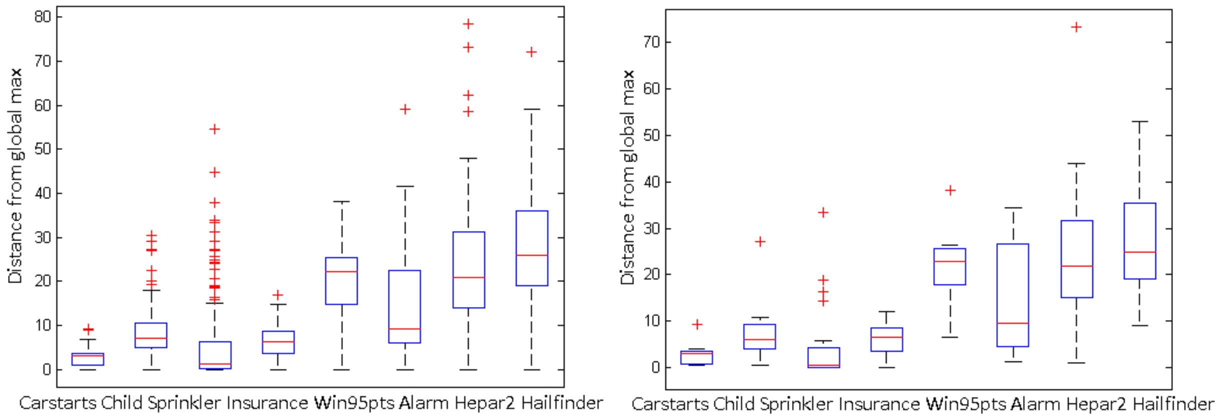


Figure 3.2: Box plots (or box and whisker diagrams) reflecting the spread of log-likelihoods for different BNs. For each BN, the plot is for $n_p = 200$ runs on the left side and $n_p = 20$ runs on the right side, at convergence, of the traditional EM algorithm. We can see that the spread of log-likelihoods is similar for the BNs in both figures.

The red line in Figure 3.2 denotes the median; everything above or below it represents the distance of 50% of the EM runs. The size of the box denotes the spread of the LL distances around the median. For Carstarts, Child, Sprinkler, and Insurance, the median is close to the max_LL. In Win95pts, the spread above the median is low, implying that 50% of EM runs are closer to the max_LL. Thus, we consider Carstarts, Child, Sprinkler, Insurance, and Win95pts as easy search spaces. For Alarm, Hepar2, and Hailfinder, the spread above the median is high and 50% of the EM runs are away from the max_LL. Thus, we consider these BNs as hard search spaces. We ran the same experiment for each BN for $n_p = 20$ traditional EM runs. The results are shown on the right side of Figure 3.2, where the spread of log likelihoods is similar to the left side plot when $n_p = 200$ EM runs. From these results, we conclude that for classifying a new BN as hard or easy search spaces, the user can run a few (such as $n_p = 20$) set of traditional EM runs and draw box plots as shown in Figure 3.2 to compare the spread of log likelihoods with the existing BNs.

3.5.3 Role of Population Size

With the above understanding of search spaces for each BN, we investigate the role of varying population sizes. Our goal is to understand the effect of global search (GA) and local search (EM) under varying population sizes. We did two experiments, one to study the GA's performance when EM (the number of EM iterations) is kept constant and another in which EM's performance is studied with GA (the number of generations) kept as constant.

In the first experiment, we investigate the GA's role. The efficiency of GA depends on different parameters such as number of generations, population size, mutation probability, crossover probability and replacement mechanism. We vary the population size for a constant number of generations. All other parameters are kept as constant.

For the second experiment, we study the role of EM. The efficiency of EM algorithm is measured by the number of iterations taken by an EM individual when the EM algorithm is allowed to iterate until convergence. So the number of generations is allowed to increase till we reach a constant number of EM iterations,for example, $n_g \in \mathbb{N}$ and $n_{GAEM} = 250$. We vary the population size for constant set of EM iterations. In both the experiments, mutation probability and crossover probability is set as $p_m = 0.05$ and $p_c = 0.5$ respectively. GAEM-TRAD replacement mechanism is used. We investigate the effect of varying population sizes on the maximum log likelihood value seen so far in the generations.

Effect of Population Size with Fixed Number of Generations

In this experiment, we study the effect of EM on varying population sizes. We run the GAEM algorithm for generations, $n_g \in \{2, 4, 6, 8, 10\}$ for the population sizes $n_p \in \{2, 4, 8, 16, 32\}$ on four different BNs. Within each generation, EM is run on each individual until convergence. So the effect of EM is more on larger population sizes. The BNs used are alarm, carstarts, hepar2 and win95pts. The results are shown in Figure 3.3. On the left side of Figure 3.3, we show four graphs, one for each BN, where the number of iterations is shown as a function of generations. The number of iterations is cumulative, that is it is the total number of iterations that the GAEM took till that generation. We can see a trend for all the four BNs, where the number of iterations increases with population size. This result is expected, as we know that larger population sizes undergo more number of EM iterations per generation compared to smaller population sizes.

On the right side of Figure 3.3, we show four graphs, one for each BN, where maximum log likelihood is shown as a function of number of generations. For alarm BN, we can see that smaller population sizes, $n_p = 2$ or $n_p = 4$, reach a poorer solution quality compared to larger population sizes, $n_p = 32$ or $n_p = 16$. Similarly for hepar2 BN, a smaller population size of $n_p = 2$ reached a poor solution quality compared to other population sizes. These results shows that for harder search spaces, a larger population size is needed to reach higher solution quality. Larger population tends to push GA to explore new regions, thus helping the individuals to get out the local maxima. But a larger population undergoes more EM iterations, so it is recommended to use an optimum population size such as $n_p = 4$ or $n_p = 8$ for hard search spaces. For carstarts and win95pts BN, we see that all population sizes converge to approximately the same solution quality. We presume that this behavior is due to the easier search space structure compared to alarm and hepar2 BN. It is recommended to use a small population size when the BN search space structure is easier, since a small population size undergoes fewer EM iterations as shown on the left side of Figure 3.3.

Effect of Population size with Fixed Number of Iterations

The goal of this experiment is to understand the impact of GA on varying population sizes for constant computational resource. We run the GAEM algorithm for a fixed number of EM iterations, $n_{GAEM} \in \{250, 500, 750, 900, 1250\}$ on different population sizes $n_p \in \{2, 4, 8, 16, 32\}$.

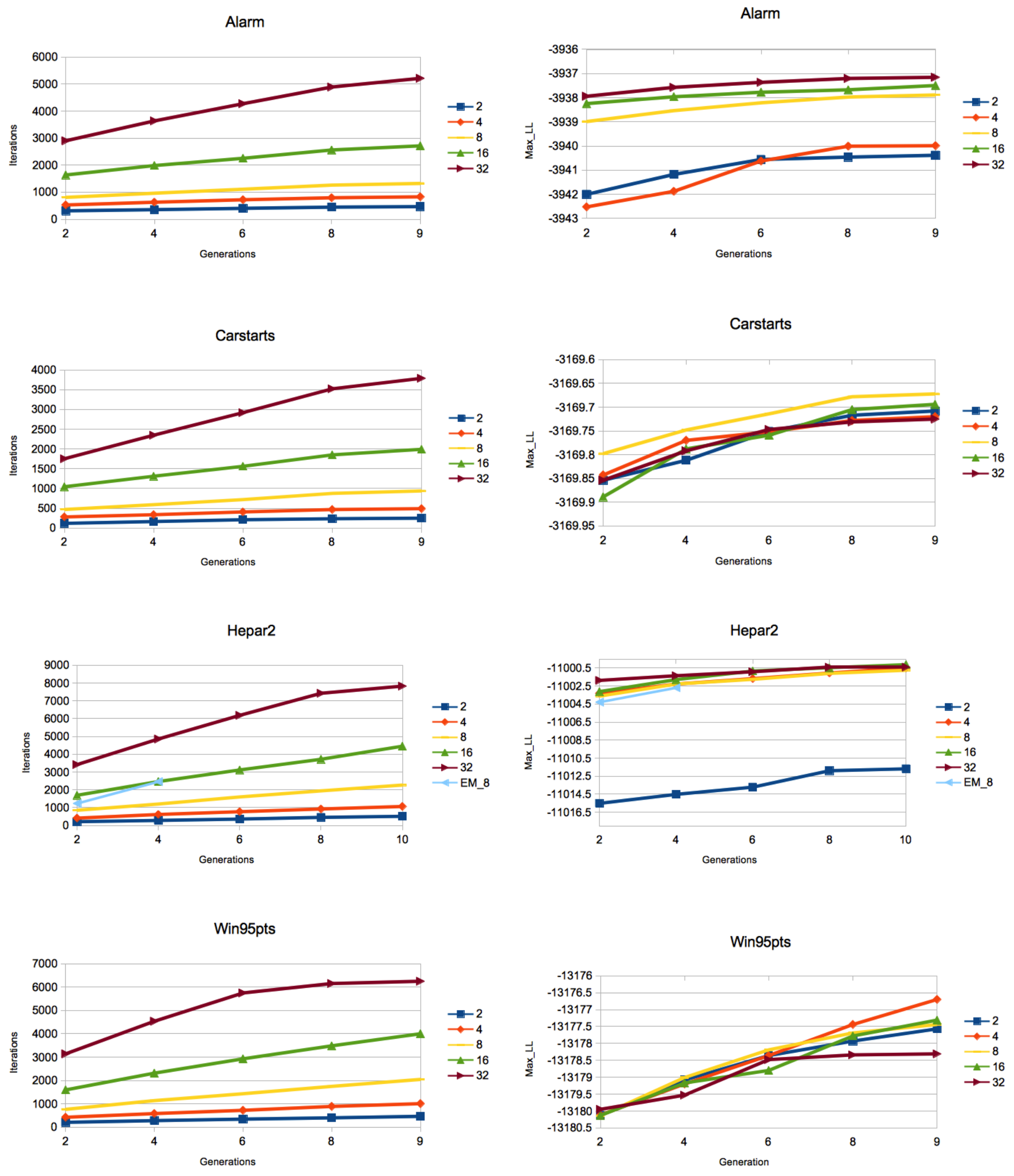


Figure 3.3: Effect of populations size on different BNs when GAEM is run for fixed number of generations, $n_g \in \{2, 4, 6, 8, 10\}$, shown on the x-axis. On the y-axis, in the left column we show the cumulative number of EM iterations, in the right column we show the maximum log likelihood.

Each GAEM run is allowed to undergo any number of generations till it reaches a fixed number of EM iterations. So the effect of the GA is greater on small population sizes, since the individuals have to undergo many generations to reach the constant EM iterations. For the larger population sizes, the effect of GA gradually decreases as we increase the population size. We tried four different BNs, alarm, carstarts, hepar2 and win95pts. The results are shown in Figure 3.4.

On the left side of Figure 3.4, we show the results of number of generations as a function of the number of iterations, for four different BNs. As expected, we can see a trend where the small population sizes undergo more number of generations to reach the fixed EM iterations. On the right side of Figure 3.3, we show maximum log likelihood (max_LL) as a function of the number of iterations. For each iteration in the x-axis, we show the corresponding maximum log likelihood seen so far in the population. During one GAEM run, the total number of iterations taken by a small initial population size of $n_p = 2$ is lesser compared to the total number of iterations for a large population size of $n_p = 32$. We see that for a smaller initial population size (for eg. $n_p = 2$), it takes more number of generations (GA search) to reach a target number of iterations (for eg. $n_{GAEM} = 1000$). This shows that GA search is doing more exploration compared to EM search. For harder search spaces such as alarm and hepar2 BN, the solution quality for small population sizes is poor compared to the best solution quality (untill iteration $n_{GAEM} = 1250$ for alarm BN and at all iterations for hepar2 BN). For easier search spaces such as win95pts and carstarts BN, the solution quality for small population sizes is the best at all iterations. Thus, small population sizes are more suitable for easier search spaces.

For a larger initial population size (for eg. $n_p = 32$), we see that it takes fewer of generations to reach a target iteration (for example, $n_{GAEM} = 1000$). In this case, the EM search is applied for a longer time compared to the GA search. For all BNs, the solution quality is poor compared to the best solution quality. For harder search spaces, it is interesting to notice that the best solution quality is obtained for population sizes of $n_p = 8$ and $n_p = 4$ for alarm and hepar2 BN respectively. For hard search spaces, both GA and EM seem to contribute to the exploration of search space. Hence we need to chose a population size where there is a balance between the global and local search. Hence we suggest to use an optimal population for hard search spaces to get high solution quality.

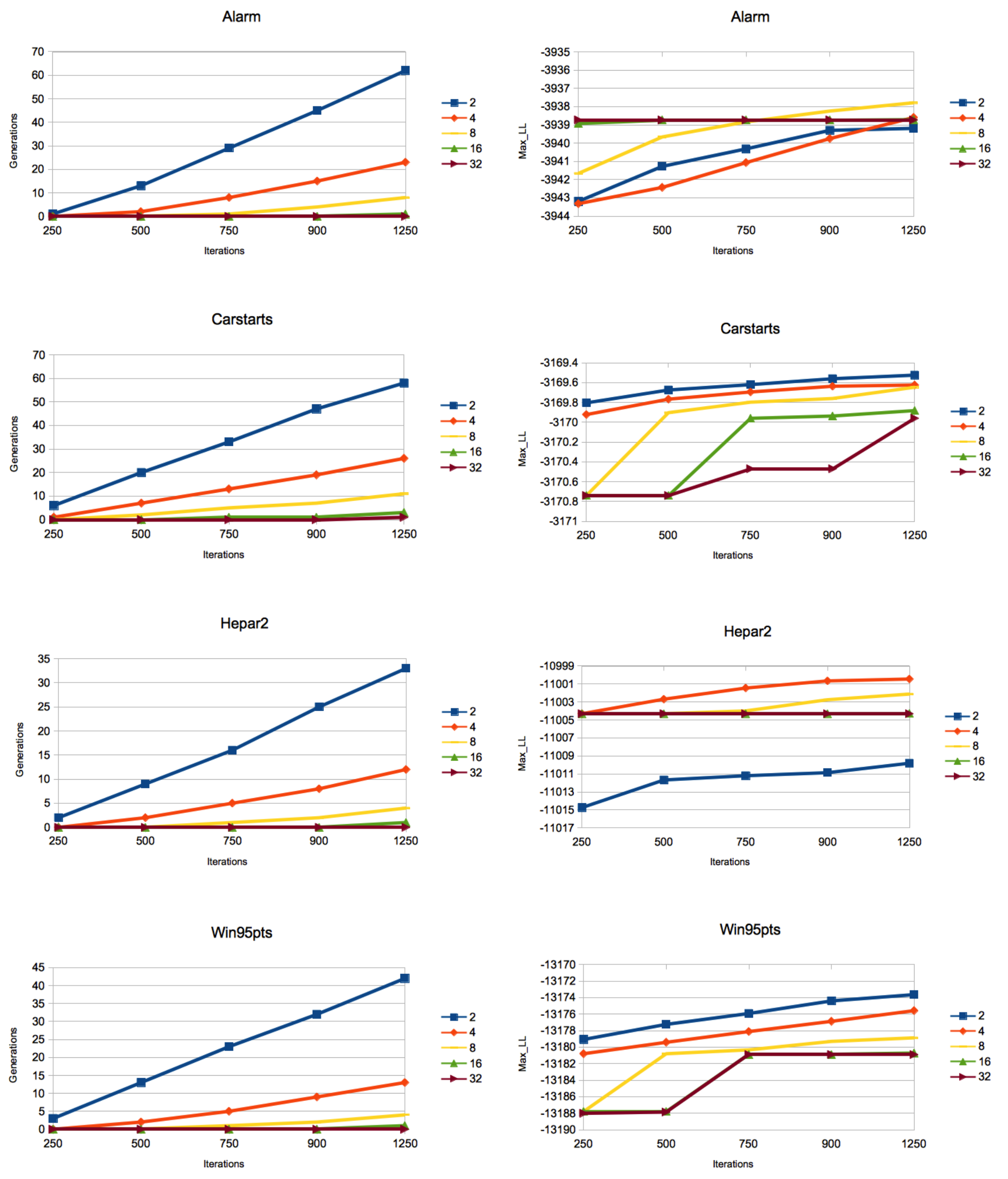


Figure 3.4: Effect of populations size on different BNs when GAEM is run for fixed number of EM iterations, $n_{GAEM} \in \{250, 500, 750, 900, 1250\}$, shown on the x-axis. On the y-axis, in the left column we show the cumulative number of generations, in the right column we show the maximum log likelihood.

From the above two experiments, we conclude that for harder search spaces, a balance between the global search (in terms of the number of generations for GA) and the local search (in terms of the number of iterations) for EM needs to be maintained for a given initial population size. A higher solution quality is shown by that population size. For easier search spaces, small population size with more number of generations is sufficient to reach a high solution quality. Previous works confirms our results where a small population size compared to traditional GAs was found to perform well in hybrid genetic algorithms with high probability of local search [25, 27].

3.5.4 Role of Mutation

If the EM algorithm is run on a converged BN individual, the algorithm does not iterate as it has already converged. But when small changes are introduced in the CPTs of any random variable, then the EM algorithm will iterate till that random variable's CPT is converged. We introduce random changes in the converged CPTs with the help of a mutation operator.

We hypothesize that if a mutation operator is applied to all random variables of the BN individual, then the EM algorithm will take more number of iterations to converge. In this case, the GA search may not help much because the local search is not taking place at the converged region of the search space. It is useful to apply mutation on only a few random variables. We experimented with different mutation probabilities for varying population sizes.

In this experiment, we study the role of mutation on the carstarts and alarm BNs. The carstarts and alarm BNs are chosen as they represent an easy and hard search space respectively. The different mutation probability values used are: $p_m = \{0.05, 0.1, 0.5, 0.9\}$. We analyze the performance of GAEM for these different mutation probability values on different population sizes. The population sizes used are: $n_p \in \{2, 4, 8, 16, 32\}$. Other parameters of GAEM such as crossover probability is set as $p_c = 0.1$ and GAEM-TRAD replacement mechanism is used. The results are based on 10 independent GA runs. We run GAEM for a fixed number of EM iterations and find the maximum log likelihood reached so far in the generations. The constant EM iterations used are: $n_{GAEM} \in \{250, 500, 750, 900, 1250\}$. We allow GAEM to undergo any number of generations till they reach the constant EM iterations. We use the constant number of EM iterations as a measure of comparing the performance of GAEM for different mutation probability values. This is done because the amount of time taken by crossover, mutation and replacement mechanisms is negligible compared to the time taken by the EM iterations. Thus, it is a way to keep the computational resources close to constant.

The results for carstarts and alarm BN are shown in Figure 3.6 and Figure 3.5 respectively. On the left side of Figure 3.6 and Figure 3.5, we show the results of generations as a function of iterations for four different mutation probability values. As expected, we can see a trend where the small population sizes undergo more number of generations to reach the constant EM iterations. For small mutation probabilities, the number of generations taken by the population sizes such as $n_p = \{2, 4, 8\}$ are higher compared to their corresponding number of generations with high mutation probabilities. The reason is high mutation probabilities changes larger portion of the converged CPTs than small mutation probabilities. This results in moving the GAEM to a non converged region of the search space which in turn will lead to more number of EM iterations in lesser number of generations. On the other hand, for population sizes such as $n_p = 16$, the number of generations is almost $n_g < 2$ for all mutation probabilities and for $n_p = 32$, the number of generations, $n_g = 0$. This is due to the fact that the total EM iterations of initial population of individuals reaches the fixed number of EM iterations within a few generations of GAEM. On the right side of Figure 3.6 and Figure 3.5, the graph shows maximum log likelihood as function of iterations.

Easy Search Spaces

For easier search spaces (carstarts BN) as shown in Figure 3.6, we see that low mutation probabilities 0.05 and 0.1 work well on small population size such as $n_p = 2$ and population sizes of $n_p = 4$ or $n_p = 8$. Small population sizes reaches more generations for a given constant EM iterations. Thus a GA dominant search reaches a better solution in easier search spaces. With low mutation probability, a larger population size also performs well. This shows the strength of GAEM for orienting the search towards better quality solution in each generation. For high mutation probability, small population achieve better solution quality than larger population but their solution quality is lower compared to the solution quality of small population with low mutation probability.

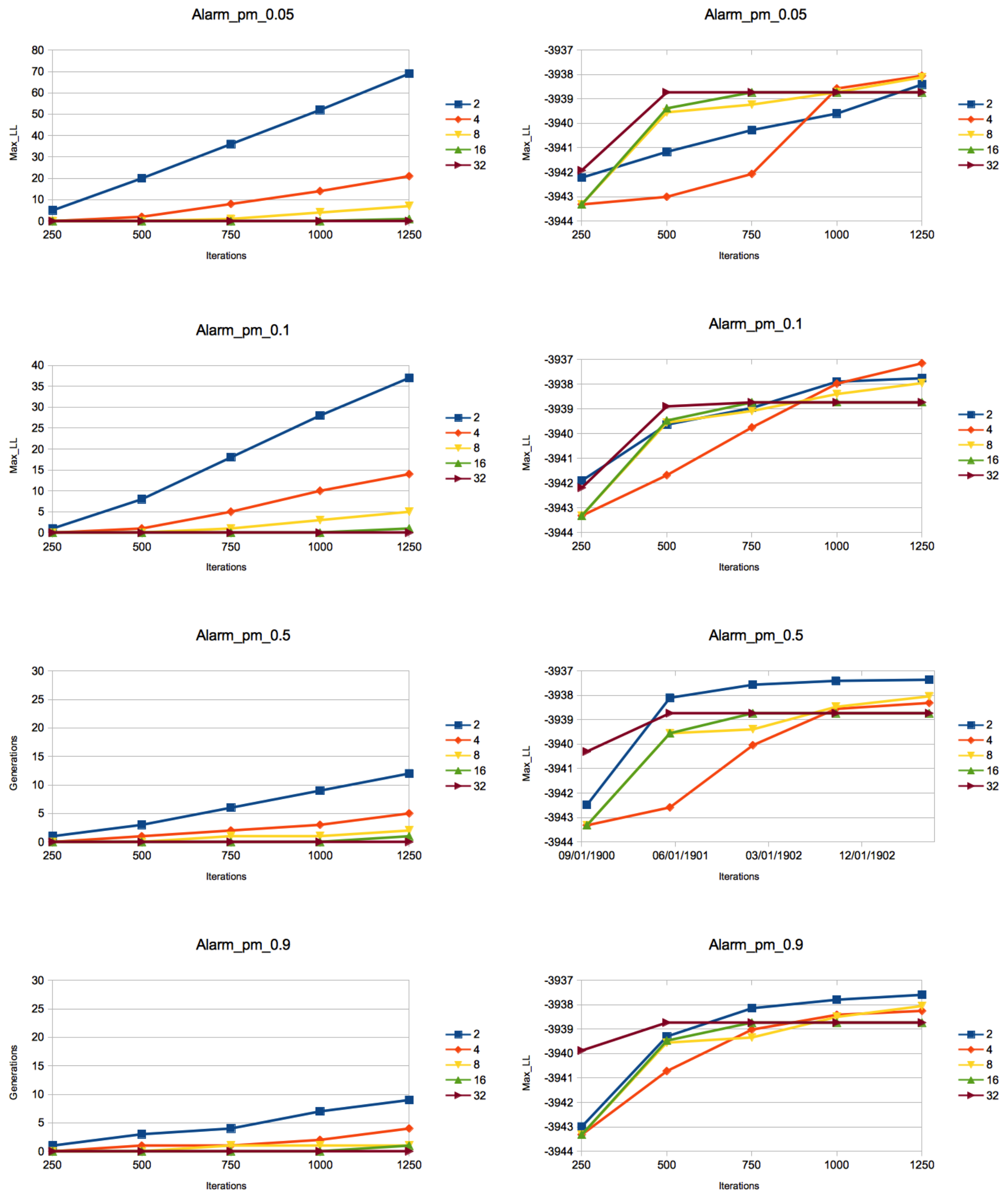


Figure 3.5: Role of mutation on alarm BN (hard search space) when GAEM is run for $n_{GAEM} = \{250, 500, 750, 900, 1250\}$ EM iterations, shown on the x-axis. On the y-axis, in the left column we show the cumulative number of generations, in the right column we show the maximum log likelihood.

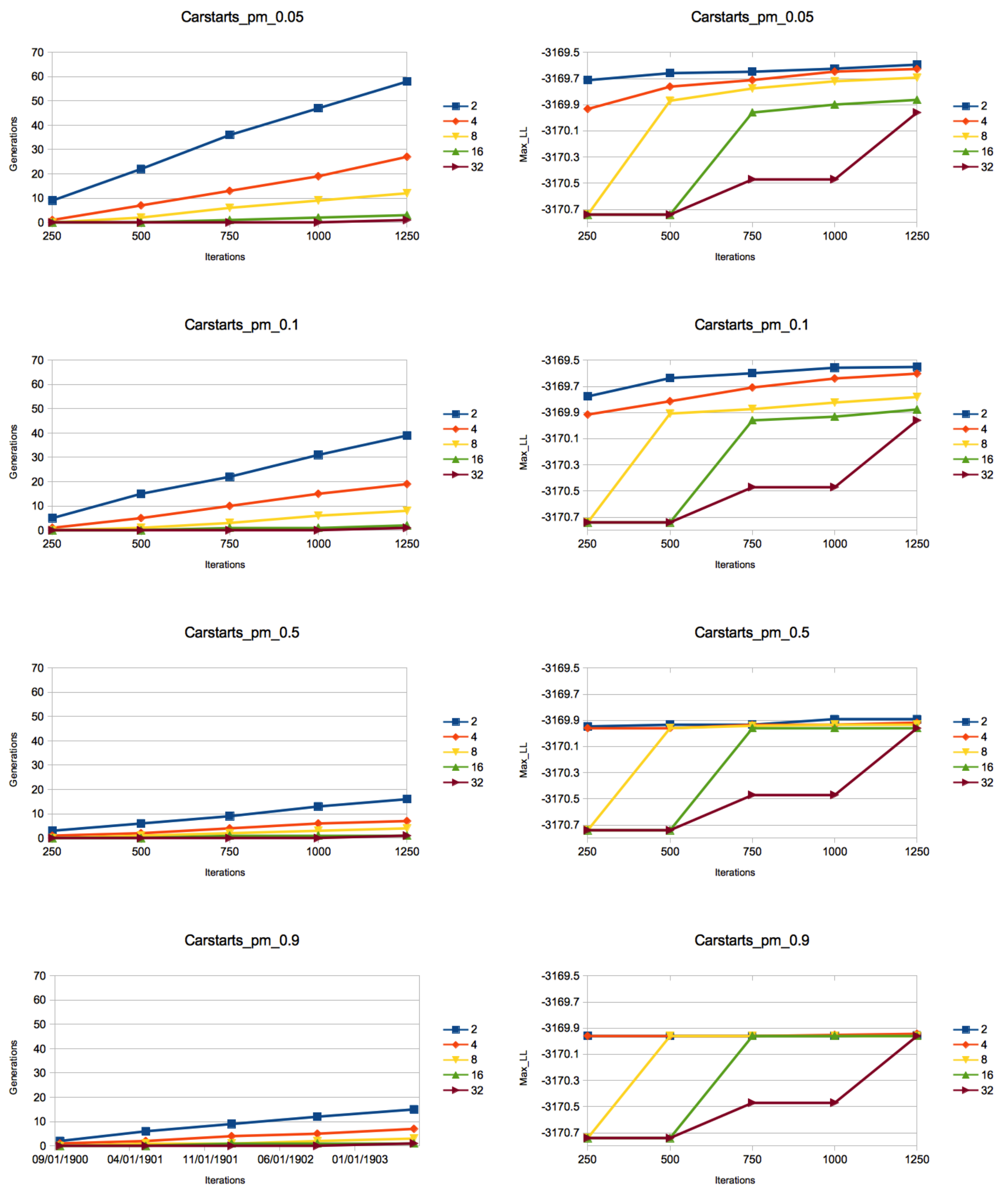


Figure 3.6: Role of mutation on carstarts BN (easy search space) when GAEM is run for $n_{GAEM} = \{250, 500, 750, 900, 1250\}$ EM iterations, shown on the x-axis. On the y-axis, in the left column we show the cumulative number of generations, in the right column we show the maximum log likelihood.

Hard Search Spaces

In Figure 3.5, we can see that for lower mutation probability values ($p_m = 0.05$, $p_m = 0.1$), a small population size of $n_p = 2$, shows a slower progress and achieves poor solution quality. Whereas the population size of $n_p = 4$ reaches the highest solution quality. For higher mutation probability values ($p_m = 0.5$ or $p_m = 0.9$), small population size of $n_p = 2$ achieves a better solution quality compared to other population sizes. The reason for this behavior in hard search spaces is discussed in the below paragraphs.

From our earlier discussion, we know that the low mutation probabilities keeps the local search exploration near the converged region and high mutation probabilities forces EM to start from a non-converged region resulting in more EM iterations. Thus low mutation probability values tend to make the EM search shorter. In general, we require more number generations (GA search) to reach a given iteration value (for example, $n_{GAEM} = 1000$ iterations). With smaller initial population size of $n_p = 2$, more generations (for example, $n_g = 100$) are needed to reach a target iteration (for example, $n_{GAEM} = 1000$) compared to other population sizes. Thus GA search dominates the EM search which in turn leads to a poorer solution quality. With larger population sizes such as $n_p = 16$, fewer generations are sufficient. Larger number of EM individuals results in more EM iterations in fewer generations. Thus EM search dominates the global search. So an initial population size such as $n_p = 4$ or $n_p = 8$, where there is a balance between the GA and EM search produces higher quality solution (as seen in the right side of Figure 3.5, $Alarm\_pm\_0.05$ and $Alarm\_pm\_0.1$)

High mutation probability values tend to make the EM search longer resulting in more iterations. So fewer number generations (GA search) is sufficient to reach a target iteration value (for example, $n_{GAEM} = 1000$ iterations). With a larger initial population size, fewer generations are sufficient to reach a target iteration (for example, $n_{GAEM} = 1000$ iterations can be reached in $n_g = 2$ generations for an initial population size of $n_p = 8$ and $n_p = 1$ generation for an initial population size of 16). So EM search takes more time than the GA search. With smaller initial population size, the number of generations needed to reach a target iteration (for example, $n_{GAEM} = 1000$ iterations can be reached in $n_g = 10$ generations for an initial population size of $n_p = 2$ and $n_g = 6$ generation for an initial population size of $n_p = 4$) is more compared to other larger population sizes. So a balance between the local and global search is achieved for high mutation probability values and small population sizes, resulting in better solution quality.

For larger initial population sizes (such as $n_p = 16$ or $n_p = 32$) mutation probability does not have a effect. Approximately, the same solution quality is obtained for different mutation probability values. So EM search takes longer time resulting in more iterations with fewer generations. Thus the role of GA is limited in large populations. So the mutation probability has only minimal impact on the solution quality.

Conclusion

We conclude that for hard search spaces (alarm BN), low mutation probabilities work well on population size such as $n_p = 4$ or $n_p = 8$ and high mutation probabilities work well for small population sizes such as $n_p = 2$. For easier search spaces, low mutation probability works well with small or optimal initial population sizes. Whereas high mutation probability values produces almost the same solution quality for all population sizes.

3.5.5 Role of Crossover

In this experiment, we analyze the role of crossover on hard and easy search spaces of Bayesian networks for varying population sizes. We take alarm and carstarts as an example of hard and easy search spaces respectively. The population sizes used are: $n_p \in \{2, 4, 8, 16, 32\}$.. The different crossover probability values used are: $p_c = \{0.1, 0.5, 0.9\}$. We keep the number of iterations as constant and allow GAEM to run as many generations till we reach the constant iterations. We run GAEM for a fixed number of EM iterations and find the maximum log likelihood reached so far in the generations. The constant EM iterations used are: $n_{GAEM} \in$ {250, 500, 750, 900, 1250}. We allow GAEM to undergo any number of generations till they reach the constant EM iterations. We use the constant number of EM iterations as a measure of comparing the performance of GAEM for different crossover probability values. This is done because the amount of time taken by crossover, mutation and replacement mechanisms is negligible compared to the time taken by the EM iterations. Thus, it is a way to keep the computational resources close to constant. The results are taken as an average over 10 independent GAEM runs. We used a constant mutation probability of $p_m = 0.05$ and GAEM-TRAD replacement mechanism is used.

On the left side of Figure 3.7 and 3.8, the number of generations is shown as a function of the number of iterations when the crossover probability is varied. We can observe that for both hard

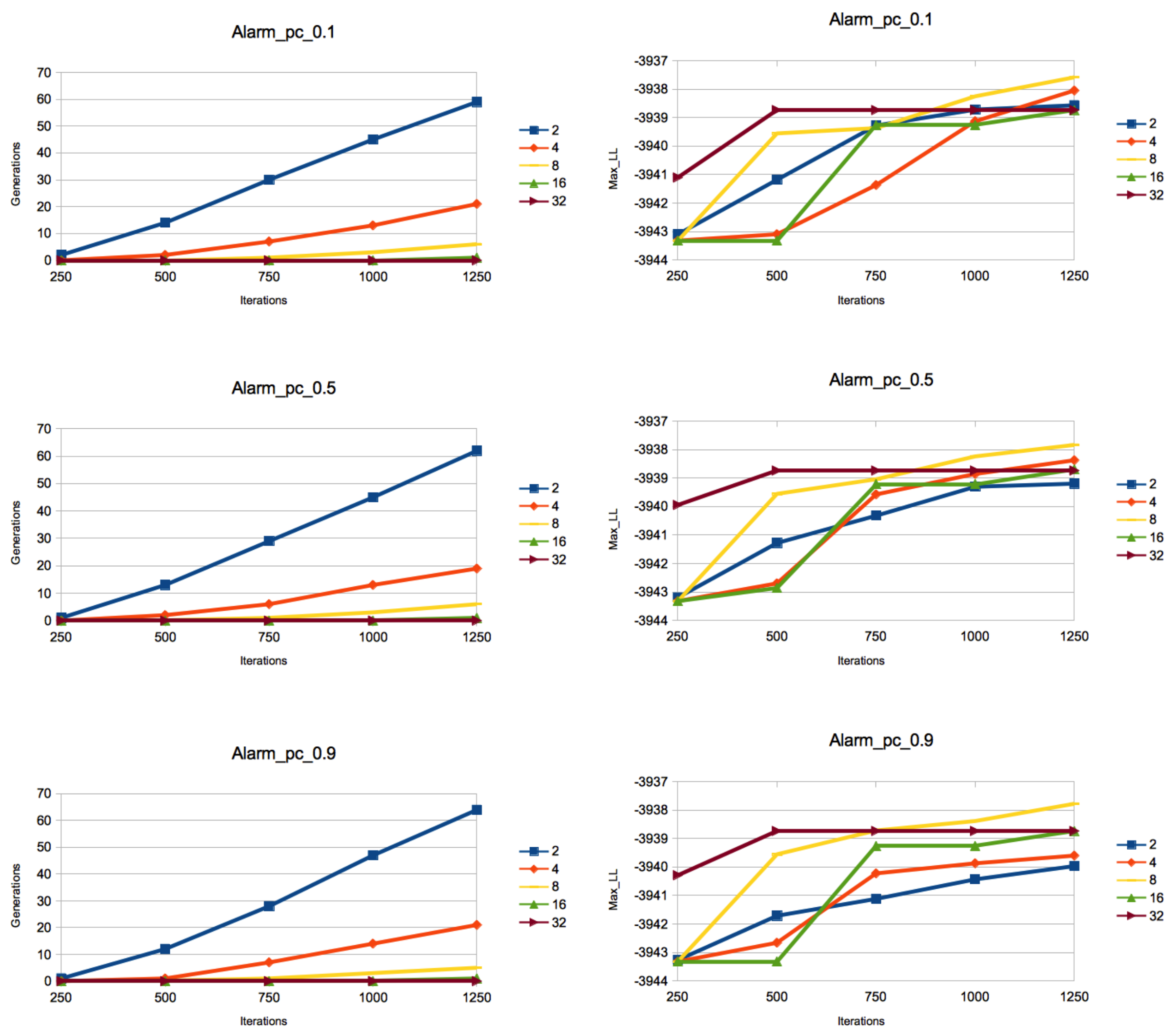


Figure 3.7: Role of crossover on alarm BN (hard search space) when GAEM is run for $n_{GAEM} = \{250, 500, 750, 900, 1250\}$ EM iterations, shown on the x-axis. On the y-axis, in the left column we show the cumulative number of generations, in the right column we show the maximum log likelihood.

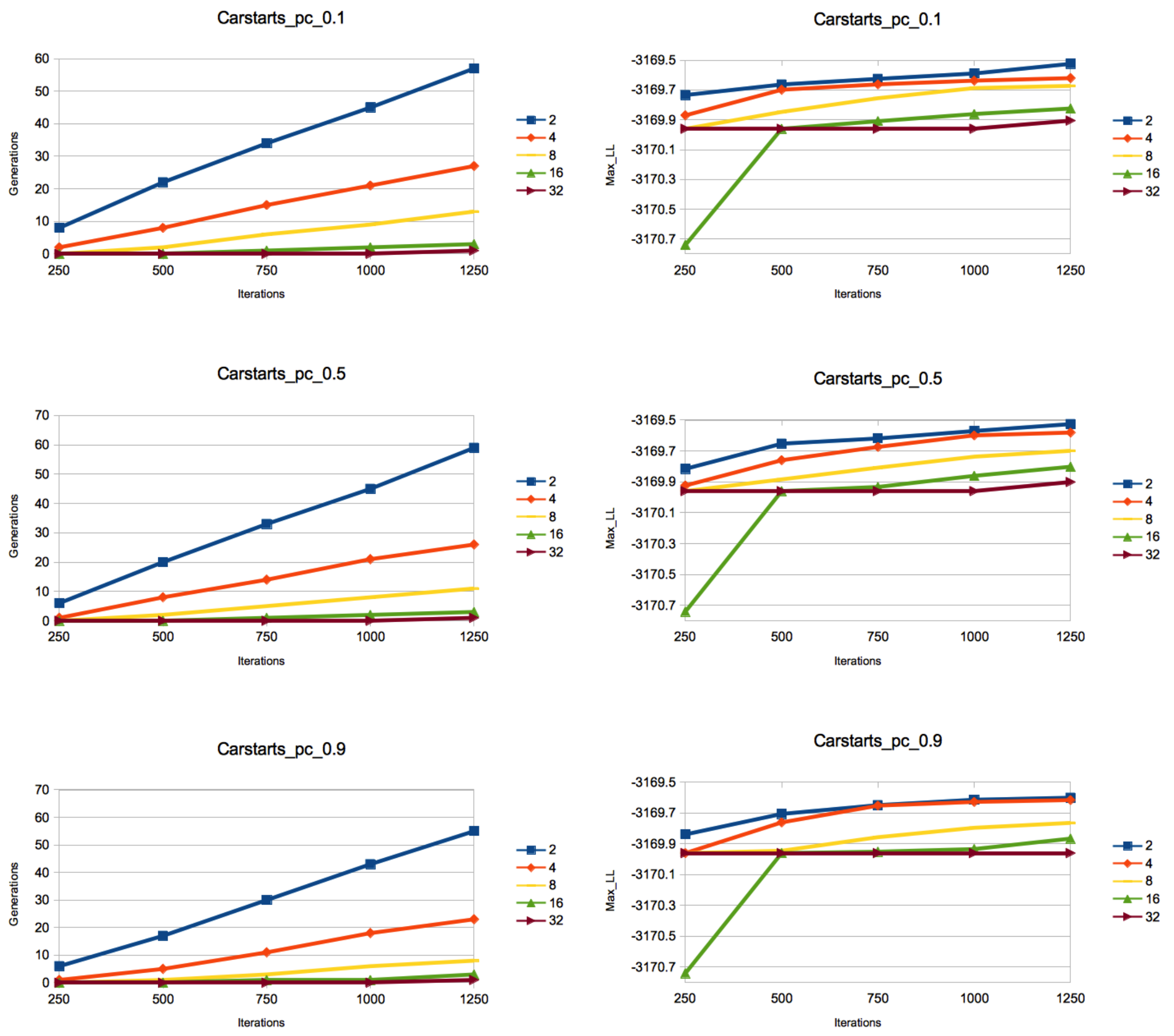


Figure 3.8: Role of crossover on carstarts BN (easy search space) when GAEM is run for $n_{GAEM} = \{250, 500, 750, 900, 1250\}$ EM iterations, shown on the x-axis. On the y-axis, in the left column we show the cumulative number of generations, in the right column we show the maximum log likelihood.

and easy search spaces, the number of generations remains almost the same. This confirms the exploitative nature of crossover. The alleles found so far are interchanged to exploit the current search space as compared to exploring new search spaces. The explorative nature of search was observed in mutation experiments (Figure 3.7 and 3.6) where the number of iterations increased with mutation probability.

Easy Search Spaces

The results are shown on the right side of Figure 3.8. We can see that the solution quality is high for low population such as $n_p = 2$ and it gets poorer as we increase the population size. A low population size takes more generations to reach the constant EM iterations compared to the high population sizes as shown in the left side of Figure 3.8. Thus, we can conclude that GA search dominates which in turn implies more crossover operations in low population sizes. For easy search spaces, a dominant exploitative search with low population sizes is sufficient to produce better solution quality. The solution quality of optimal population sizes is seen to improve slightly when the crossover probability is increased.

Hard Search Spaces

The results are shown on the right side of Figure 3.7. We saw that for hard and easy search spaces, low population sizes were sufficient to produce good solution quality when high mutation probabilities are used. But in crossover, we can see that a population size of $n_p = 4$ or $n_p = 8$ has to be maintained to observe a good solution quality. This is because a sufficient number of individuals is needed so that the there is a high chance of the exchange of good alleles being used in crossover. If the number of individuals are high $n_p = 32$ or $n_p = 16$, the solution quality is poor. With too many individuals, chance of crossing over good individuals might be less. On the other hand, when the population is low such as $n_p = 2$, the solution quality is very poor (as shown in the right side of Fgure 3.7) which implies that with very less number of individuals, the number of good alleles occurring is less and hence it is difficult for GAEM to come up with good combinations of alleles. Thus a population size of $n_p = 8$ shows the highest solution quality for all crossover probabilities.

Conclusion

For hard search spaces (alarm BN), a population size ($n_p = 4$ or $n_p = 8$) performs better for any crossover probability. For easy search spaces (carstarts BN), a low population size ($n_p = 2$) produces high solution quality for any crossover probability.

3.5.6 Role of Replacement

We tried four different replacement mechanisms such as traditional replacement (GAEM-TRAD), deterministic replacement (GAEM-DETER), probabilistic crowding replacement (GAEM-PC) and ALEM based replacement (GAEM-ALEM) on alarm and carstarts BN representing hard and easy search spaces respectively. For alarm BN, we tried population sizes $n_p = 4$ and $n_p = 8$, the crossover and mutation probabilities are taken as $p_m = 0.1$ and $p_c = 0.1$ respectively. For carstarts BN, we tried population sizes of $n_p = 2$ and $n_p = 4$, the crossover and mutation probabilities are taken as $p_m = 0.05$ and $p_c = 0.5$ respectively. The experiments are run for $n_g = 100$ generations for alarm BN and $n_g = 200$ generations for carstarts BN. The results are taken as an average over 10 independent GAEM runs. The results are shown in Figure 3.9 and Figure 3.10 for alarm and carstarts BN respectively where the left side graphs the number of iterations as a function of the number of generations and the right side graphs shows the maximum log likelihood as a function of the number of generations.

Easy Search Spaces

For a small population size of $n_p = 2$, we see that the number of iterations taken by the GAEM-ALEM replacement technique is lesser (as shown in the left side of Figure 3.10) and the corresponding solution quality is higher (as shown in the right side of Figure 3.10) compared to other replacement techniques. For a higher population size of $n_p = 4$, we see that the number of iterations for GAEM-ALEM is lesser than other techniques. The solution quality is higher for probabilistic crowding based replacement technique.

Hard Search Spaces

From the left side of Figure 3.9, we can see that the number of iterations almost remains the same for all GAEM-PC, GAEM-TRAD and GAEM-DETER replacement mechanisms. This implies that the processor time taken is the same for all replacement mechanisms. We can see that GAEM-ALEM shows a much lesser number of iterations. From the right side of Figure 3.9, we can see that the probabilisitic crowding based replacement technique produces a better solution quality compared to others. For a larger population size of $n_p = 8$, the probabilistic crowding technique produces a higher log likelihood starting from the first generation.

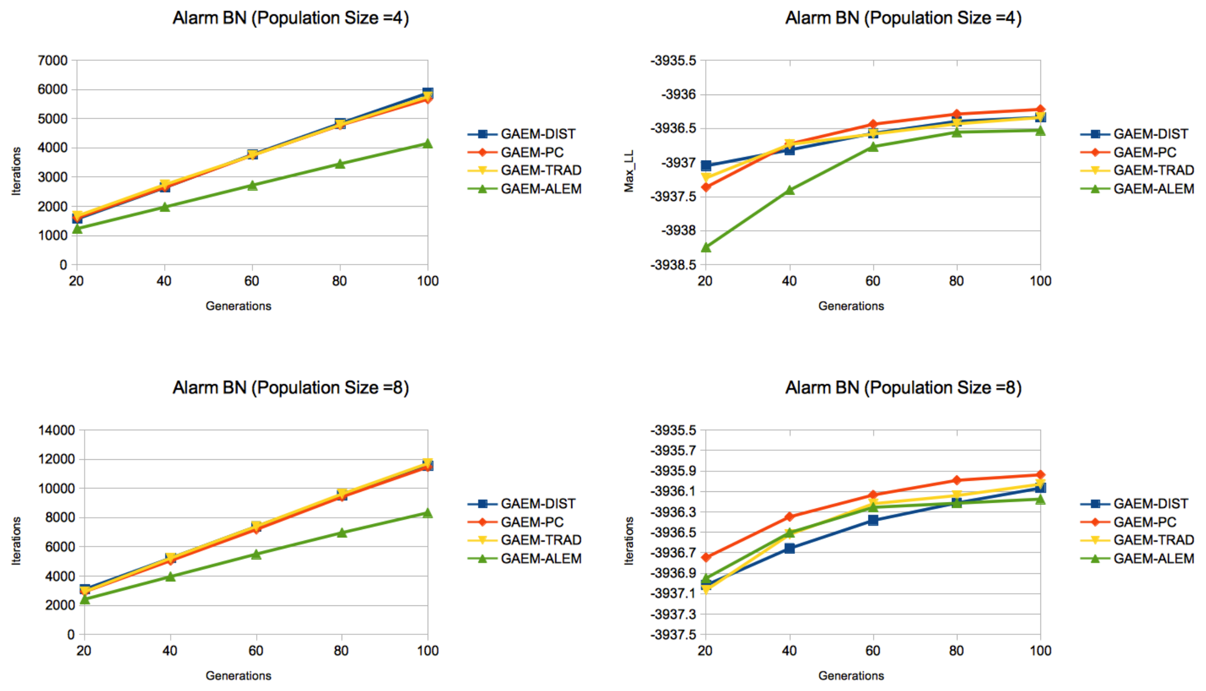


Figure 3.9: Role of replacement techniques on alarm BN (hard search space) when GAEM is run for $n_g = \{20, 40, 60, 80, 100\}$ generations, shown on the x-axis. On the y-axis, in the left column we show the cumulative number of iterations, in the right column we show the maximum log likelihood.

Conclusion

From the above experiments, we see that for all the three replacement mechanisms, the solution quality increases with the increase in population size. We also conclude that for hard search spaces, a larger population size with GAEM-PC based replacement technique produces better solution quality compared to other techniques and GAEM-ALEM based technique produces much lesser number of iterations. For easy search spaces and low population sizes, GAEM-ALEM based replacement technique produces high solution quality in lesser number of iterations compared to other replacement techniques.

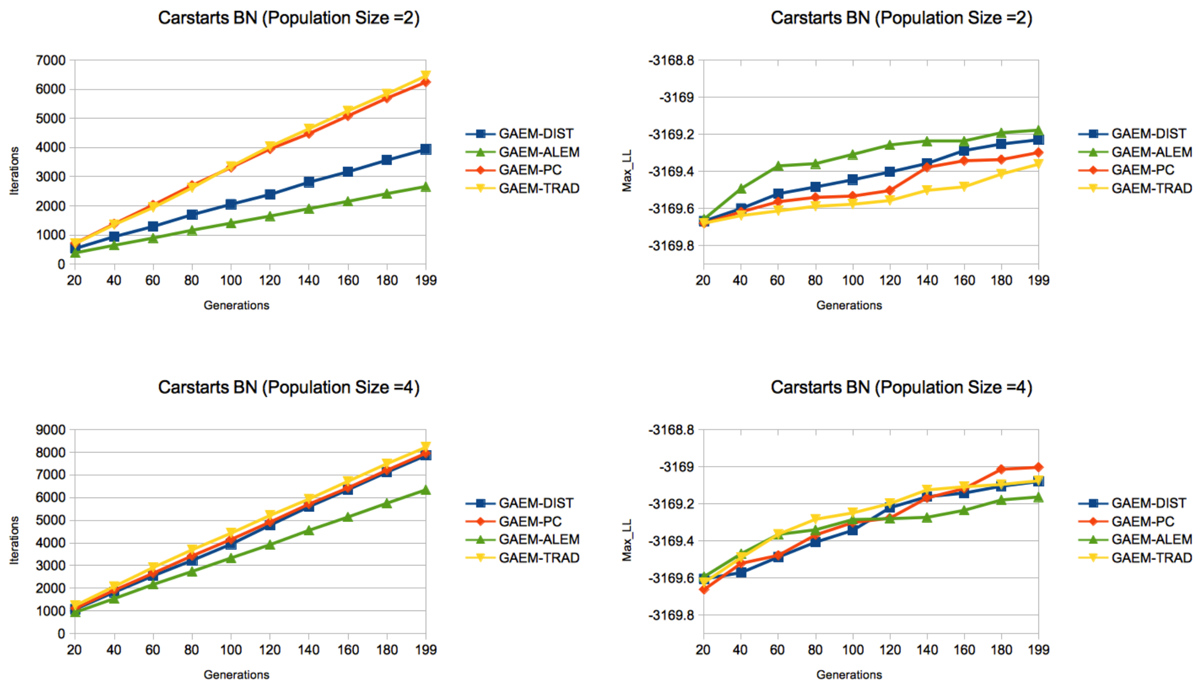


Figure 3.10: Role of replacement techniques on carstarts BN (easy search space) when GAEM is run for $n_g = \{20, 40, 60, 80, 100, 120, 140, 160, 180, 200\}$ generations, shown on the x-axis. On the y-axis, in the left column we show the cumulative number of iterations, in the right column we show the maximum log likelihood.

	Carstarts				Alarm			
Samples	**GAEM-TRAD**	**GAEM-PC**	**GAEM-ALEM**	**EM**	**GAEM-TRAD**	**GAEM-PC**	**GAEM-ALEM**	**EM**
500	-3169.40	-3169.55	**-3169.31**	-3169.96	-3936.93	**-3936.69**	-3937.31	-3937.44
1000	-6924.56	-6924.56	**-6924.54**	-6924.80	**-16047.15**	-16048.18	-16048.6	-16050.20
1500	**-8646.85**	**-8646.85**	**-8646.85**	**-8646.85**	**-22977.85**	-22977.89	-22979.56	-22981.7
2000	-13743.87	-13744.50	**-13743.84**	-13743.85	-31217.09	**-31217.02**	-31217.90	-31218.10
2500	-14504.44	-14504.46	-14504.43	**-14504.40**	**-40550.40**	-40550.97	-40552.73	-40556.00
3000	-18888.15	**-18887.61**	-18887.84	-18887.90	**-51210.45**	-51211.13	-51217.46	-51220.50

Table 3.1: Comparing max_LL for the GAEM-TRAD, GAEM-PC, and GAEM-ALEM methods with the traditional EM method for Carstarts and Alarm. The highest max_LL are in bold. The total number of iterations taken to reach the max_LL is shown in Table 3.2.

3.5.7 Processor Time Comparison

The goal of this experiment is three-fold. First, we want to measure the solution quality (max_LL) obtained by GAEM versus EM. Second, we want to compare the processor time (CPU time) taken by GAEM and EM. Third, we want to understand the speed-up in terms of the total number of iterations taken by GAEM (denoted as n_{GAEM}) versus traditional EM (denoted as n_{EM}). We consider Alarm and Carstarts representing the hard and easy search spaces, respectively. For Alarm, we let $p_c = 0.1$, $p_m = 0.1$, $n_p = 4$, and $n_g = 50$, resulting in a total of $4 \times 50 = 200$ EM runs. For Carstarts, we let $p_c = 0.5$, $p_m = 0.05$, $n_p = 2$, and $n_g = 100$, resulting in a total of $2 \times 100 = 200$ EM runs. For apples-to-apples comparison, we transfer the concept of generation from GAEM to traditional EM. In GAEM, we have n_p EM runs in the first generation and use the learned individuals as parents in subsequent generation, also with n_p EM runs. Similarly, in traditional EM, we start with n_p EM runs in the first generation and use a new set of n_p EM runs in subsequent generations.[5] We tried traditional replacement (GAEM-TRAD), PC based replacement (GAEM-PC) and ALEM based replacement (GAEM-ALEM) and varied the sample sizes from 500 to 3000 for both BNs.[6]

For Alarm, Table 3.1 shows that the solution quality for GAEM-TRAD is higher for most of the sample sizes compared to GAEM-PC and GAEM-ALEM. But all the three replacement methods have a higher solution quality than EM. For Carstarts, all three replacement techniques achieve better solution quality than traditional EM (see Table 3.1) in all cases except for sample size = 2500, where the difference in log-likelihood is very small (0.0002%).

We now consider the number of iterations until convergence, as well as the speed-up, for the

[5]We keep the total number of EM runs as 200 in both BNs because we assume that a new user who wants to compare GAEM with traditional EM would allot time based on the time taken by traditional EM per EM run.

[6]The results for GAEM-DETER is found to be similar to GAEM-PC in terms of the number of iterations and solution quality hence it is not shown in the Tables 3.1 and 3.2.

	Carstarts			Alarm		
Samples	**GAEM-TRAD**	**GAEM-PC**	**GAEM-ALEM**	**GAEM-TRAD**	**GAEM-PC**	**GAEM-ALEM**
500	2049 (4.1)	2757 (3.1)	1397 (**6.0**)	3062 (4.6)	3172 (4.5)	2322 (**6.1**)
1000	1227 (4.6)	1765 (3.2)	1016 (**5.6**)	1659 (3.5)	1631 (3.6)	1386 (**4.2**)
1500	624 (**1.5**)	624 (**1.5**)	624 (**1.5**)	2515 (4.2)	2522 (4.2)	1940 (**5.5**)
2000	1282 (4.4)	1231 (4.6)	989 (**5.8**)	1097 (3.6)	1107 (3.6)	997 (**4.0**)
2500	688 (2.0)	683 (**2.1**)	684 (**2.1**)	2507 (3.8)	2499 (3.8)	1774 (**5.4**)
3000	1214 (5.2)	1208 (5.2)	977 (**6.5**)	4082 (4.0)	4002 (4.1)	2353 (**7.0**)

Table 3.2: Comparing total iterations for GAEM-TRAD, GAEM-PC and GAEM-ALEM methods with traditional EM method. Speedups are shown in parentheses, with the highest speedups in bold. These results corroborate the CPU-time experiments in Figure 4.5.

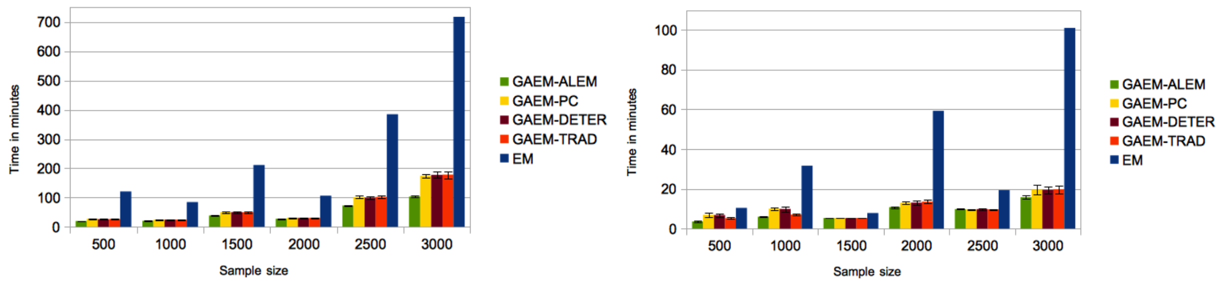


Figure 3.11: Processor time comparison for EM and GAEM on Alarm (left) and Carstarts (right) for 200 EM runs. The corresponding max_LL for these configurations are shown in Table 3.1.

replacement methods. The speed-up is calculated as $s_X = \frac{n_{EM}}{n_X}$ where X reflects the replacement method. In Table 3.2, for Alarm, we can see a speed-up of $3.5 \leq s_X \leq 4.6$ (as shown in the parentheses) for all sample sizes using GAEM-TRAD and GAEM-PC techniques and a much higher speed-up $4.0 \leq s_{ALEM} \leq 7.0$ for GAEM-ALEM. For Carstarts, GAEM-TRAD and GAEM-PC show a speed-up $1.5 \leq s_X \leq 5.2$ for all sample sizes (see Table 3.2). But GAEM-ALEM has a higher speed-up $1.5 \leq s_{ALEM} \leq 6.5$. These results suggest that GAEM can achieve a better solution quality in a shorter time, measured by the number of EM iterations, compared to the traditional EM for given set of random starting point configurations.

Figure 4.5 shows the results of comparing processor time for Carstarts and Alarm. All GAEM-based methods take less time than traditional EM for both BNs across all sample sizes. For both Alarm and Carstarts, we can see that the processor time consumption for GAEM-PC and GAEM-DETER is almost the same. Among the methods, GAEM-ALEM is the fastest. For Carstarts, GAEM-ALEM is the most suitable since there is an improvement both in solution quality and compute time. For Alarm, GAEM-ALEM shows a decrease in the total number of iterations which in turn shortens CPU time. The solution quality, though poorer than for GAEM-TRAD and GAEM-PC, is better than for traditional EM. We conclude that GAEM-ALEM obtains better solution quality (LL) in less time, relative to traditional EM, for both BNs.

3.6 Conclusion

In this work, we develop a Genetic Algorithm for EM (GAEM) for parameter learning from incomplete data in Bayesian networks. From experiments, we find that GAEM achieves better log-likelihoods in most cases compared to the traditional EM algorithm. A key component of GAEM is a novel replacement technique, GAEM-ALEM. Using GAEM-ALEM, poorly performing EM runs, in terms of their log-likelihoods, are discarded early during the progress of EM optimization. In experiments, GAEM-ALEM produced a speed-up of 1.5 to 7, along with better solution quality, compared to the traditional EM algorithm. These results suggest a general capability of GAEM to produce better solutions in shorter time relative to traditional EM. We also prove the global convergence of GAEM theoretically. For future work, we will explore other evolutionary methods and compare their performance to that of GAEM. We will also investigate other ways of characterizing the structure of the BN parameter search spaces.

Chapter 4

ALEM: Age Layered Expectation Maximization

The EM algorithm is a popular algorithm for parameter estimation in models with hidden variables. However, the algorithm has several non-trivial limitations, a significant one being variation in eventual solutions found, due to convergence to local optima. Several techniques have been proposed to allay this problem, for example initializing EM from multiple random starting points and selecting the highest likelihood out of all runs. In this chapter, we a) show that this method can be very expensive computationally for difficult Bayesian networks, and b) in response we propose an age-layered EM approach (ALEM) that efficiently discards less promising runs well before convergence. Our experiments show a significant reduction in the number of iterations, typically two- to four-fold, with minimal or no reduction in solution quality, indicating the potential for ALEM to streamline parameter estimation in Bayesian networks.

4.1 Introduction

The EM algorithm [18] is one of the most established ways to perform parameter estimation with incomplete or hidden data (e.g., [8, 87]). The basic idea of the algorithm is to alternate between an E-step, wherein the algorithm uses current parameter estimates to generate a complete data likelihood, and a M- step, in which the parameters are modified with the goal of maximizing the data likelihood. These steps repeat until convergence. The algorithm and its variants and special cases are prevalent in machine learning, as one can view training algorithms as optimizers, where the overall aim is to estimate a set of parameters while maximizing a 'score', most often the data likelihood, on training data.

Despite its successful and widespread use, the EM algorithm has at least one severe limitation: it has a strong tendency to gravitate towards the locally optimal solution, depending primarily on the initialization of the algorithm. This phenomenon can in turn lead to other issues, for example poor generalization to unseen test data, and is especially exacerbated in parameter estimation for difficult Bayesian networks, where we attempt to estimate parameters for a joint distribution with many conditional dependencies. Another limitation is that the EM algorithm can be very time-consuming, due to its slow convergence speed, in terms of the number of iterations undergone.

In this work, we focus on speeding up the EM algorithm for difficult Bayesian networks with no or minimal degradation in solution quality.

- We present and analyze our approach, age layered EM (ALEM) (Section 4.3), where we incorporate an age-layered population structure heuristic in an attempt to reduce the average number of iterations in a multiple starting point EM setup.
- We highlight the local optima and slow convergence problems for a multiple starting points EM strategy for select difficult Bayesian networks (Section 4.4.2). Our experiments with these networks show that the main problem is the slow convergence of the EM algorithm rather than the log-likelihood shortfall of the local optima.
- We demonstrate ALEM's ability to significantly reduce the average number of iterations (Section 4.4.3), typically two- to four-fold, and for larger sample sizes the wall-clock time as well by the same magnitude.

4.2 Related Work

Several approaches have attempted to mitigate the local optima problem in the EM algorithm. One can use an upper bound on the eventual log likelihood of a run at termination as a criterion for early dismissal of a run [107]. Alternatively, the training data can be perturbed, rather than the hypothesis, to generate new directions for greedy hill-climbing ascent [24]. The training data is then annealed back to its original state over time. A genetic algorithm approach has also been used [46]. Some work has looked at alternative stopping criteria that are less likely to converge to local optima [64].

A prevalent way to tackle local optima is the multiple starting point or multiple restart strategy [43, 68], wherein we initialize the EM algorithm from N random starting points. After the N

EM runs have completed, we choose the run that resulted in the highest data log likelihood. This strategy allows us to search the parameter space more extensively for the optimal values, but can be extremely expensive, considering that we expend a significant amount of processing cycles on runs that are in no way guaranteed to be the global optimum or even close to it. We show this experimentally in Section 4.4.

The effects of parameter sharing on local maxima during Bayesian network parameter learning when there is incomplete data has been explored [89]. The severity of the local optima problem in EM has been demonstrated [105], but the authors restricted their analysis to a specific type of Bayesian network, the hierarchical latent class model. In Section 4.4, we perform a similar analysis on two networks that do not fall into the hierarchical latent class model, and show that the issue is prevalent in a wider class of networks but that the main problem is the variation in average number of iterations needed, and not so much log likelihoods of the local optima.

Additional work has looked at scaling up EM to very large datasets, for example incremental and lazy variants of EM [102]. Incremental EM partitions a dataset into blocks, with EM computed incrementally on these blocks, the main variable being optimal block size selection [78]. Lazy EM picks a significant subset of the dataset after a given number of iterations, and then proceeds with EM using only this subset. The emphasis of these techniques is how to scale EM to large databases, whereas our work focuses not necessarily on scale but on the orthogonal issue of how to speed up parameter estimation on *difficult* Bayesian networks. As such, the scalability methods and our approach can easily be combined. Simulated annealing [59] and Tabu search methods [76] have been incorporated into EM, but they serve primarily as an alternative to the multiple starting point strategy with the goal of avoiding local minima, and often require a re-implementation of the core algorithm.

Our approach is influenced by the age-layered population structure (ALPS) paradigm, originally implemented for genetic algorithms [40]. ALPS assigns an *age* to each individual in the population, which is then used to sort the individuals into different age layers, each layer having its own age limit. Competition amongst individuals in the population is restricted to within age layers, and all layers evolve independently. This division of the population aims to create a more 'fair' competition between individuals, as older individuals in the population compete amongst themselves and not with substantially younger individuals. When an individual's age exceeds upper limit of its current layer, it is moved up to the next layer if it can replace a poor performing individual in the target layer, otherwise it is discarded.

4.3 Expectation Maximization Approach

We present and analyze our algorithm which, like ALPS, relies on an age-layered structure to efficiently remove underachieving runs early on, along with pseudocode for our algorithm.

4.3.1 The ALEM Algorithm: Age-Layered EM

The EM algorithm with multiple random starting points (henceforth referred to as 'traditional') can be viewed as a genetic algorithm: each starting point can be seen as an individual. For the age-layered paradigm, a natural measure of age would be the number of iterations an EM run has undergone. With these basic concepts, we developed ALEM (Figure 4.1), the Age-Layered Expectation Maximization algorithm.

In the *main* procedure of Figure 4.1, we have a set of L layers, $\Gamma = \{\Gamma_1, \Gamma_2, \Gamma_3, \ldots, \Gamma_L\}$, and a set of N EM runs, $\rho = \{\rho_1, \rho_2, \rho_3, \ldots, \rho_N\}$ where $\Gamma_i \subseteq \rho$. We denote the age limit or the maximum number of iterations for the i^{th} layer to be β_i, the minimum number of runs in the i^{th} layer to be M_i (these parameters control how runs move between layers and will be elaborated upon shortly), and the set of runs[1] in the i^{th} layer to be Γ_i. Note that $\Gamma_1 \cup \cdots \cup \Gamma_L = \rho$, and $\Gamma_i \cap \Gamma_j = \emptyset$, for $1 \leq i, j \leq L,\ i \neq j$. The number of iterations reached by the j^{th} EM run is given by $\eta(\rho_j)$ and its log likelihood is given by $LL(\rho_j)$.

Each EM run ρ_j is created by initializing its variables with randomly generated initial probabilities. We predefine the age limit β_i for each layer Γ_i, except for the last layer, through an exponential relationship between age limit and layer number: $\beta_i = a \cdot 2^{i-1}$, where β_i is the age limit for the i^{th} layer, and a is a predetermined constant (the age gap). Other relationships can also be used, but we chose this representation as it corresponded well empirically with the distribution of iterations in a multiple starting points strategy (Section 4.4.2). The last layer has no age limit, it is set to the maximum number of iterations ω that an EM run can undergo as specified for the EM algorithm: $\beta_L = \omega$.

[1] Γ_i refers to both the layer as well as the set of runs in that layer.

In ALEM, there are three ways to terminate a run. First, no run can exceed ω, the maximum number of iterations for a run. Second, there is a log likelihood difference tolerance ε (the relative difference in log likelihood between two successive iterations). These two termination criteria, as shown in ALEMCHECK (Figure 4.1), are standard convergence criteria used for the traditional EM algorithm [12, 106] and are not specific to ALEM.[2] A run that terminates through these two termination criteria is called an *inactive* run and resides in the layer it was last in; all other non-terminated runs are called *active* runs. The third way, which we call *culling*, is ALEM-specific and amounts to a run failing the log likelihood comparison test. As shown in CHECKRUNS (Figure 4.1), if the age of an EM run ρ_j in layer Γ_i reaches β_i, then we remove it from layer Γ_i and try to insert it into the next layer Γ_{i+1}. An attempted insertion works as follows: each layer has an associated *minimum runs* parameter, M_i. We define the minimum run M_i for a layer Γ_i as the minimum number of runs that need to be in the layer before ALEM does a log likelihood comparison check. If the number of active runs in Γ_{i+1} is less than M_{i+1}, then we automatically insert the EM run from Γ_i into Γ_{i+1}. Otherwise, we perform a log likelihood comparison between the EM run to be shifted up, ρ_j, and runs, active or inactive, in Γ_{i+1}: if a run with worse log likelihood is found in Γ_{i+1}, then that run is *discarded*, i.e., it is completely removed from the age layers, and ρ_j takes its place. If both the minimum runs and the log likelihood comparison conditions fail, then ρ_j is discarded and we proceed. Discarding only takes place when a run fails the log likelihood comparison test.

We can intuitively view M_i as a parameter that can trade off computational efficiency and solution quality. The lower the value of this parameter for a given layer Γ_i, the more likely ALEM is to conduct a log likelihood check. On the other hand, inserting the run without a check will mean we do not get a chance to cull. For simplicity in the rest of this paper we assume all internal layers have the same minimum runs value ($M_2 = M_3 \cdots = M_{L-1}$), except for the first layer Γ_1 and the last layer Γ_L. In Γ_1, new runs are inserted when the number of active runs in that layer is below M_1, and thus M_1 controls the rate at which we introduce these new runs. The last layer can accommodate all N EM runs, and thus $M_L = N$.

The main assumption is that log likelihood comparisons between similarly aged runs are a reliable indicator of comparisons at convergence. Justification for this assumption is provided

[2]Despite the prevalence of these criteria, it has been noted [61] that the difference in successive log likelihoods is a measure of lack of progress rather than convergence.

in Section 4.4.2, where we discuss empirical results on the distribution of log likelihoods and iterations. We note a significant difference in iterations between runs that achieve the highest log likelihood and runs that do not, which led us to posit that many runs are unsuccessful because they are initialized in bad areas of the search space.

4.3.2 Analysis using Poisson Processes

In traditional EM, runs mostly terminate because the algorithm fails to improve the log likelihood by a pre-specified amount ϵ between successive iterations. In ALEM, a run terminates for the same reason, but in addition it can also be culled. In this section, we formalize this intuition by analyzing ALEM's rate of convergence as compared to that of traditional EM.

In a shared resource environment (e.g., all processes running on the same machine), the total time required to complete a multiple starting points experiment can be thought of as the product of the number of runs and the average time taken per run (or upper-bounded by the maximum time over all runs). If we take into account the fact that as certain runs terminate, computational resources free up and can be assigned to the runs that are still iterating, it then becomes a question of which strategy results in a higher probability of undesirable (runs that iterate for too long, runs that will converge to local optima, etc.) runs terminating early on. The main condition for ALEM's successful performance is that the probability of a young, poorly performing run turning into an older, strongly performing run is low, and can be made low by the ALEM parameters, i.e., β_i and M_i. We model the termination of runs stochastically by assuming that terminations in traditional EM and ALEM follow a Poisson process, motivated by the discrete, independent, and uniformly distributed nature of the events (terminations) occurring over a continuous time interval. Specifically, let $N(t)$ be a Poisson process that dictates the number of terminations in a given time interval, such that

$$P[(N(t+\tau) - N(t)) = k] = \frac{e^{\lambda\tau}(\lambda\tau)^k}{k!} \quad \text{for } k = 0, 1, \ldots \tag{4.1}$$

where k is the number of terminations. We can model ALEM terminations as the superposition of two independent Poisson processes: $N_{\text{EM}}(t)$ with Poisson parameter λ_{EM}, which is the process for terminations occurring through standard EM termination, and $N_{\text{cul}}(t)$ with Poisson parameter λ_{cul} which is the culling process. The resulting Poisson process $N_{\text{ALEM}}(t)$ has parameter $\lambda_{\text{ALEM}} =$

Algorithm 4.3.1: ALEM (N, L, M)

```
procedure CheckRuns($\Gamma_i, \rho_l$)
  if $|\Gamma_{i+1}| < M_{i+1}$
    then { $\Gamma_{i+1} \leftarrow \Gamma_{i+1} \cup \{\rho_l\}$
           $inserted \leftarrow true$
    else { comment: find if there is a worse run
           for each $\rho_m \leftarrow \Gamma_{i+1}$
             do { if $LL(\rho_m) < LL(\rho_l)$
                    then { $\Gamma_{i+1} \leftarrow \Gamma_{i+1} - \{\rho_m\}$
                           Discard EM run $\rho_m$
                           $\Gamma_{i+1} \leftarrow \Gamma_{i+1} \cup \{\rho_l\}$
                           $inserted \leftarrow true$
  if $inserted = false$
    then {Discard EM run $\rho_l$
  $\Gamma_i \leftarrow \Gamma_i - \{\rho_l\}$

procedure ALEMCheck $(N, L, M)$
  comment: x denotes the number of terminated runs
  $x \leftarrow 0$
  while $x \leq N$
    do { for $i \leftarrow 1$ to $L$
           do { if $i = 1$ and $|\Gamma_i| < M_i$
                  then { $k \leftarrow M_1 - |\Gamma_1|$
                         Insert k EM runs
                         $\Gamma_1 \leftarrow \Gamma_1 \cup \{\rho_j, \rho_{j+1} \cdots \rho_k\}$
                         Start traditional EM algorithm
                         for each $\rho_j \leftarrow \Gamma_1$
                           do EM$(\rho_j, \varepsilon, \omega)$
                for $\rho_l \leftarrow \Gamma_i$
                  do { If EM run terminated based on $\omega$ or $\varepsilon$
                       if $\rho_l.isTerminated$
                         then $x \leftarrow x + 1$
                       else if $\eta(\rho_l) = \beta_i$
                         then CheckRuns($\Gamma_i, \rho_l$)

main
  { Initialize layers with age limit $\beta$
    for $i \leftarrow 1$ to $L - 1$
      do $\beta_i \leftarrow (a) \cdot 2^{i-1}$
    Set $\beta_L$ for the top layer
    $\beta_L \leftarrow \omega$
    Initialize layers with minimum runs $M$
    for $i \leftarrow 2$ to $L - 1$
      do $M_i \leftarrow 2$
    Set $M$ for the bottom-most layer
    $M_1 \leftarrow q$
    Set $M$ for the top layer
    $M_L \leftarrow N$
    Insert EM runs in the bottom-most layer
    $\Gamma_1 \leftarrow \{\rho_1, \rho_2 \cdots \rho_q\}$
    Start traditional EM algorithm on each run
    for each $\rho_q \leftarrow \Gamma_1$
      do EM$(\rho_q, \varepsilon, \omega)$
    ALEMCheck$(\Gamma, \rho)$
    Find EM run with max log likelihood
    for $i \leftarrow 1$ to $L$
      do $LL \leftarrow \arg\max(\Gamma_i)$
```

Figure 4.1: Pseudocode for ALEM algorithm. The values for β_i, M_i and a can be set depending on the nature of the Bayesian network. In our experiments, we have set $a = 5$, $\omega = 1000$, $N = 200$, $M_1 = 5$ for the bottom layer, $M_L = N$ for the top layer, $M_i = 2$ for $i < 2$ to $L - 1$ and $\varepsilon = 0.00001$

$\lambda_{\text{EM}} + \lambda_{\text{cul}}$ [50].

Lastly, if we let T^k_{ALEM} be the time taken for ALEM to reach k terminations, we can can say $T^k_{\text{ALEM}} = X^1_{\text{ALEM}} + X^2_{\text{ALEM}} + \cdots + X^k_{\text{ALEM}}$, where each element in the sequence X^i_{ALEM} is an i.i.d. exponential random variable with density function $f_{\text{ALEM}}(t) = \lambda_{\text{ALEM}} e^{-\lambda_{\text{ALEM}} t},\ t \geq 0$, and is the distribution of run i terminating before time t. We can similarly define T^k_{EM} for the traditional EM approach and also define it as a sum of i.i.d. exponential random variables with $\lambda = \lambda_{\text{EM}}$. Now let $Y_k = T^k_{\text{ALEM}} - T^k_{\text{EM}}$. Then,

$$P[T^k_{\text{ALEM}} < T^k_{\text{EM}}] = P[Y_1 + Y_2 + \dots Y_k < 0] \tag{4.2}$$

noting that $Y_i, i = 1, \dots, k$ are i.i.d. random variables with mean $\mu = \frac{1}{\lambda_{\text{ALEM}}} - \frac{1}{\lambda_{\text{EM}}}$ and variance $\sigma^2 = \frac{1}{\lambda^2_{\text{ALEM}}} + \frac{1}{\lambda^2_{\text{EM}}}$ (addition of independent exponential random variables). Thus,

$$\begin{aligned} \frac{\mu}{\sigma} &= \frac{\lambda_{\text{EM}} - \lambda_{\text{ALEM}}}{\lambda_{\text{ALEM}}\lambda_{\text{EM}}} \frac{\lambda_{\text{ALEM}}\lambda_{\text{EM}}}{\sqrt{\lambda^2_{\text{ALEM}} + \lambda^2_{\text{EM}}}} \\ &= \frac{\lambda_{\text{EM}} - \lambda_{\text{ALEM}}}{\sqrt{\lambda^2_{\text{ALEM}} + \lambda^2_{\text{EM}}}} \end{aligned} \tag{4.3}$$

Now let $T^k_Y = Y_1 + Y_2 \dots Y_k$, and $Z_k = \frac{T^k_Y - k\mu}{\sigma\sqrt{k}}$. Then,

$$\begin{aligned} P[Y_1 + \dots Y_k < 0] &= P\left[\frac{T^k_Y - k\mu}{\sigma\sqrt{k}} < \frac{-k\mu}{\sigma\sqrt{k}}\right] \\ &= P\left[Z_k < \frac{\lambda_{\text{ALEM}} - \lambda_{\text{EM}}}{\sqrt{\lambda^2_{\text{ALEM}} + \lambda^2_{\text{EM}}}}\sqrt{k}\right] \end{aligned} \tag{4.4}$$

From the central limit theorem, Z_k converges in distribution to the normal distribution with $\mu = 0, \sigma^2 = 1$. Therefore, if $\lambda_{\text{ALEM}} > \lambda_{\text{EM}}$, i.e., $\lambda_{\text{cul}} > 0$, then Equation 4.4 tends to 1 as $k \to \infty$. The analysis gives the probability of the time taken for ALEM to reach k terminations being less than the time taken for traditional EM to reach k runs. This probability approaches 1 as the number of terminations increases.

4.4 Experiments with Bayesian Networks

The objectives of the experiments are two-fold: to compare the average number of iterations undergone by all runs in traditional EM and in ALEM, and to analyze the solution quality of ALEM vis-à-vis traditional EM. In all sets of experiments, we set ω=1000, and $\varepsilon = 0.00001$, i.e., if the relative difference in log likelihood between two successive iterations is less than ε, then we deem the EM run to have converged or terminated and set its status to inactive. We also analyze the effects of varying the minimum runs parameter on the average iterations and the solution quality, and provide wall-clock times comparing the two strategies.

4.4.1 Datasets and Evaluation

We performed experiments on four Bayesian networks of different complexities, the Carstarts with 18 nodes, Alarm with 37 nodes, Hepar2 with 70 nodes and Win95pts with 76 nodes. The Carstarts BN [36] is based on the various operations in a car. The more complex Alarm BN [9] represents a real life situation for monitoring a patient in intensive care unit. Hepar2 BN [83] is used for diagnosing liver disorders. The Win95pts BN [41] is created for printer troubleshooting in Windows 95. We collected these BNs from the bnlearn repository [3].

For all these networks, we generated $N = 200$ EM runs by randomly choosing starting points in the search space, in this case starting conditional probability distribution values, for each of the random variables in each run. We then varied two experimental parameters, namely the training set size and the number of hidden variables. Firstly, through a Gibbs sampler, we generated training sets consisting of 100, 250, 500, 1000, 2000, and 4000 samples. For Carstarts and Alarm BNs, the EM runs are run in parallel on the same Linux machine, a 2.5 GHz Intel quad core processor with 8GB RAM. For Hepar2 and Win95pts BNs, the EM runs are run in parallel on the same Linux machine, a 2.2 GHz Intel 22 core processor with 16GB RAM. We used the libDAI graphical models library [72], which contains an EM implementation suitable for Bayesian network parameter estimation.

For each sample size, we generated n different hidden variable configurations, where n is the number of random variables in the network, and for each configuration, we randomly chose m variables to be hidden. For Alarm and Carstarts BN, m ranges from 0 to $n - 1$. For example, for the Carstarts network, we had 18 different hidden variable configurations, ranging from 0 hidden variables to 17 hidden variables. For Win95pts and Hepar2, we hid 25%, 50% and 75% of random variables.

For our first set of experiments (Section 4.4.2), we focus on Carstarts and Alarm BN. We

[3]http://www.bnlearn.com/bnrepository/

calculated the *average relative likelihood shortfall* for each hidden variable configuration-sample size pair as:

$$\mathrm{RLS}_{\mathrm{avg}} = \frac{l_{\mathrm{avg}} - l^*}{l^*} \tag{4.5}$$

where l^* is the maximum log likelihood from the 200 EM runs and is deemed the global maximum,[4] and l_{avg} is the average log likelihood across all 200 EM runs. We also computed the average number of iterations undergone by all runs. This exhaustive analysis allowed us to identify which hidden variable configurations and sample sizes took a large number of iterations to converge (we term these configurations as 'problematic'), as well as the distribution of log likelihood values at convergence.

For the second set of experiments (Section 4.4.3), we focused on the hidden variable configurations from the first set of experiments that we found problematic (for Alarm and Carstarts BN) and specific set of hidden variables (*i.e.,* we hid 25%, 50% and 75% of randomly chosen variables for Hepar2 and Win95pts BN). We perform EM parameter estimation through ALEM on these hidden variable configurations. We compare average iterations and solution quality (through Equation 4.5) between the traditional and ALEM approaches. The third set of experiments (Section 4.4.4) focuses on varying the minimum runs parameter; we restrict ourselves to the hidden variables used for the second experiment. The fourth group of experiments (Section 4.4.5) compares the wall-clock time between the traditional EM and ALEM, once again restricted to the hidden variables used for the second experiment.

4.4.2 Slow Convergence in Traditional EM

In the first set of experiments, we focus on Carstarts and Alarm BN. We used traditional EM and went through the 6 sample sizes and the n different hidden variable configurations ($n = 37$ for Alarm, $n = 18$ for Carstarts) for each of the 200 EM runs. Therefore, the total number of EM runs amounted to $6 \times 37 \times 200 = 44,400$ for Alarm, and $6 \times 18 \times 200 = 21,600$ for Carstarts. For each hidden variable configuration-sample size pair, we calculated the average RLS as per Equation 4.5.

Figures 4.2 and 4.3 show the average number of iterations for the networks as we vary hidden variables and sample size. We found that $\mathrm{RLS}_{\mathrm{avg}}$ never exceeds 1%, which means that on average

[4]We use for simplicity the term 'global' maximum for the purposes of this paper, although we recognize that the maximum achieved from multiple starting points is not necessarily the true global maximum.

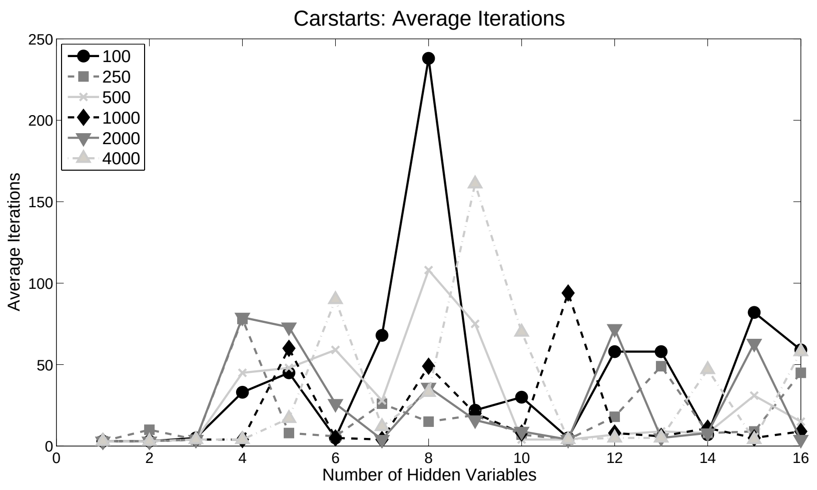


Figure 4.2: Carstarts network: average number of iterations for 200 EM runs, across all hidden variable and sample size configurations. Notice the peaks at particular hidden variable configurations.

the log likelihood values obtained from each of the 200 EM runs was never more than 1% away from the global maximum. In addition, the average RLS peaks and average iteration peaks in the figures tracked each other fairly well. We generally find that the highest peaks occur with the 100 sample size experiments, although one cannot establish a trend that a lower sample size leads to a higher average number of iterations.

We also looked at the runs that eventually reached the global maximum (successful runs), and calculated the average iterations for just those runs versus the average over all runs, and found significant differences between these two sets of runs. The average number of iterations for all runs exceeds the average number of iterations for successful runs by 17.6 iterations in Carstarts and 5.2 iterations in Alarm, on average. This thorough analysis allowed us to identify problematic hidden variable configurations and sample sizes which were more time-consuming. Therefore, we felt that focusing on reducing the average number of iterations would result in the most significant improvement in terms of the overall time taken to complete the runs and find the global maximum.

4.4.3 ALEM: Mitigating Slow Convergence

Our ALEM implementation consisted of seven layers ($L = 7$). Based on results from traditional EM (Section 4.4.2), we chose the age gap a to be 5, and so $\beta_1 = 5,\ \beta_2 = 10,\ \beta_3 = 20,\ \beta_4 = 40,\ \beta_5 = 80,\ \beta_6 = 160$, and $\beta_7 = 1000$, as the last layer has no age limit per se and is simply

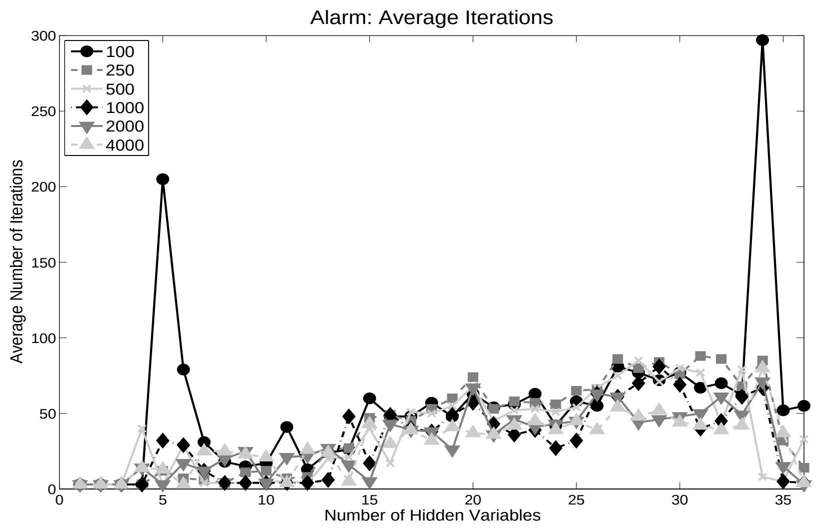


Figure 4.3: Alarm network: average number of iterations for 200 EM runs, across all hidden variable and sample size configurations. Notice the peaks at particular hidden variable configurations.

restricted by ω. For layers 2 to 6 we set $M_i = 2$, with $M_1 = 5$ and $M_7 = 200$.

For the second set of experiments, we run ALEM on the hidden variable configurations from Section4.4.2 that we found problematic, which were 3, 4, 7, and 16 hidden variables for the Carstarts network, and 7, 13, 19, 28, and 33 hidden variables for the Alarm network. These configurations broadly correspond to the peaks in Figures 4.2 and 4.3. For Hepar2 and Win95pts BN, we hid 25%, 50% and 75% of randomly chosen variables. Using ALEM, we did not*exactly* achieve the global maximum in 4 out of 24 hidden variable-sample size configurations (6 sample sizes $\times$ 4 hidden variable configurations) in Carstarts, although on average we were within 0.007% of the global maximum for the ones we missed. For Alarm, we were off the global maximum for 12 out of the 30 experiments (6 sample sizes $\times$ 5 hidden variable configurations), but the average shortfall for the configurations where we did not achieve the global maximum was also 0.007%. However, both of these RLS_{avg} values are orders of magnitude less than the RLS_{avg} obtained in the traditional experiments, which were 0.13% and 0.10% for Carstarts and Alarm respectively, indicating ALEM's effectiveness even on configurations where the global maximum was not found.

In no case did the average number of iterations ever increase with ALEM: mostly there is a significant decrease, especially on very hard (i.e., high iterations in traditional EM) instances. In some cases when the average number of iterations is already low there is no change. Table

	Number of hidden variables								
	Carstarts Bayesian network				**Alarm Bayesian network**				
Sample size	**3**	**4**	**7**	**16**	**7**	**13**	**19**	**28**	**33**
100	15 (2.2)	19 (2.4)	93 (2.6)	**59 (3.2)**	14 (1.3)	13 (2.0)	27 (2.3)	27 (2.7)	**6 (49.5)**
250	31 (2.5)	7 (1.1)	9 (1.7)	**30 (3.1)**	4 (1.0)	16 (1.7)	28 (2.6)	35 (2.4)	**6 (14.2)**
500	26 (1.7)	20 (2.4)	38 (2.8)	**18 (3.1)**	4 (1.0)	4 (1.0)	29 (2.3)	**26 (2.7)**	7 (1.1)
1000	4 (1.0)	24 (2.5)	17 (2.9)	**31 (3.2)**	4 (1.0)	19 (2.5)	22 (2.6)	31 (2.7)	**19 (3.5)**
2000	26 (3.0)	**22 (3.3)**	13 (2.8)	8 (1.1)	10 (2.0)	14 (1.1)	**23 (2.9)**	19 (2.4)	25 (2.8)
4000	4 (1.0)	9 (1.9)	13 (2.5)	**27 (2.7)**	14 (1.8)	5 (1.0)	17 (2.2)	19 (2.7)	**30 (2.7)**

Table 4.1: Average number of iterations for chosen hidden variable configurations for the ALEM approach, with speedup $= \frac{\text{traditional iterations}}{\text{ALEM iterations}}$ in parentheses, for Carstarts & Alarm networks. Highest speedups are in bold. These results corroborate with the wall-clock time experiments shown in Figure 4.5.

	Number of hidden variables					
	Win95pts Bayesian network			**Hepar2 Bayesian network**		
Sample size	**18**	**35**	**53**	**19**	**38**	**57**
100	19 (2.8)	25 **(3.7)**	20 (1.8)	21 **(3.9)**	16 (3.3)	6 (1.0)
250	19 (2.6)	27 **(3.9)**	14 (3.1)	21 (3.1)	25 **(3.4)**	10 (1.8)
500	18 (2.9)	25 **(4.0)**	18 (2.8)	19 (3.6)	23 **(3.8)**	11 (1.3)
1000	16 (3.1)	20 **(4.8)**	15 (2.7)	16 **(3.3)**	16 (2.7)	8 (1.3)
2000	14 (2.9)	20 **(4.6)**	18 (3.1)	14 **(3.0)**	14 (2.6)	9 (1.7)
4000	13 (2.4)	18 **(4.1)**	21 (2.8)	15 **(2.9)**	13 (2.5)	7 (1.4)

Table 4.2: Average number of iterations for chosen hidden variable configurations for the ALEM approach, with speedup $= \frac{\text{traditional iterations}}{\text{ALEM iterations}}$ in parentheses,for Win95pts & Hepar2 networks. Highest speedups are in bold.

4.1 (for Carstarts and Alarm BN) and 4.2 (for Hepar2 and Win95pts BN) contains the average number of iterations for the chosen hidden variable configurations in the ALEM approach, with the speedup $\frac{\text{traditional \#iterations}}{\text{ALEM \# iterations}}$ in parentheses. On average the decrease in average iterations for Carstarts is 51.4%, for Alarm it is 47.8%, for Hepar2 it is 65.4% and for Win95pts it is 69.6%. The fact that we managed to reduce the average number of iterations for these hidden variable configurations so significantly yet still achieve the global maximum in most instances (or very close to it in all instances) demonstrates to us how expensive the traditional strategy is. There are clearly some initial points that result in a very high number of iterations but tend not to find the global maximum due to the starting position in the search space. The ALEM approach effectively culls these iterations, thus saving processing cycles.

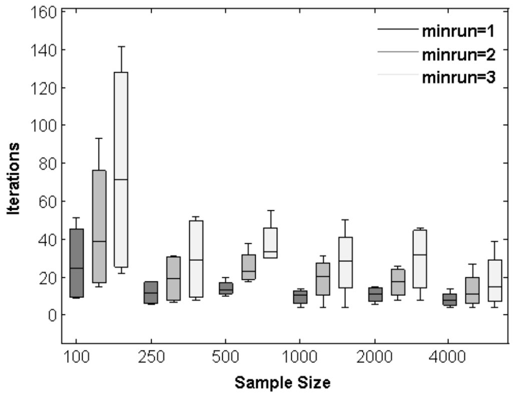
Bayesian network	**Minimum runs(M_i)**	**Failures**	**$\mathrm{RLS_{avg}}$**
Carstarts	1	5	0.009%
Carstarts	2	4	0.007%
Carstarts	3	4	0.001%
Alarm	1	14	0.013%
Alarm	2	12	0.007%
Alarm	3	6	0.006%
Hepar2	1	12	0.11 %
Hepar2	2	8	0.06 %
Hepar2	3	4	0.04 %
Win95pts	1	8	0.06 %
Win95pts	2	5	0.07 %
Win95pts	3	4	0.08 %

Table 4.3: Minimum runs variation and solution quality: 'Failures' is a count of the number of hidden variable-sample size configurations (out of a total of 24 for each entry for Carstarts, 30 for each entry for Alarm, and 18 for each entry of Win95pts and Hepar2) where we fail to achieve the global maximum as we vary the minimum runs, and $\mathrm{RLS_{avg}}$ of *only those experiments* that did not achieve the global maximum.

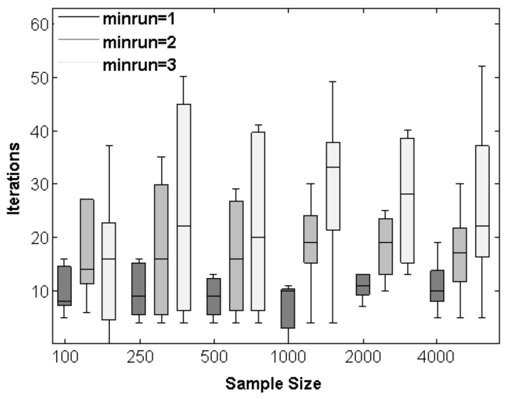
4.4.4 Parameter Variation

We performed a set of experiments where we varied the minimum runs parameter M_i to see its effects on the average number of iterations as well as the solution quality. Figure 5.1 is a box plot that shows how the average number of iterations (across the hidden variables that we tested ALEM for) varies as a function of the sample size and the minimum runs parameter. One can see a clear trend: as we reduce the minimum runs, the average iterations also decreases. In fact, some of the average iteration reductions when setting the parameter to 1 were extremely significant (e.g., 10x reduction in iterations with minimal or no reduction in solution quality, see Table 4.3).

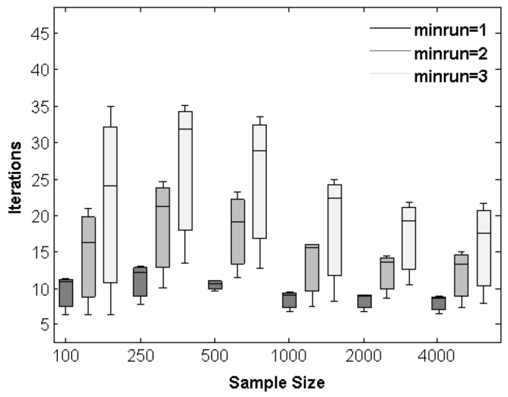
However, generally the solution quality is also reduced; as we reduce M_i, there are some hidden variable-sample size configurations where the global maximum is not attained. Table 4.3 provides a count of the number of such hidden variable-sample size configurations. Each value in the 'Failures' column is out of a total of 24 configurations for Carstarts, 30 for Alarm (which corresponds to 6 sample size configurations $\times$ the number of hidden variable configurations chosen for the network) and 18 (which corresponds to 6 sample size configurations $\times$ 3 hidden variable configurations chosen for the network) for Win95pts and Hepar2. We also provide the $\mathrm{RLS_{avg}}$ of *only those experiments* that did not achieve the global maximum (the RLS for the others would be 0). As with average iterations, we can see a consistent variation of solution



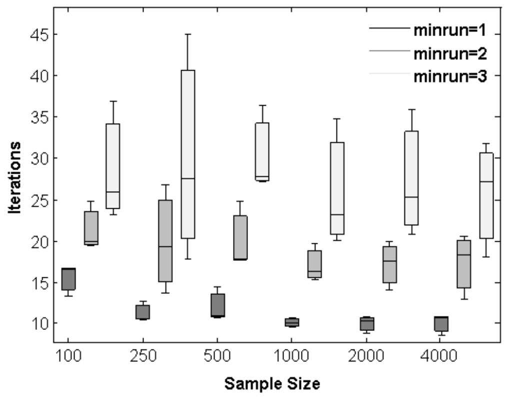
(a) Carstarts Bayesian network



(b) Alarm Bayesian network

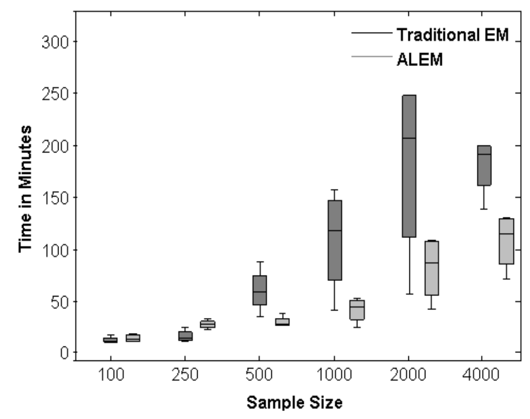


(c) Win95pts Bayesian network

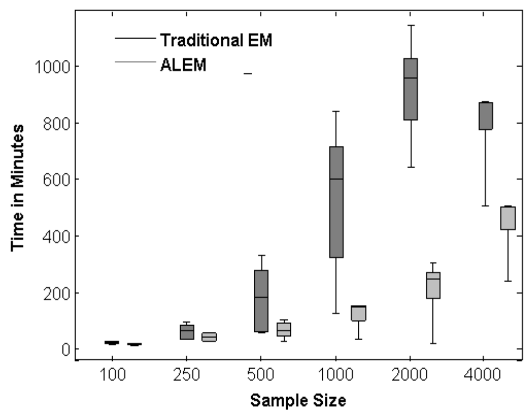


(d) Hepar2 Bayesian network

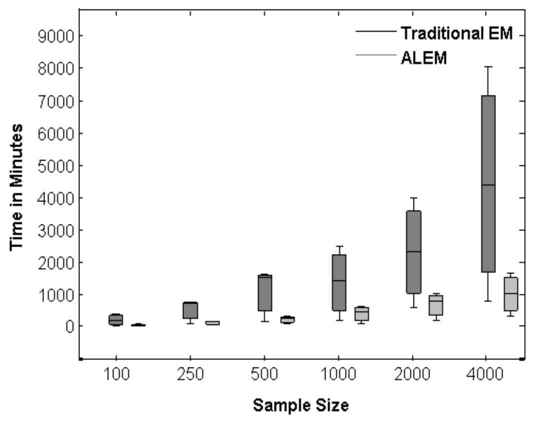
Figure 4.4: Variation in the number of iterations ALEM runs undergo, on average, as a function of the minimum runs parameter M_i for both networks. The lower M_i is, the fewer iterations undergone. Darker shades of gray denote lower values of M_i.



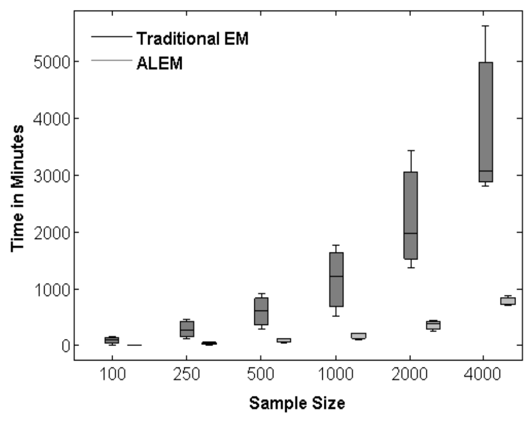
(a) Carstarts Bayesian network



(b) Alarm Bayesian network



(c) Win95pts Bayesian network



(d) Hepar2 Bayesian network

Figure 4.5: Wall clock time comparison between traditional EM and ALEM. ALEM is, for larger sample sizes, significantly faster and the variation amongst runs tends to be much smaller. Traditional EM is in dark gray, ALEM is in light gray.

quality as we vary M_i to take values between 1 and 3.

M_i can be adjusted by methods like simulated annealing where a temperature parameter can be increased initially to allow less discarding of runs for exploration of the search space and then gradually decreased to allow more discarding of EM runs. Future work will concentrate on a method to tune this parameter.

4.4.5 Wall Clock Time Comparison

We recorded the time taken by traditional EM and ALEM when running the EM experiments across all six sample sizes for the subset of hidden variables. Figures 4.5 show the box plots of the average time taken for both approaches. From these graphs we can clearly see a considerable decrease in time taken by ALEM for the Alarm and Hepar2 network. For Carstarts and Win95pts, similar results are evident, but with sample sizes 500 and above. The variance in the number of iterations is also significantly smaller in ALEM. Thus, we can conclude that a reduction in average iterations, as reported in Table 4.1 an Table 4.2, translates well to a reduction in wall-clock time, especially for larger sample sizes.

4.5 Summary

In this work, we present an age-layered influenced algorithm to mitigate the local optima problem in the EM algorithm. Specifically, our approach can be seen as a way to manage multiple EM runs, randomized with different initial starting points. We have shown that parameter estimation using EM on difficult Bayesian networks can be extremely computationally wasteful, underlying the need for an efficient method to restrict the overall average number of iterations undertaken for a given parameter estimation problem instance. We then show that ALEM manages to significantly decrease the average number of iterations required on four difficult Bayesian networks, but at the same still achieves the global optimum, or gets very close, in all instances.

Chapter 5

NetEyes: Multi-Focus Visualization Techniques

Networks analysts often need to compare nodes in different parts of a network. When zoomed to fit a computer screen, the detailed structure and node labels of even a moderately-sized network (say, with 500 nodes) can become invisible or difficult to read. Still, the coarse network structure typically remains visible, and helps orient an analyst's zooming, scrolling, and panning operations. These operations are very useful when studying details and reading node labels, but in the process of zooming in on one network region, an analyst may lose track of details elsewhere. To address such problems, we present in this chapter our NetEyes visualization software integrated with multi-focus and multi-window techniques that improve interactive exploration of networks. Based on an analyst's selection of focus nodes, our techniques partition and selectively zoom in on network details, including node labels, close to the focus nodes. Detailed data associated with the zoomed-in nodes can thus be more easily accessed and inspected. The approach enables a user to simultaneously focus on and analyze multiple node neighborhoods while keeping the full network structure in view. We demonstrate our technique by showing how it supports interactive debugging of a Bayesian network model of an electrical power system and EM learning in Bayesian networks. In addition, we show that it can simplify visual analysis of an electrical power network as well as a medical Bayesian network. This chapter is based on the multi-focus visualization technique [101] .

5.1 Introduction

Many datasets can be partly represented as networks or graphs. For example, a Bayesian network [47] provides an elegant method of representing complex relationships among random variables in the form of graphs. It is a directed acyclic graph model representing a set of variables, in the form of nodes, and their conditional dependencies, in the form of edges (see Chapter 2 for more details). Bayesian networks have become an important statistical machine learning tool and are being used effectively in uncertainty reasoning for many real world problems such as electrical power system diagnosis [69], medical diagnosis and image recognition.

As an example of an application, consider a Bayesian network for medical diagnosis. In this case, nodes may represent diseases and symptoms. The edges represent the causal relationship between diseases and their symptoms. The conditional probability table (CPT) table for each disease node reflect prior probabilities, while a symptom node CPT has probability values for the different combinations of states of its parent disease nodes. Given observed symptoms, also known as evidence, the network can be used to compute the posterior probability distribution for the presence of the diseases. Unlike other networks containing millions of nodes and edges, Bayesian networks found in most applications to date contain up to a few thousand nodes. In addition, the in- and out-degrees of Bayesian network nodes are quite small, typically a dozen or less, while some social networks nodes have thousands or millions of neighbors.

While software tools like Hugin [1] and GeNIe/SMILE [21] provide powerful visualization support for nodes, edges, and conditional probability tables (CPTs), many real-world Bayesian networks are becoming so large and complex that more advanced visualization and interaction techniques would be beneficial. Interactively investigating these networks, with hundreds or thousands of nodes and edges, can be quite challenging [47]. Existing visualization tools used for scrolling, zooming or panning the network do not elegantly combine network structure visualization with interactive exploration and understanding of node details. Even when merely moderately sized (say, with a few hundred nodes) Bayesian networks are zoomed to fit the screen, node labels may become unreadable. Consequently, an analyst may need to zoom, pan, and scroll in order to read node labels and thereby better understand the joint role of different network nodes in Bayesian network computations. Unfortunately, in the process of zooming, panning, and scrolling, an analyst may easily lose context. As an example, given a node of interest, it is hard to understand interactions with its children and parents if some of them are located far-off in the network layout. It is also difficult to keep values of the conditional probability tables (CPTs) in mind when studying and understanding zoomed-in details. The multi-focus and multi-window visualization techniques discussed in this chapter enable an analyst to better compare and analyze internal details of nodes in different parts of a network while retaining network context.

The existing fisheye technique [31] seeks to enable zooming in on details while retaining context; it was introduced to address the fundamental visualization problem of finding a specific address node in a address book structure of AT&T employees. The fisheye technique maintains context and lets users study details, but allows focus on only one part of a network. In many cases, however, a user might need to compare multiple things in multiple parts of a network. Remembering the details of a previously studied zoomed node becomes a tedious memory taxing and error prone activity as the number of nodes for comparison increases. This is the limitation of traditional single fisheye techniques.

The remainder of this chapter is structured as follows: Design goals on NetEyes are discussed in Section 5.2. Section 5.4 provides an overview of previous related work, while the multi-focus algorithm is described in detail in Section 5.5. Application and expert evaluation are discussed in Section 5.6. Finally, Section 5.7 concludes the chapter with some hints for future research directions.

5.2 Design Goals of NetEyes

To overcome the limitations of traditional fisheye technique, as well as others, we formulate a number of design goals (**DGs**) that address our requirements associated with interactive exploration and analysis of networks, in particular Bayesian networks. The design goals assume that zooming is being used. For the zooming algorithm, we integrate the previously stated goals [90] and the suggestions from Bayesian experts. We hypothesized that the inherent complexity in Bayesian network representations could be overcome with visualization tools that focus on comparing parts of the network and their contents.

DG1 *Multi-focus zooming*: Multiple focus nodes selected in different parts of the network should be zoomed simultaneously, under user control, thus making their labels more readable regardless of their location in the network. The process of zooming in and zooming out should be animated, to avoid hard to follow abrupt layout changes.

DG2 *Topology maintenance*: The continuity of layout adjustment should not challenge the user's mental map of the structure of the network, that is the topology, and the proximity relations of the nodes should be maintained [99].

DG3 *Focus nodes selection*: The user should be able to determine which nodes to zoom by studying the details associated with a subset of nodes. A similar set of nodes such as current nodes in an electrical network or disease nodes in a medical network, should be selectable by means of a search operation. A particular section of the network should also be selectable by means of a group selection operation.

DG4 *Scoped zooming*: The zooming should be restricted to a region of interest or a partition, around a focus node to avoid distorting the whole network layout. A partition algorithm should intuitively decide on the region of interest around the focus node for zooming. The set of nodes in the network should be partitioned accordingly. There is one partition per focus node. With no focus node, there is one partition for all the nodes. The partitions around the focus nodes is used to localize the zooming effect within that region.

DG5 *Dynamic partitioning*: The partitioning algorithm should dynamically adjust the partitions as the user interactively adds or removes focus nodes for scoped zooming.

DG6 *No ghost regions*: After zooming, discontinuities or 'ghost regions' between the partitions should not exist [10].

DG7 *Context visibility*: We consider both local and global context. For local context visibility, the user should be able to control zooming to make the region surrounding the focus node more visible. For global context, the whole network should fit to the screen space and be visible during the user interactions. The user should still be able to control zooming the whole network.

DG8 *Label zooming*: The user should be able to make the node labels of focused nodes bigger and more easily readable. The degree of zooming decreases as we move away from a focus node.

DG9 *Data exploration*: Users should be able to simultaneously open multiple detail windows associated with the network nodes. Depending on the type of network, the data-level windows should show node details such as CPTs, time-series graphs, or individuals bio-data for comparison or data analysis.

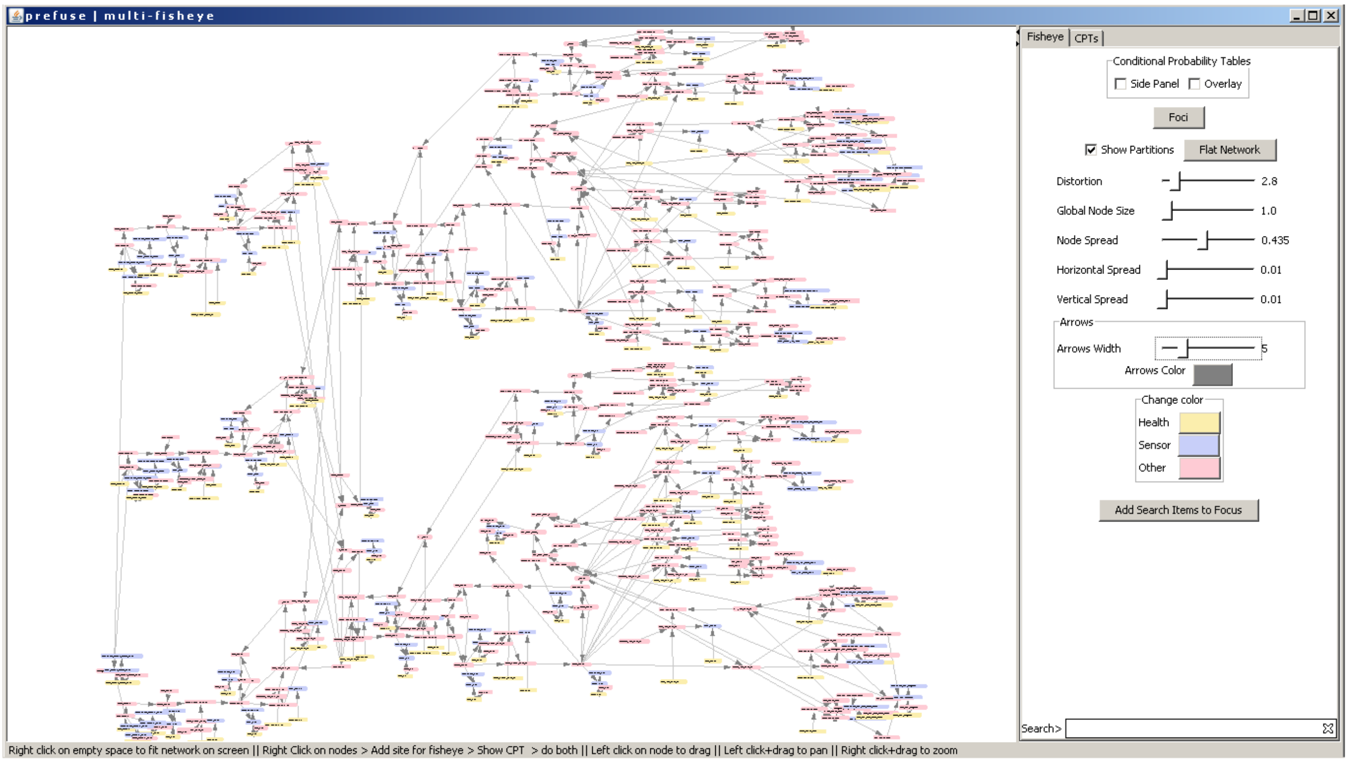
Existing visualization techniques do not integrate all of the above design goals. We introduce in this chapter a multi-focus technique to help analysts compare different parts of a network simultaneously while retaining network structural context. Our novel approach satisfies the above goals and supports the visualization principles “overview first”,“ zoom and filter”, then “details-on-demand” [98].

5.3 NetEyes

Our software tool illustrated in Figure 5.1 has several GUI components. Our multi-focus technique starts with an initial layout in the *network window* as shown in Figure 5.1(a). It allows analysts to select a set of focus nodes to zoom-in on multiple parts of a network. In order to minimize distortions while zooming, we localize a fisheye-like effect to a region surrounding a focus node. The distortion neighborhood is defined by partitioning the display into polygonal regions, using a Voronoi algorithm [30]. All the nodes inside a polygonal region are closer to the focus node in that region than to the focus nodes in other regions. A localized fisheye zooming is applied to all partition regions, several such zoomed partitions can be created at the same time allowing users to zoom several parts of the network without losing structural context (see Figure 5.1(b)). The degree of distortion can be adjusted by using the slider in the *control panel* as shown in Figure 5.1. Our analysis and visualization is targeted on two tasks, the validation task and the diagnosis task (explained in more detail in Section 5.6.2). In the validation task, the analyst understands and checks the overall network by inspecting various nodes in different parts of the network. In the diagnosis task, the analyst focuses on the causal nodes that lead to a particular outcome, for an in-depth comparison and analysis. The analysts can also use the search operation in the control panel for selecting the nodes. There is also an option to show all available node details, in a *dataview window*.

5.4 Related Work

A number of existing techniques partially fulfill the design goals (DG1 - DG9) outlined above. This section discusses how visualization, visual distortion, multi-focus and interactive visual-



Conditional Probability Tables
☐ Side Panel ☐ Overlay

☑ Show Partitions

Distortion 2.8
Global Node Size 1.0
Node Spread 0.435
Horizontal Spread 0.01
Vertical Spread 0.01

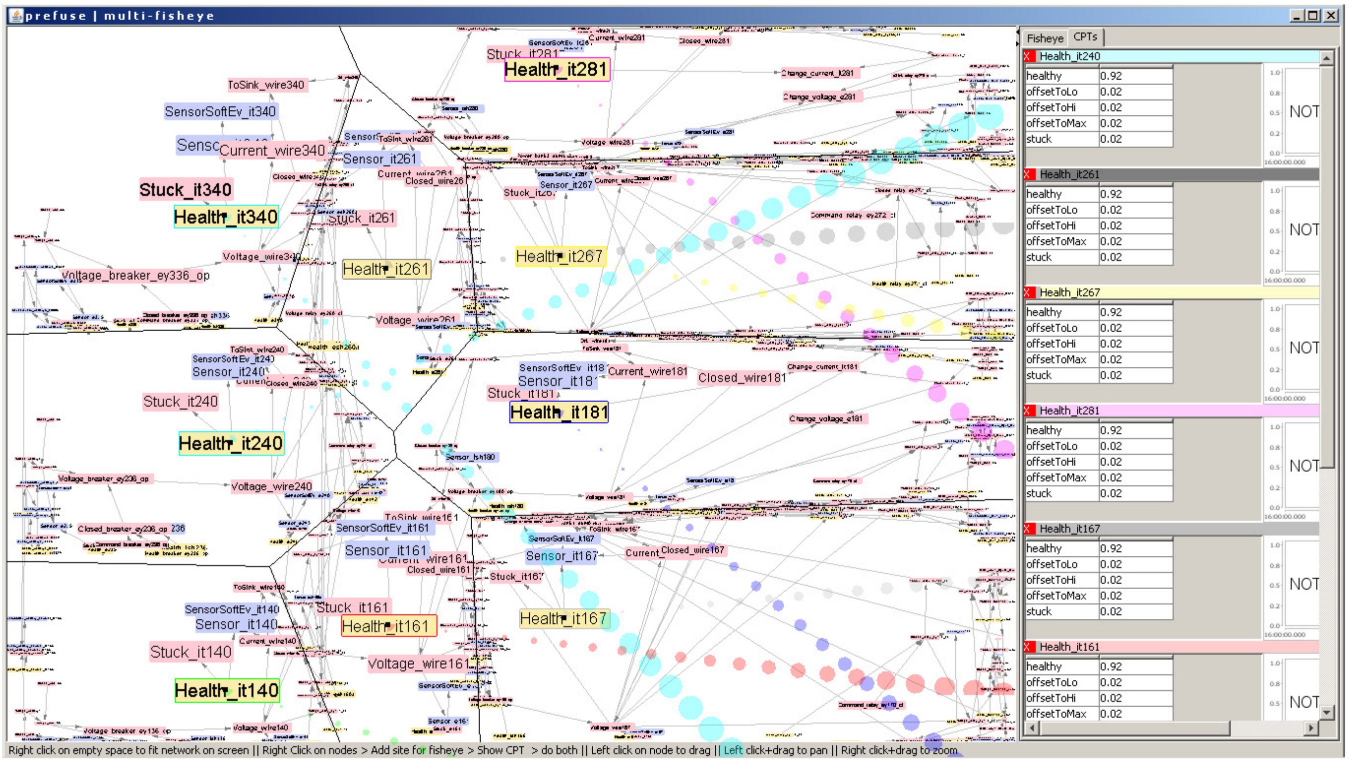
Arrows
Arrows Width 5
Arrows Color ▨

Change color
Health 🟨
Sensor 🟦
Other 🟥

Search>

Right click on empty space to fit network on screen || Right Click on nodes > Add site for fisheye > Show CPT > do both || Left click on node to drag || Left click+drag to pan || Right click+drag to zoom

(a) Before fisheye zooming. Node labels are unreadable.



Health_it240

healthy	0.92
offsetToLo	0.02
offsetToHi	0.02
offsetToMax	0.02
stuck	0.02

Health_it261

healthy	0.92
offsetToLo	0.02
offsetToHi	0.02
offsetToMax	0.02
stuck	0.02

Health_it267

healthy	0.92
offsetToLo	0.02
offsetToHi	0.02
offsetToMax	0.02
stuck	0.02

Health_it281

healthy	0.92
offsetToLo	0.02
offsetToHi	0.02
offsetToMax	0.02
stuck	0.02

Health_it167

healthy	0.92
offsetToLo	0.02
offsetToHi	0.02
offsetToMax	0.02
stuck	0.02

Health_it161

healthy	0.92
offsetToLo	0.02
offsetToHi	0.02
offsetToMax	0.02

Right click on empty space to fit network on screen || Right Click on nodes > Add site for fisheye > Show CPT > do both || Left click on node to drag || Left click+drag to pan || Right click+drag to zoom

(b) After fisheye zooming. Node labels are readable.

Figure 5.1: Visualization of multi-variate probability distributions of ADAPT electrical Bayesian network. *Side panel* in Figure 5.1(b) shows the *multiple detail windows*. The bubbles help the user to trace a detail window to its corresponding node in the network view.

ization approaches might each improve on a user's ability to understand and solve problems in networks such as Bayesian networks.

5.4.1 Network Based Fisheye Techniques

A number of focus+context display techniques have been introduced [14]. A powerful and popular way to retain overview and detail is the fisheye approach [31]. In a single-fisheye view, the entire node structure can be always visible. The user-selected node and its neighboring network nodes are magnified and distant nodes in terms of graph structure are demagnified. The fisheye techniques described by Sarkar and Brown [95] use filtering and distortion, but support focus on a single item and distort the whole layout. An improved spatial distortion approach for the focus+context transformation using hyperbolic geometry is presented [55]. It places nodes around the root and provides smooth and continuous animation as users click or drag nodes to read the focus point of the layout. This approach also distorts the whole layout for each selection of focus point. In these techniques, some of our design goals are not satisfied, such as multi-focus zooming (DG1), topology maintenance (DG2), focus nodes selection (DG3), scoped zooming (DG4), dynamic partitioning (DG5) and data exploration (DG9). The design goals that are satisfied are continuity in layout (DG6), context visibility (DG7) and label zooming (DG8). Topology maintenance (DG2) is achieved in Furnas's fisheye approach [31] but not in the hyperbolic approach [55].

Zooming techniques with orthogonal and polygonal stretching are investigated [96]. The simple orthogonal distortion method maintains topological ordering of points (nodes maintain their left-of, above, etc. relationships), but the polygonal method does not. The zooming action is such that a user acts indirectly on the focus nodes through a 'rubber sheet.' This technique unfortunately violates the design goal, no ghost regions (DG6). Formella and Keller [29] distort network layout outside a polygonal area to make space for zooming all nodes inside the circumscribed polygon. This distortion mechanism does not scale to large networks as the user has to manually select the focus area by using a rectangular selection. Both these techniques do not support dynamic partitioning (DG5).

A topological fisheye method [33] precomputes coarsened graphs and renders the level of detail from the combined graphs, depending on the distance from one or more foci. This system supports more than one focus (DG1). The drawback, however, is the computation involved in pre-computing the coarsened graphs, making it unsuitable for interactive exploration. Dynamic insets [34] uses the connectivity of the graph to bring offscreen neighbours of on-screen nodes and their context into the viewport as insets. Unfortunately, the entire structure of network is not visible in this technique, thus violating DG7. A multi-focus+context technique [60] for generating spatiotemporal coherent time-varying graphs is introduced. This technique utilizes a triangle mesh to partition the graph nodes and leverages this underlying mesh for constrained multi-focus+context visualization. Multi-focus+context visualization was achieved through formulating an energy function for optimized deformation. This approach supports design goals such as the topology maintenance (DG2), continuity in layout (DG6) and context visibility (DG7). The above approaches do not support design goals such as focus nodes selection (DG3), scoped zooming (DG4), dynamic partitioning (DG5), label zooming (DG8) and data exploration (DG9). Other zoom algorithms [4] [97] provide more scalable multi-focus distortions, but without scoping of distortion, any focus change affects the entire network layout in these approaches. Also, the user has no direct control over the sizes of nodes aside from opening or closing them.

As described, previous techniques do not support one of our design goal, namely scoped zooming (DG4), as they distort the whole layout when rendering graphs at different levels. While the issue with distorting the whole layout is addressed by an improved fisheye zoom algorithm [90], it results in wasted screen space called 'ghost regions,' thus violating DG6.

5.4.2 Tree Based Fisheye Techniques

Several previous systems demonstrate focus+context techniques for tree visualization, like SpaceTree [86], which uses extensive zooming animation to help users stay oriented within its focus+context tree presentation. Unfortunately, this technique does not support focus nodes selection (DG3), scoped zooming (DG4), dynamic partitioning (DG5), and creates ghost regions (DG6). The TreeJuxtaposer [74] technique uses focus+context methods to support comparisons across hierarchical datasets. The technique also creates ghost regions (DG6). The reason is that this technique allows the user to do a rectangular selection which usually gives rise to 'ghost regions'.

Tu and Shen present 'balloon focus,' a multi-focus context technique for treemaps. Their user study confirms that users prefer the multi-focus treemaps to identify select players in a multi-year NBA dataset consisting of conferences, divisions, teams and player [104]. While the treemaps provide good usage of the available space, network structure can be difficult to identify [7]. So this approach does not retain the topology of the network (DG2). Bayesian networks are in general not trees, making tree visualizations limiting.

5.4.3 Image Based Fisheye Techniques

A new distortion technique that folds the space between focus regions to guarantee visibility of multiple focus regions is proposed [26]. The folds themselves show contextual information and support unfolding and paging interactions [26]. A drawback of this technique is the lack of user control over the scope of the focused regions. Non-linear magnification [49], pliable surfaces [11] and compressed arc tangent graph algorithm [48], when applied to graphs, distort the labels within the zoomed areas. Such distortions make labels difficult to recognize or read (DG8). These techniques support multi-focus zooming (DG1), topology maintenance (DG2), dynamic partitioning (DG5), no ghost regions (DG6), and context visibility (DG8). But they do not support focus nodes selection (DG3), scoped zooming (DG4) (as the whole layout is distorted for each selection of focus point), label zooming (DG8) and data exploration (DG9).

5.5 Multi-focus Zooming and Multi-window Techniques

Our *multi-focus* zooming algorithm helps to retain the network structure, thus limiting distortion to preserve the user's mental map of the Bayesian network. The distortion algorithm is independent of the graph or network layout algorithm and is defined as a separate processing step on the layout of the graph. This allows for a modular organization of software [38] and helped us to easily understand, modify and reuse existing code to suit our visualization. However, care must be taken for the fisheye distortion not to reduce readability of the display. To combat the negative aspect of distortion while giving an analyst control, we take the three steps shown in Figure 5.2:

- In the *focus nodes selection* step, the analyst selects a set of nodes for zooming. This can be done using a search operation over node labels or manually by selecting interesting nodes from the dataview window where all the detail windows of the nodes are displayed.

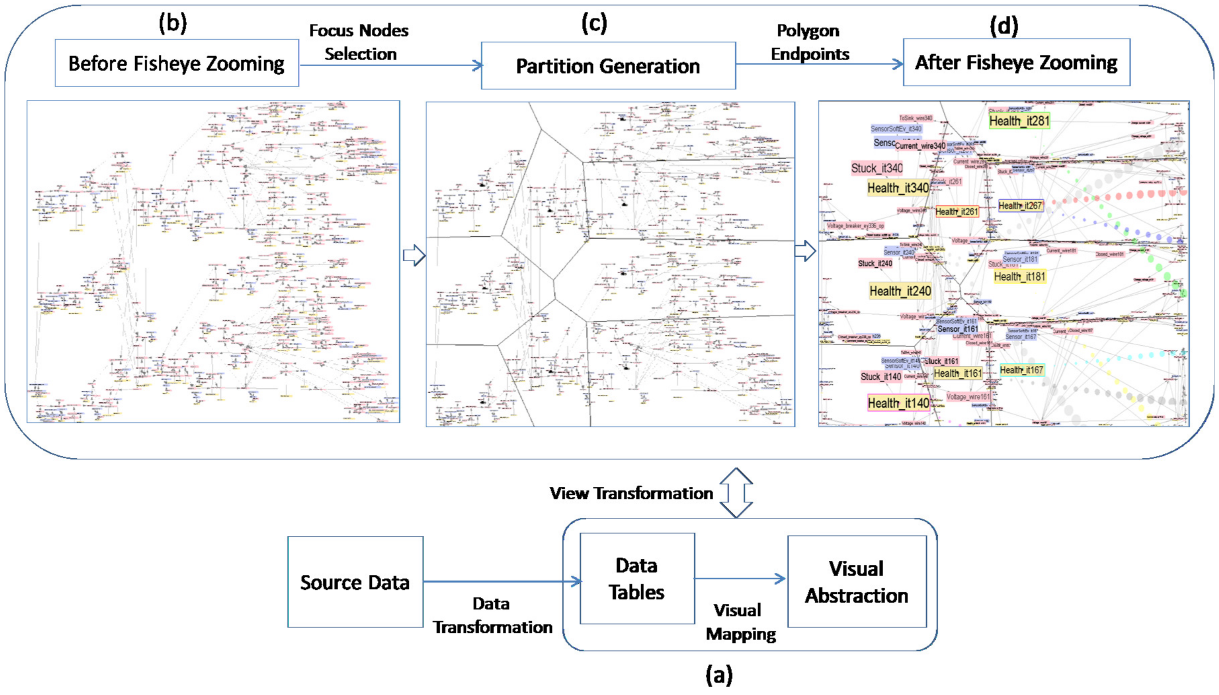


Figure 5.2: The diagram depicts the visulaization pipeline flow in our multi-focus algorithm. (a) The source data is loaded into Prefuse data tables and converted to visualizable attributes in the visual abstraction. View transformation takes place in three main steps: (b) before fisheye zooming; (c) partition generation; and (d) after fisheye zooming, also showing bubble anchors.

- After selection of the focus nodes, regions around the focus nodes are created by the *partition generation* step, which applies an incremental algorithm that maintains a set of partitions that varies over time by insertion or deletion [2]. This is shown in Figure 5.2c which has nine focus nodes and polygonal partitions. This second step also ensures that all the nodes inside one partition are within that partition after distortion as shown in Figure 5.2d. This is done by measuring the maximum distance to move the node during distortion; the black lines as shown in Figure 5.2c denotes the maximum distance to position the node so that it stays within the Voronoi partition.
- The third step, *fisheye zooming*, distorts each partition as shown in Figure 5.2d. This results in zooming the focus nodes. For each focus node, the zooming gradually decreases as we approach the edges of the Voronoi partition.

The above three steps repeat as the new focus nodes are selected and zoomed. Our technique is based on distorting the size and position of the label box based on its euclidean distances from the focus node. This helps to identify the focused node and its neighboring nodes. The process of zooming in our approach therefore attempts to provide interactive rendering, ability to compare multiple parts of a graph without excessive distortions and ghost regions, and exploration of node details.

Both Voronoi and rectangular partitioning approaches for a multi-focus technique have been proposed [100]. We use the Voronoi partitioning approach [3] as opposed to the traditional rectangular partitioning approach [100]. This aids incremental partitioning (DG5) and prevents ghost regions(DG6) while preserving both structure and efficient use of space for arbitrary network structures. Another benefit of the Voronoi approach is that it does not create ghost regions (DG6). After applying the fisheye technique locally inside each Voronoi polygon, the nodes near the sides of all polygons are compressed, to preserve layout continuity. Previous work has, to our knowledge, not combined the Voronoi and the fisheye algorithm for multi-focus zooming.

In addition to multi-focus, our approach is *multi-window*. The recent GraphPrism [10] shows graph measures in stacked histograms and highlights nodes in a network based on selections in the histograms. We follow, and use multiple small windows to show more details such as the CPTs of the nodes. Having multiple windows with node detail information raises the question of maintaining a connection between a node and its details. We do this by means of 'lines of bubbles' (see Figure 5.5). Like bubbles connecting thoughts to a character in a cartoon, bubbles act as anchors and connect nodes to their details. Using these bubbles, the user can trace a detail window either in the side panel or floating, to its corresponding node in the network view. The bubbles and the title bar of the detail window have the same color to clearly show their connection. We use an improved version of previously used bubbles [16], replacing the solid row of large dots with a progressively enlarged row of hollow bubbles. By using hollow bubbles, the user can now more easily see the network underneath the bubbles, see Figure 5.7 for solid bubbles and Figure 5.5 for hollow bubbles. We experimented with different bubble color representations, solid versus hollow bubbles, and different sizes of the bubbles. We found that hollow colored bubbles which progressively increase in size as it reaches the side panel are more effective.

The set of nodes in the network is denoted by $\mathbf{X} = \{X_1, X_2 \ldots X_n\}$. The focus nodes are stored in a list $\mathbf{Y}$. Each focus node is associated with a polygon ρ. The polygons are stored in a list $\boldsymbol{\rho} = (\rho_1, \rho_2, \ldots, \rho_m)$. The pseudo-code of the multi-focus algorithm is shown in Algorithm 5.5.1; we now discuss each of the three steps in more detail.

5.5.1 NetEyes Step 1: Selection of Focus Nodes

Many current visualization techniques mainly address how to display the data while the user's primary concern, especially for large datasets, is what are interesting nodes $\mathbf{Y}$, that need to be focused [31, 32]. In NetEye, we provide several options as discussed below to help users to study the details associated with each node X_i to find the truly interesting nodes $\mathbf{Y}$ (line (viii) in Algorithm 5.5.1), satisfying design goal, focus nodes selection (DG3). We denote a focus node as X_f^*.

1. The user can study the details of the nodes (via detail windows) such as the time-series graphs or the CPTs (DG9). If an interesting behavior is found, the detail window can be selected so that the corresponding node in the network window also gets selected.
2. In the network view, the detail windows can be viewed as a tooltip as the user hovers over a node with a mouse. The detail window associated with a node can be anchored if the time-series graph or the CPT requires further inspection, see Figure 5.5.
3. A search operation can be used to select a set of nodes. This operation is useful, for example, when the user wants to study all the current nodes or the voltage-sensor nodes in

```
Algorithm 5.5.1: MULTI-FOCUS($\boldsymbol{X}$)

 procedure FISHEYEZOOMING($\boldsymbol{\rho}, \boldsymbol{Y}$)
  comment: Apply fisheye distortion for each node
  for each $\rho_i \in \boldsymbol{\rho}$
   do
    $X_f^* \leftarrow getFocus(\boldsymbol{Y}, \rho_i)$
    for each $X_j \in \boldsymbol{X}$
     do
      Ray casting is used to find if a node is inside a polygon
      if $(IsNodeInPolygon(X_j, \rho_i))$
       then
        Get the intersectPoint $(x_p, y_p)$ of the node
        $(x_p, y_p) \leftarrow intersectPoint(X_j, X_f^*, \rho_i)$
        Compute $D_{max}$                                                        (i)
        $D_{max} \leftarrow \sqrt{(x^* - x_p)^2 + (y^* - y_p)^2}$
        Get start and end distance ($D_s$ and $D_e$) of the node $X_i$ from the focus $X_f^*$   (ii)
        $D_s \leftarrow \sqrt{(x^* - x_s)^2 + (y^* - y_s)^2}$
        $D_e \leftarrow \sqrt{(x^* - x_e)^2 + (y^* - y_e)^2}$
        Apply arcTan fisheye distortion for the node
        Conversion from Cartesian to Polar co-ordinates                          (iii)
        $\theta_s = \arctan(\frac{y^* - y_s}{x^* - x_s})$
        $\theta_e = \arctan(\frac{y^* - y_e}{x^* - x_e})$
        Normalize the distance                                                   (iv)
        $d_s^{norm} = D_s / D_{max}$
        $d_e^{norm} = D_e / D_{max}$
        Calculation of radial distance $r$ and de-normalization                  (v)
        $r_s = a * \arctan(b * d_s^{norm}) * D_{max}$
        $r_e = a * \arctan(b * d_e^{norm}) * D_{max}$
        Conversion from Polar to Cartesian co-ordinates                          (vi)
        $x_s' = r_s \cos(\theta) + x^*$
        $y_s' = r_s \sin(\theta) + y^*$
        $x_e' = r_e \cos(\theta) + x^*$
        $y_e' = r_e \sin(\theta) + y^*$
        Compute the new font size and position                                   (vii)
        $x_c' = (x_e' - x_s')/2$
        $y_c' = (y_e' - y_s')/2$
        $X_j.position \leftarrow setPosition(x_c', y_c')$
        $X_j.font \leftarrow setFont((x_e' - x_s'), (y_e' - y_s'))$

 main
  Graph layout is rendered and wait for user operation
  if $(X_i.selected)$
   then
    User can select nodes to start analysis                                      (viii)
    $\boldsymbol{Y} \leftarrow \boldsymbol{Y} \cup X_i$
  Fortune's Voronoi algorithm is called to create the polygons                   (ix)
  $\boldsymbol{\rho} \leftarrow$ DRAWVORONOI($\boldsymbol{Y}$)
  FISHEYEZOOMING($\boldsymbol{\rho}, \boldsymbol{Y}$)
```

Figure 5.3: Pseudocode for the multi-focus algorithm. The DrawVoronoi function takes the focus nodes ($\boldsymbol{Y}$) as inputs and outputs the endpoints of the polygons ($\boldsymbol{\rho}$) for each focus node. The FisheyeZooming function takes the $\boldsymbol{\rho}$ and the $\boldsymbol{Y}$ as inputs and renders the distorted nodes.

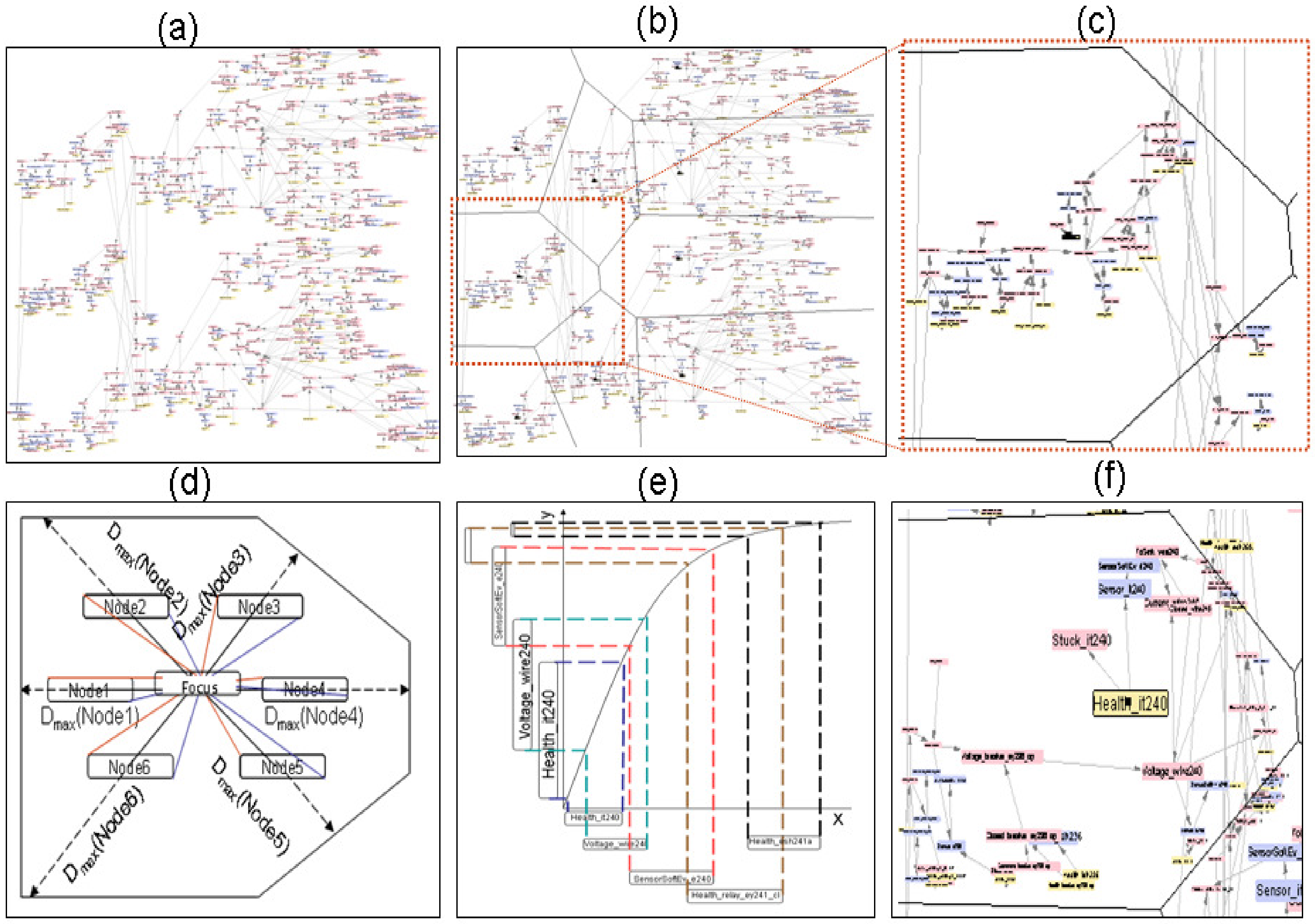


Figure 5.4: (a) A hard to read baseline network; (b) Voronoi partition lines have been drawn; (c) Viewing one zoomed-in partition; (d) Showing, in principle, how the maximum distance for each node label positions are computed (so that they do not move out of the polygon). Blue and red lines show the start and end distance of each node label rectangle from the focus; (e) arcTan distortion is applied for some node label rectangles based on the start(upper-left) and end(lower-right) coordinates; (f) Polygon shows focus nodes after distortion.

an electrical network or in a Bayesian network, see Figure 5.7.

4. Rectangular selection allows the user to select a group of nodes in a particular region of the network layout. After selection, the detail windows associated with those nodes can be opened and studied in the side panel, see Figure 5.5.
5. Users can select neighboring nodes in a graph. The selected nodes can be zoomed and studied, see Figure 5.8 and 5.10.

5.5.2 NetEyes Step 2: Partition Generation

After selecting a set of focus nodes $\mathbf{Y}$, a bounded region ρ around each of the focus node X_f^* should be automatically generated by the partitioning algorithm. The zooming algorithm is applied inside this region.

We experimented with a variety of rectangular partitioning approaches but found them causing discontinuities [100]. The Voronoi algorithm [2] satisfies several design goals (DG4, DG5 and DG6); its works by dividing the display area into n polygonal regions $\boldsymbol{\rho}$, given n node selections (line (ix) in Algorithm 5.5.1). This algorithm is based on the principle that any node in the region will be nearer to the focus node in that region than to any other focus node.

When a node is selected as a focus node, a partitioning algorithm [2] is applied to that node and neighboring existing focus nodes to generate a new polygon, showing the incremental aspect of the algorithm. We apply the local fisheye to a bounded area by retrieving the corner coordinates of the region and updating the display accordingly. Each node in the graph is checked to see if it is present in the selected nodes' partitioned area using a ray casting technique.[1] Only those nodes in the selected nodes' partitioned area undergoes the fisheye distortion.

5.5.3 NetEyes Step 3: Fisheye Zooming

The analyst may want study the focus nodes $\mathbf{Y}$ in each partition. To do this, he or she may use NetEyes to zoom-in on the focus nodes. A local fisheye effect does this; the selected focus node is zoomed in and the sizes of the nearby nodes increase as a by-product. We minimize the traditional fisheye effect that distorts the whole layout by localizing the fisheye effect within a Voronoi partition. The user-selected node and its nearby nodes are magnified (DG7); the size of

[1] `http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/pnpoly.html`

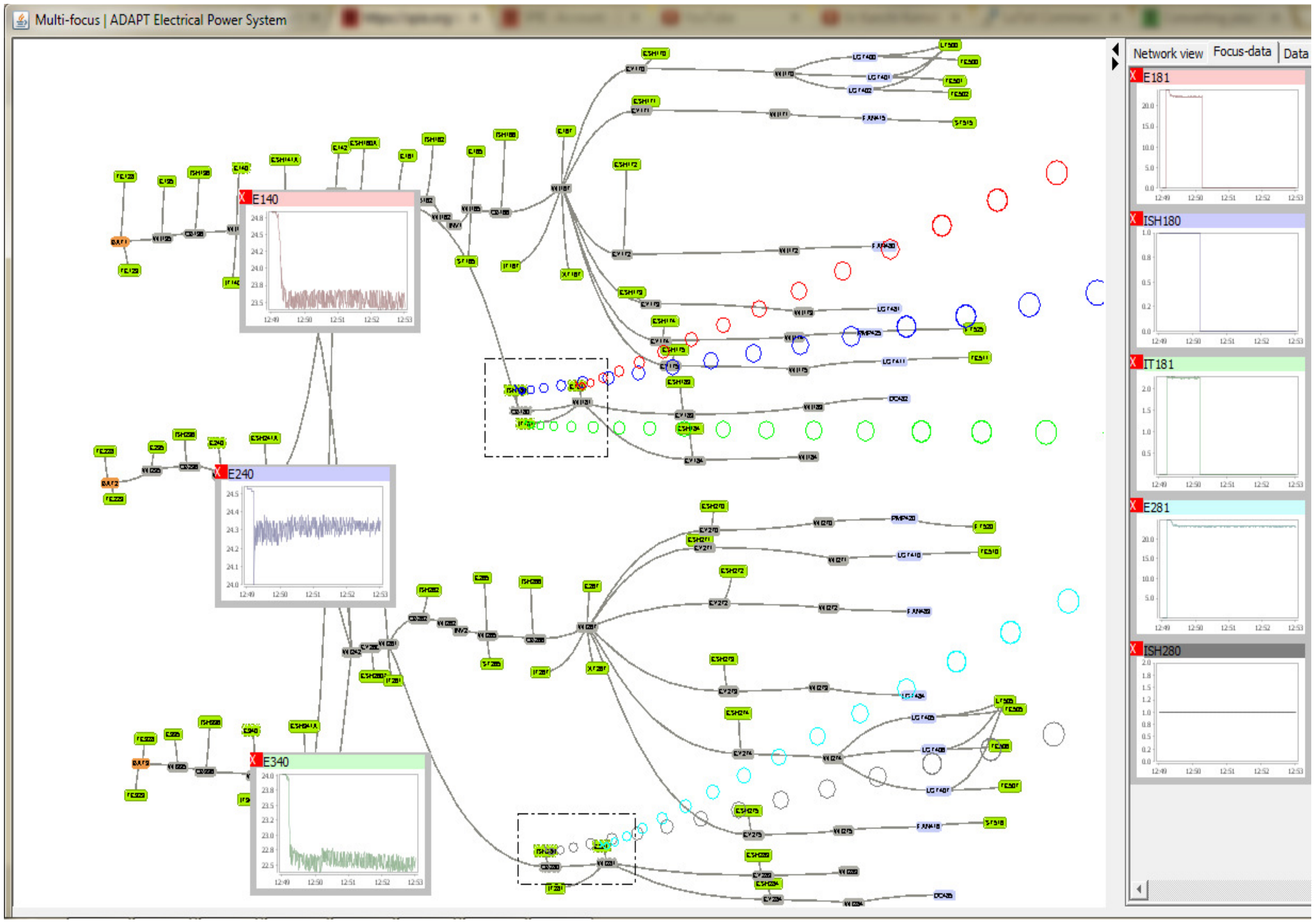


Figure 5.5: The time-series graphs of all the nodes inside the rectangular selections are aligned and shown in the side pane of NetEyes. Analysts can hover over a node to display its time-series data as a tooltip. The time series data for the voltage sensors E140, E240 and E340, shown as floating windows, have been anchored in the network view by clicking on the network nodes.

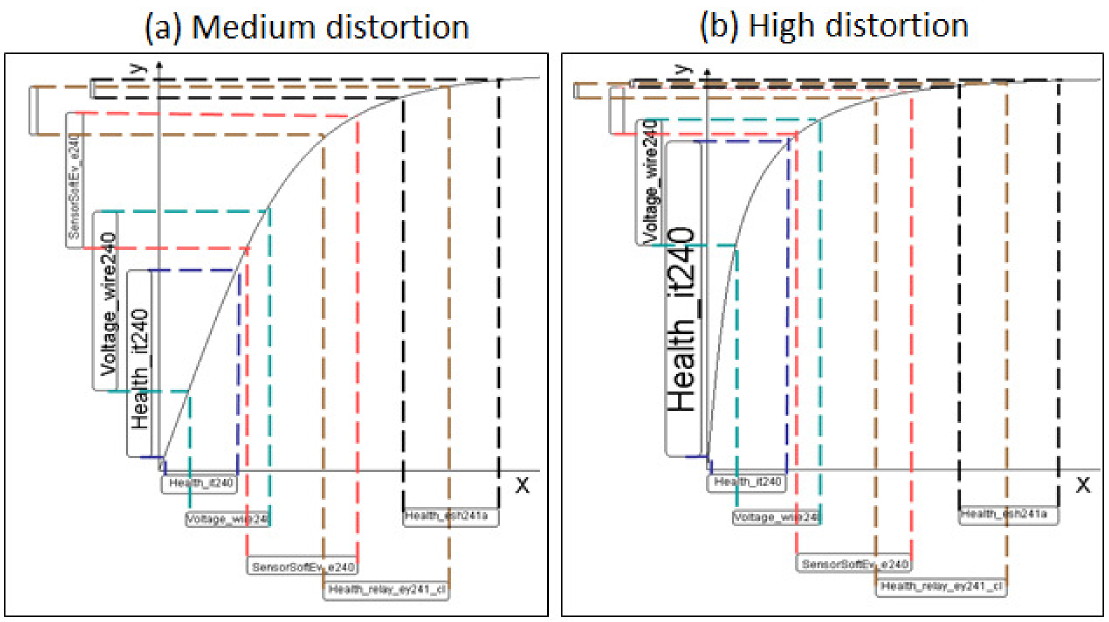


Figure 5.6: Arctan curve for different values of the distortion factor, b and its effect on the node sizes: (a) $b = 2.5$ (b) $b = 5.0$. The original undistorted nodes are shown along the x-axis, while the nodes after distortion are shown along the y-axis. It is hard for an analyst to read the small labels along the x-axis, while the labels along the y-axis close to $(0, 0)$ have become easier to read.

the nearby nodes decrease by the arctan of the distance from the focus node as they reach the edge of the polygonal partition providing a continuous layout.

Each node in the network has start coordinates (upper-left corner) (x_s,y_s), center coordinates (x_c,y_c) and end coordinates (lower-right corner) (x_e,y_e). Let (x, y) denote either the start or the end coordinates of the node. The center coordinates of the focus node are denoted by (x^*,y^*). The start and end distance of a node from the focus is denoted by D_s and D_e respectively. In Figure 5.4(d), D_s is indicated by the red line and D_e is indicated by the blue line. The distance D_{max} is measured as the distance from the focus node's center (x^*,y^*) through the node's center (x_c,y_c) to the point of intersection with the edge of the polygon as shown in Figure 5.4(d) (line (i) in Algorithm 5.5.1). These distance values are used to compute the transformed start (x'_s, y'_s) and end (x'_e, y'_e) co-ordinates of the node label box. The font size of the labels are then determined based on the difference between the transformed start and end co-ordinates in the X-dimension. The new position (x'_c, y'_c) of the node is obtained by finding the center coordinates. The arctan fisheye distortion for start or end co-ordinates (x,y) is done in the following steps:

Finding the distance of the node (x, y) (where (x, y) can represent either the start coordinates

(x_s,y_s) or the end coordinates (x_e,y_e) of a node) from the focus (x^*, y^*) is done (line (ii) in Algorithm 5.5.1) as shown below:

$$D = \sqrt{(x - x^*)^2 + (y - y^*)^2},$$

where D = D_s or D = D_e and (x,y) = (x_s,y_s) or (x,y) = (x_e,y_e).

Conversion from Cartesian to Polar co-ordinates is done (line (iii) in Algorithm 5.5.1) as shown below:

$$\theta = \arctan(\tfrac{y-y^*}{x-x^*}).$$

We normalize the distance so that the node is not moved outside its Voronoi polygon (line (iv) in Algorithm 5.5.1) as shown below:

$$d^{norm} = D/D_{max}.$$

Calculation of radial distance r and de-normalization is done (line (v) in Algorithm 5.5.1) using:

$$r = a * \arctan(b * d^{norm}) * D_{max},$$

where b is the distortion factor and a = $\frac{1}{\arctan(b)}$ so that $r \in [0, 1]$ before de-normalization.

Distortion can be increased or decreased by respectively increasing or decreasing the value of the distortion factor, b. We use the arctan function to distort; see Figure 5.6 for example arctan curves and their effects on the sizes of nodes including labels for different values of b. The topology or the relative position of the nodes are retained after the distortion (DG2).

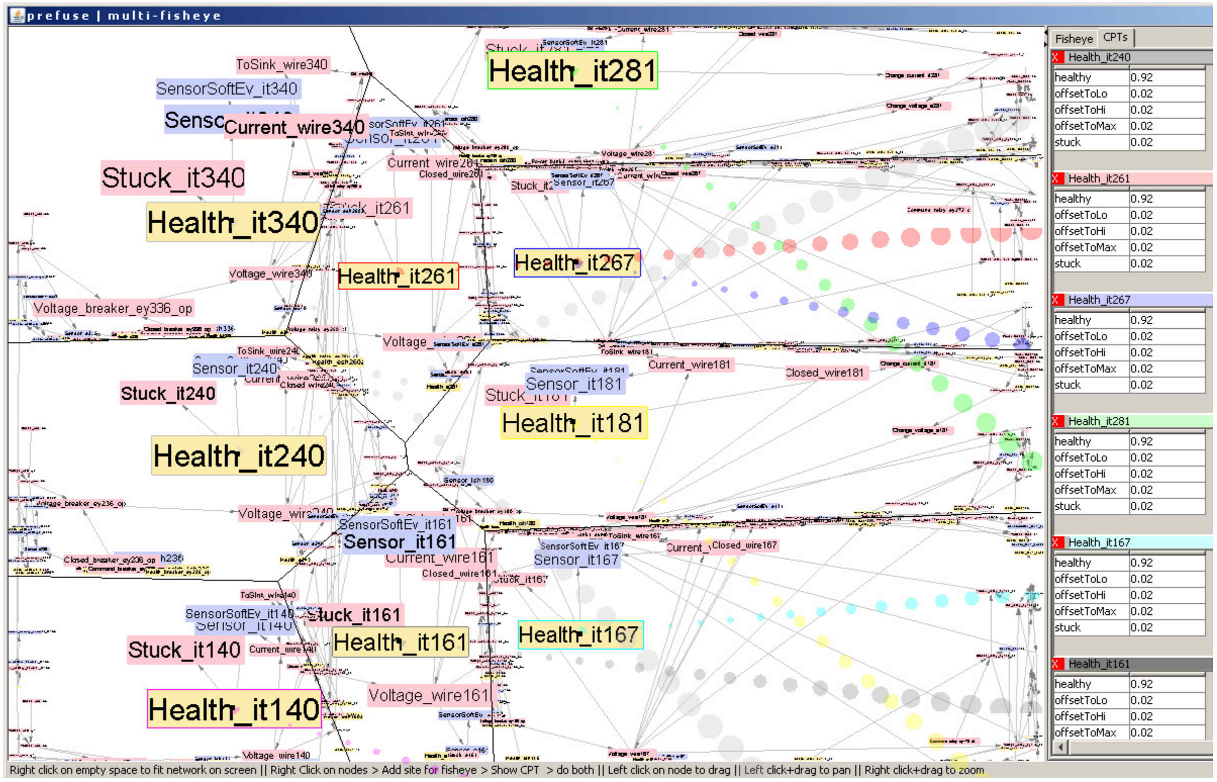
Conversion from Polar to Cartesian co-ordinates is done (line (vi) in Algorithm 5.5.1) using:

$$x' = r\cos(\theta) + x^*$$
$$y' = r\sin(\theta) + y^*.$$

The above steps are applied to both the start and end coordinates of a node to get the transformed coordinates (x'_s,y'_s) and (x'_e,y'_e). Size distortion is then done by finding the new label based on the difference between the transformed coordinates ($x'_e - x'_s$), where $x'_e \geq x'_s$ (line (vii) in Algorithm 5.5.1). The new center position (x'_c, y'_c) of the node is calculated as:

$$x'_c = (x'_e - x'_s)/2$$
$$y'_c = (y'_e - y'_s)/2.$$

Figure 5.7 shows the multi-focus zooming effect on nine focused nodes for a distortion factor $b = 18$ (DG1). The health node labels (Health_it281, Health_it340, Health_it261, ..) are all clearly readable (DG8).



Health_it240	
healthy	0.92
offsetToLo	0.02
offsetToHi	0.02
offsetToMax	0.02
stuck	0.02

Health_it261	
healthy	0.92
offsetToLo	0.02
offsetToHi	0.02
offsetToMax	0.02
stuck	0.02

Health_it267	
healthy	0.92
offsetToLo	0.02
offsetToHi	0.02
offsetToMax	0.02
stuck	0.02

Health_it281	
healthy	0.92
offsetToLo	0.02
offsetToHi	0.02
offsetToMax	0.02
stuck	0.02

Health_it167	
healthy	0.92
offsetToLo	0.02
offsetToHi	0.02
offsetToMax	0.02
stuck	0.02

Health_it161	
healthy	0.92
offsetToLo	0.02
offsetToHi	0.02
offsetToMax	0.02

Figure 5.7: A zoomed-in visualization of nine focus nodes (Health_it281, Health_it340, Health_it261, ..) using the distortion factor $b = 18$. The health node labels are all clearly readable. The focus node Health_it281 has a green color border; the bubbles connecting the node and the title bar of the detail window in the side panel are also represented in green color.

5.6 Applications of NetEyes

Our novel multi-focus technique is implemented in Java using the Prefuse framework [37]. ADAPT Bayesian network as well as other networks have been visualized. We discuss the Bayesian network visualization in Section 5.6.1. Analytical tasks, to validate the CPTs and diagnosis tasks, to find the faulty components are described in Section 5.6.2. We visualize the progress of the CPT values and the solution quality (log likelihood) of the traditional EM algorithm and GAEM in Section 5.6.3.

5.6.1 Network View for Bayesian Networks

The ADAPT Bayesian network, which can be used for automatic fault diagnosis [69], models an electrical power network that is representative of those found in aerospace vehicles. A visualization of the ADAPT network in which labels are hard to read (and with no distortion) is shown in Figure 5.1(a). The network consists of 671 nodes and 790 edges. The font size of the labels is the same for all the nodes. With the network fit to screen, as it is here, it is impossible to read these labels on the computer screen making it extremely difficult to understand, validate or edit the Bayesian network.

The ADAPT testbed has capabilities for power storage, distribution, and consumption, and contains batteries, electromechanical relays, circuit breakers, and different kinds of loads, such as pumps, fans, and light bulbs. Each component in ADAPT is modeled as a set of nodes. Health nodes ($\mathbf{H}$) and the evidence ($\mathbf{e}$) are of particular interest. The health nodes serve as the query variables, *e.g.,* whether a component is defective or not, and the evidence nodes serve as the input variables, *e.g.,* a command such as closing a relay to allow current from the battery to flow to the load. As an example fault scenario [69], suppose that $\mathbf{e}$ = {CommandRelay = cmdClose, SensorCurrent = readCurrentLo, SensorVoltage = readVoltageHi, SensorTouchSensor = readClosed}. This gives $\arg\max P(\mathbf{H} \mid \mathbf{e})$= {HealthBattery = healthy, HealthLoad = healthy, HealthCurrent = stuckCurrentLo, HealthVoltage = healthy}. In this scenario, given the evidence of the command and sensor readings, the current flows from the battery to the load as both of them are healthy, also the voltage sensor is healthy. So the defective component is the current sensor which reads low instead of high.

Using our NetEyes software, the analyst interacts with the graph by changing the position of focus. The focus nodes (in this case, the health nodes) that have been zoomed, will use a larger font for labels than their neighboring nodes as shown in Figure 5.7. It is now possible to read the node labels for the focus nodes and the nodes close to them. In general, multi-focus selection is used to make the labels readable and for comparing various nodes to explore their differences and similarities. The side panel in Figure 5.7 shows detailed information about the nodes, specifically the conditional probability tables, for further comparison and analysis [16]. The bubbles help the user to trace a detail window to its corresponding node in the network view.

5.6.2 Analytical Tasks for Bayesian Networks

Several problem solving tasks can be performed with NetEyes for a Bayesian network; we now consider the validation and the diagnosis tasks.

Validation Task

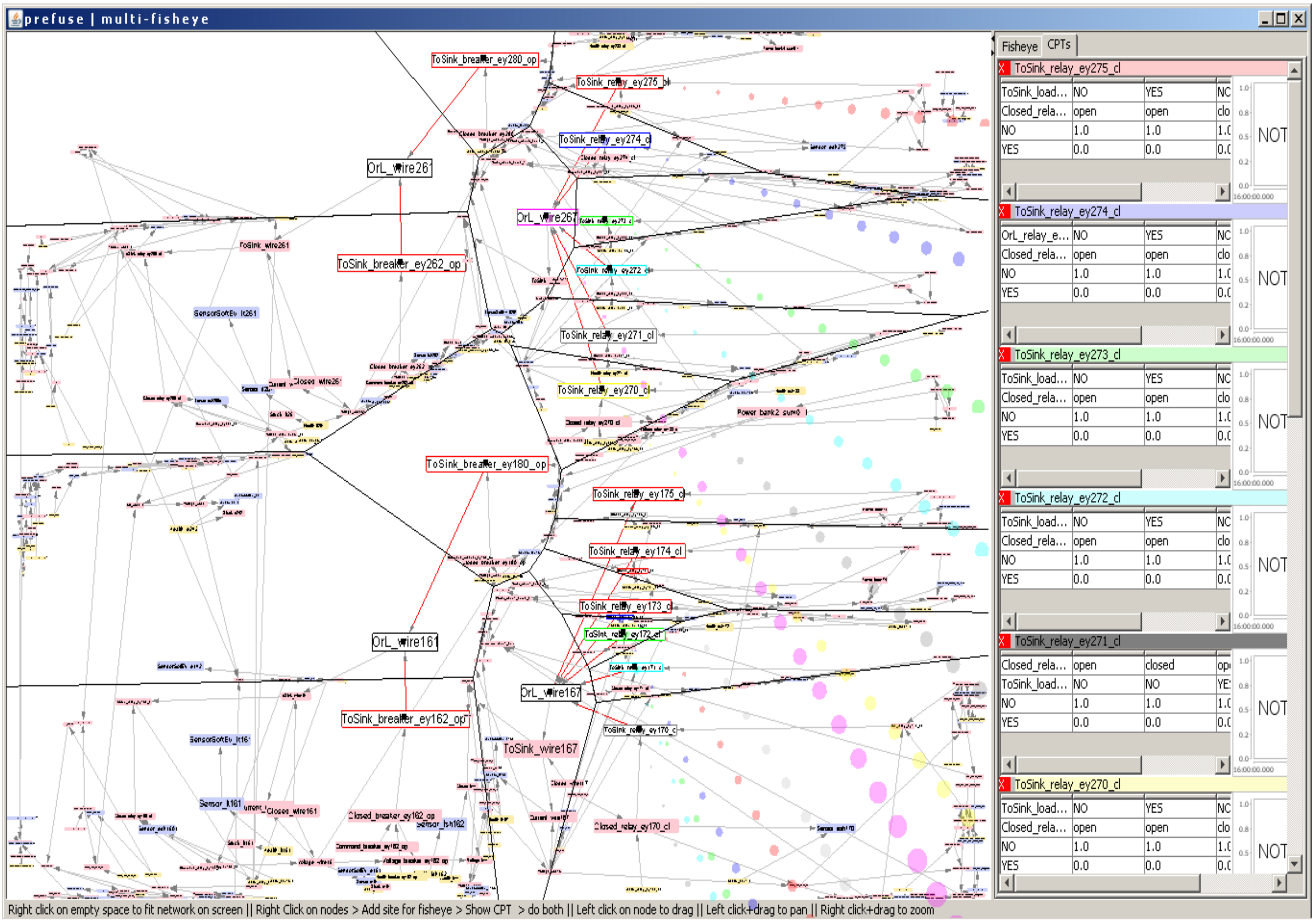
When validating a Bayesian network such as ADAPT, the user may want to compare the CPTs of a set of nodes, for example, health nodes that represent health of different components but with similar conditional probability tables as shown in Figure 5.7. Thus, we investigate $P(H_i|pa(H_i))$ where $H_i \in \mathbf{H}$ is a Bayesian network health node and $pa(H_i)$ denote the parent nodes of H_i.

There can be multiple nodes that may be the major causal nodes for certain hidden or observable effects. These nodes can lie quite far-apart in a large Bayesian network as shown in Figure 5.8. Using our multi-focus and multi-window technique, they can be zoomed to analyze their CPTs.

Diagnosis Task

The diagnosis task investigates $P(H_i|\mathbf{e})$ where H_i is a Bayesian network health node and $\mathbf{e}$ is the evidence. Here, multi-focus can help a Bayesian analyst by allowing him or her to focus, at the same time, on multiple H_i's with interesting posteriors $P(H_i|\mathbf{e})$. For example, it might be that multiple nodes have high posterior probabilities of being defective in a diagnostic Bayesian network such as ADAPT. If the nodes are distant and their labels are hard to read in the original network layout (as they can easily be in ADAPT and other Bayesian networks), our multi-focus technique provides valuable support for interactive analysis. Figure 5.9 [2] shows the usage of the

[2]http://www.youtube.com/watch?v=oJh1kbQVcXc



ToSink_relay_ey275_cl

ToSink_load...	NO	YES	NC
Closed_rela...	open	open	clo
NO	1.0	1.0	1.(
YES	0.0	0.0	0.(

ToSink_relay_ey274_cl

OrL_relay_e...	NO	YES	NC
Closed_rela...	open	open	clo
NO	1.0	1.0	1.(
YES	0.0	0.0	0.(

ToSink_relay_ey273_cl

ToSink_load...	NO	YES	NC
Closed_rela...	open	open	clo
NO	1.0	1.0	1.(
YES	0.0	0.0	0.(

ToSink_relay_ey272_cl

ToSink_load...	NO	YES	NC
Closed_rela...	open	open	clo
NO	1.0	1.0	1.(
YES	0.0	0.0	0.(

ToSink_relay_ey271_cl

Closed_rela...	open	closed	op
ToSink_load...	NO	NO	YE
NO	1.0	1.0	1.(
YES	0.0	0.0	0.(

ToSink_relay_ey270_cl

ToSink_load...	NO	YES	NC
Closed_rela...	open	open	clo
NO	1.0	1.0	1.(
YES	0.0	0.0	0.(

Right click on empty space to fit network on screen || Right Click on nodes > Add site for fisheye > Show CPT > do both || Left click on node to drag || Left click+drag to pan || Right click+drag to zoom

Figure 5.8: The zoomed parent nodes of the *Orl_wire* nodes and their CPTs in the ADAPT electrical Bayesian network. These zoomed node labels are readable on the computer screen even though they are difficult to read in this screenshot.

multi-focus technique in detecting faulty components in an electrical power system.

Evaluation

Our evaluation approach has been to explore complex Bayesian networks while taking note of how the tool aided an analyst in finding and remembering nodes of value during a problem solving session. Specifically, we compare and contrast NetEyes with other Bayesian network visualization tools like Hugin [1], Netica [77] and GeNIe/SMILE [21]. Many features and algorithms were explored, including alternative distortion approaches under rectangular and Voronoi polygonal partitioning. We ended up using user controllable distortion and Voronoi polygonal partitioning. Simplifying the controls and amplifying the mechanisms for remembering where one is in the network exploration process, were helpful for the user to make sense of the network.

The interface is a dramatic simplification over pop-up style controls, and helps focusing action on the essential nodes of a network. The interface felt agile and powerful to our Bayesian network expert as he was able to reformulate network questions several times a minute. He commented on and enjoyed discovering six mechanisms to orient, annotate and understand the relations between nodes. (1) Partitioning allowed him to quarantine (he used the word ”sacred”) parts of the network that contained interesting nodes. (2) The fisheye, he said, allowed him to highlight and remember which nodes he deemed important in a very visible way. (3) The bubbles gave him easy to follow indications of where important nodes were. (4) The motion of panning made the bubble lines show how distant the nodes were separate from other mechanisms. (5) Panning motions and mouse-over helped resolve nodes that were overlapping. (6) The use of node coloring helped to focus on nodes of similar types.

The interesting nodes in the network were found by using the search techniques as discussed in Section 5.5.1. For Bayesian networks, in reviewing the conditional probability tables associated with the focus nodes, our network expert found himself using a collect-review-dispense loop to home in on the conditional probability tables that needed to be compared. Often he would collect 10-20 of these tables and then prune down to 4-5 in one iteration of this collect-review-dispense loop. He described the activity as a network review, similar to code review in software engineering, as he poked around hunting and foraging with the support of the system's many memory aids. In particular, the tool helped in identifying important nodes for further analysis and comparison in the side panel.

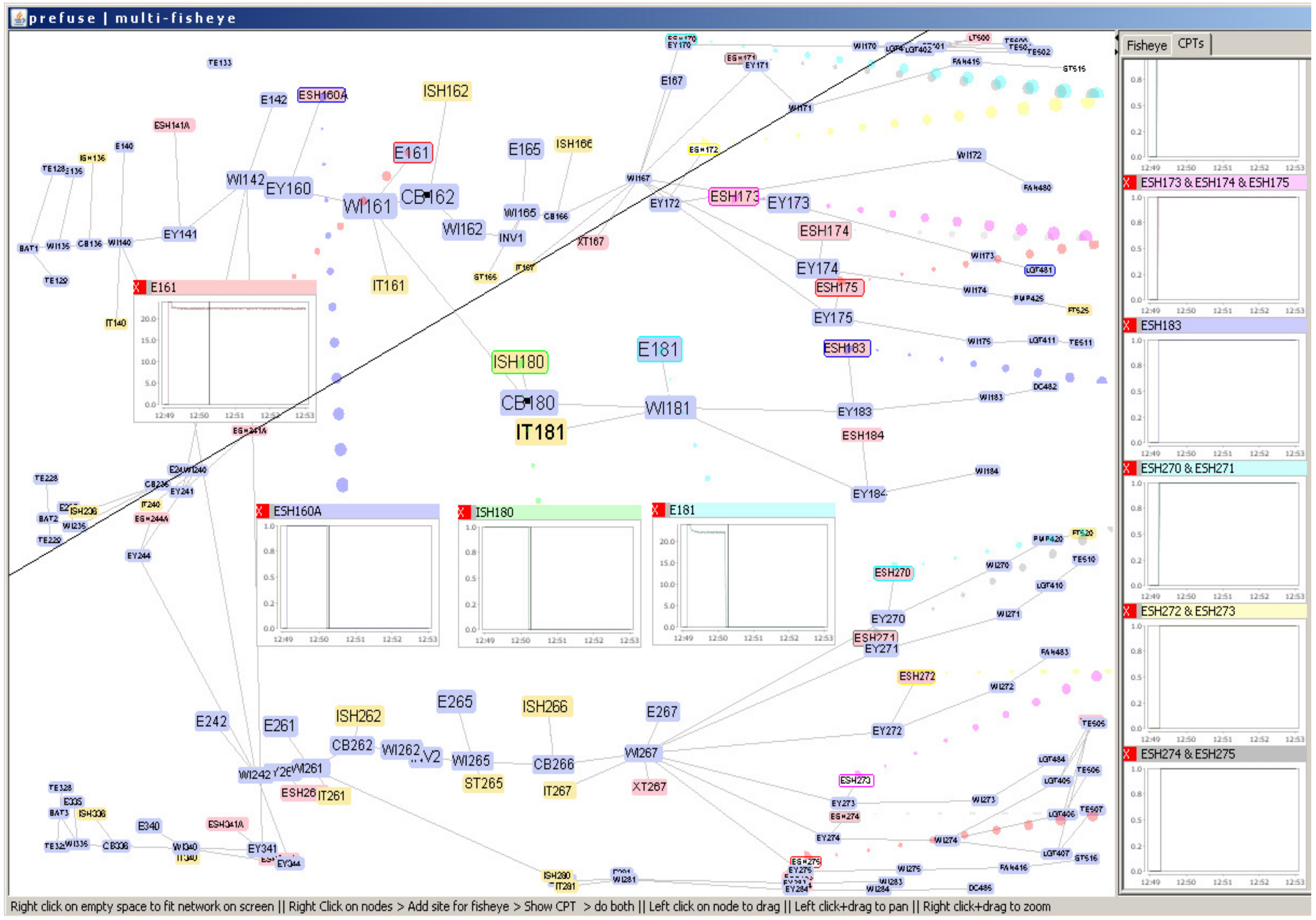
5.6.3 EM Learning

Our multi-focus technique can be used to study the progress of EM runs during traditional EM learning. The Alarm Bayesian network, as an example, is visualized in the network window (see Figure 5.11). Nodes can be zoomed and their CPTs can be studied (see Figure 5.12). CPTs are visualized as time series graphs showing EM iterations on the x-axis versus probability values on the y-axis. Each row in the detail window represents a random variable and each column represents an EM run. Bubbles connect the nodes in the network view with the time series graphs in the focus window (as shown on the right).

We can see from Figure 5.12, for the top two CPTs (INSUF and CO), the probability values remain the same throughout the iterations (*i.e.,* these random variables converge much earlier during EM learning). The bottom CPT (PRESS) shows more variations during EM learning. For a given training dataset, this behavior seems to be the same for all EM runs. From the network view, we can see that the top CPT (INSUF) belongs to a parent variable and the bottom two CPTs (CO and PRESS) belong to child variables. This analysis helps to identify the random variables that can have significant impact on the solution quality.

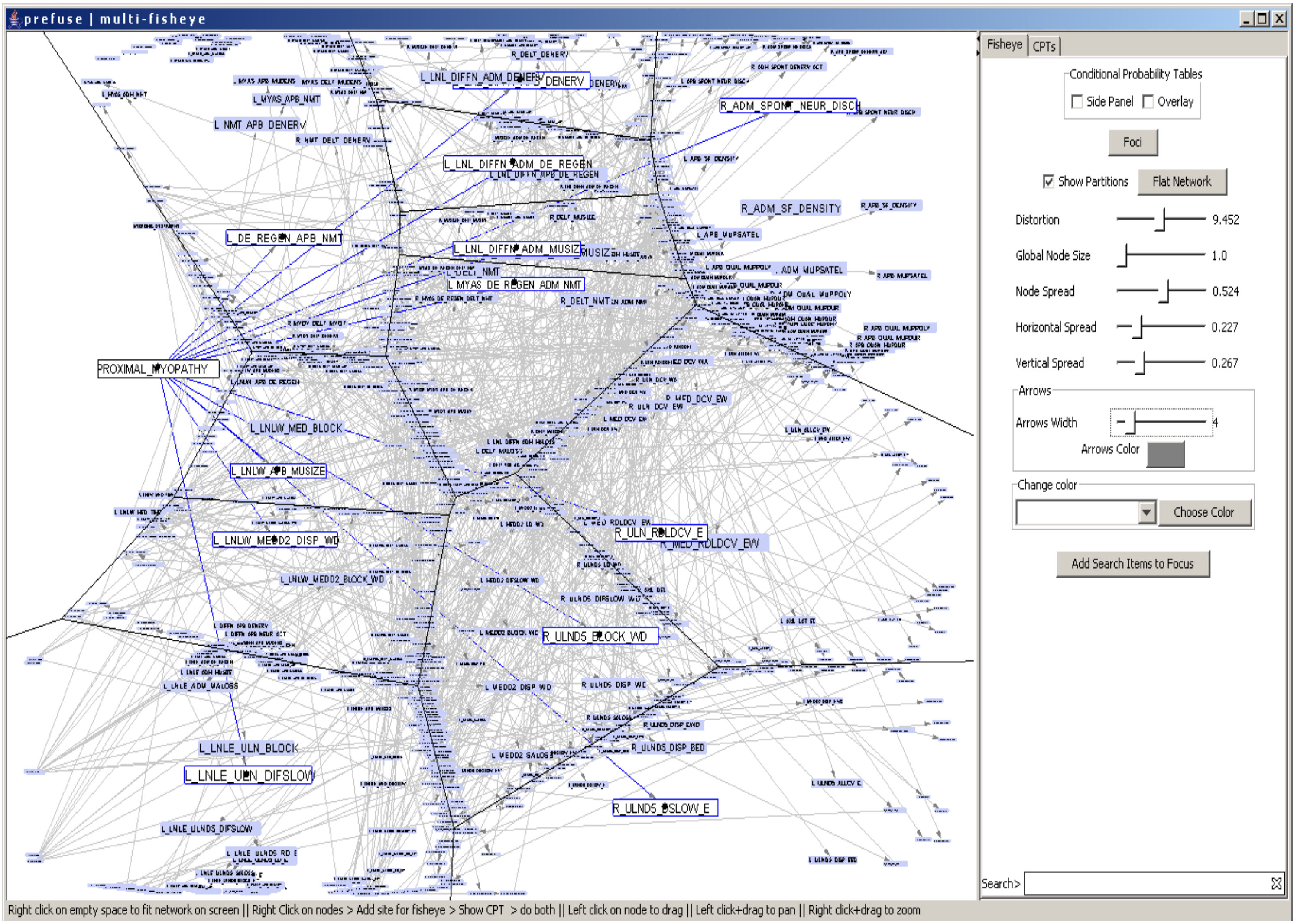
In Figure 5.13, we show the progress of probability values in the CPT (detailed window) and the solution quality (window on the left) for traditional EM. The black trace line (as shown in Figure 5.14) is used to study the progress of an EM run at a particular EM iteration. For example, the trace line is shown for an EM run (alarm 3 shown in blue color) at EM iteration 30. We can notice that the EM run (alarm 3) shows significant changes in the CPTs values (right window) and there is an increase in solution quality (left window) at EM iteration 30.

For GAEM learning, Figure 5.14 shows the progress of probability values in the CPT (detailed window), the solution quality and iterations (on the left side window) for each generation. The time series graphs (generations on x-axis versus probability values on y-axis), is used to show the progress of probability values for each generation of GAEM. Each row in the detailed window represents a random variable in the Alarm Bayesian network and each column represents one EM run. The black trace line (as shown in Figure 5.14) is used to study the progress of EM run at a particular generation in GAEM. The changes in solution quality (measured in terms



Right click on empty space to fit network on screen || Right Click on nodes > Add site for fisheye > Show CPT > do both || Left click on node to drag || Left click+drag to pan || Right click+drag to zoom

Figure 5.9: Time-series graphs of the zoomed sensor nodes in the ADAPT electrical power network. The node labels and their time-series graphs are readable for further analysis. The time-series graphs around the node CB180 (light blue) show a drop in their reading, suggesting that the component CB180 is faulty.



Conditional Probability Tables

☐ Side Panel ☐ Overlay

☑ Show Partitions

Distortion 9.452

Global Node Size 1.0

Node Spread 0.524

Horizontal Spread 0.227

Vertical Spread 0.267

Arrows

Arrows Width 4

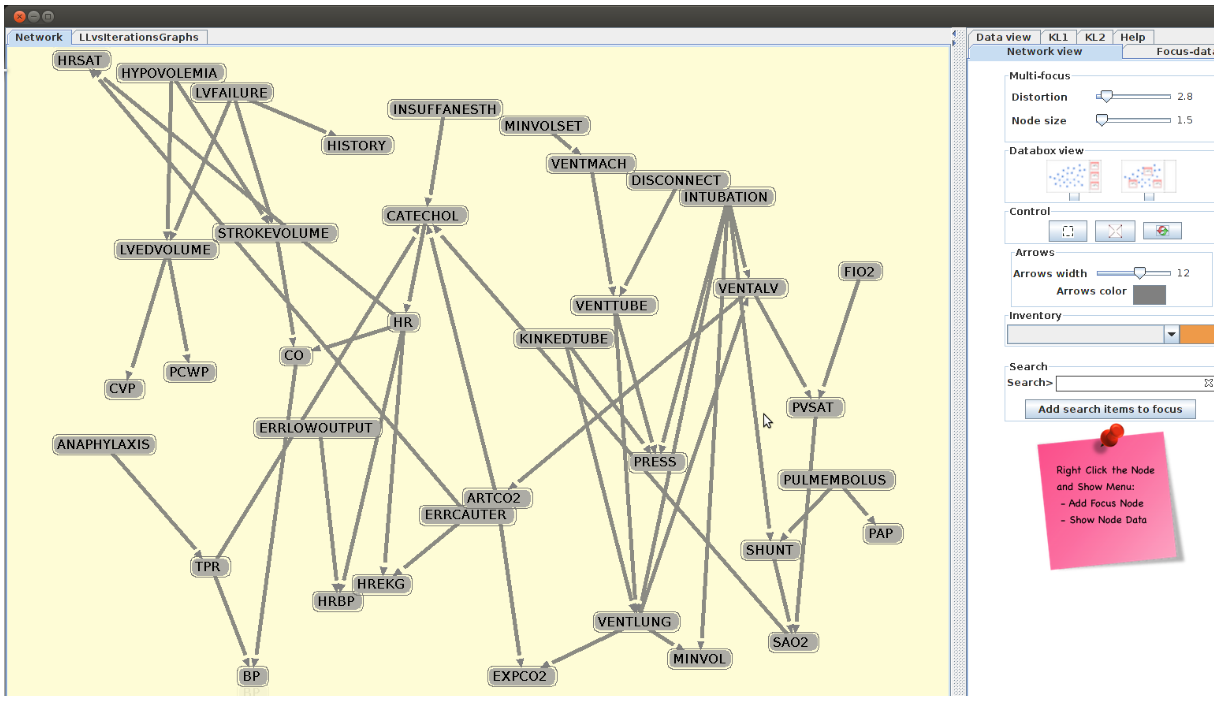
Arrows Color

Change color

Search>

Right click on empty space to fit network on screen || Right Click on nodes > Add site for fisheye > Show CPT > do both || Left click on node to drag || Left click+drag to pan || Right click+drag to zoom

Figure 5.10: The large Munin2 Bayesian network and application of multi-focus on the children nodes (*L_LNLE_ULN_DIFLOW*, *L_LNLW_MED2_DISP_WO* etc.) for a neural disorder disease node called *Proximal Myopathy* (the left most focus node). After the application of multi-focus zooming, the children and the disease nodes are clearly readable on the computer screen even though they are difficult to read in this screenshot.



Multi-focus
Distortion 2.8
Node size 1.5
Arrows
Arrows width 12
Arrows color
Search
Search>

Right Click the Node and Show Menu:
- Add Focus Node
- Show Node Data

Figure 5.11: Alarm BN is shown.

of log likelihood (LL) value) and the number of iterations for each generation is shown on the left window. For example, in Figure 5.14, EM run 3 (alarm 3 shown in gray color), shows an increase in LL from generation 4 to 5 and change in CPT values. Traceline at generation 5 is used to compare the LL values (window on the left) and CPT values (window on the right) for EM run 3. This analysis is useful to understand the impact of random perturbations generated using genetic operators (mutation, crossover and replacement) at each generation of GAEM. GAEM was run with a mutation probability of $p_m = 0.9$ in Figure 5.14 and $p_m = 0.05$ in Figure 5.15. On comparing these two figure, we can see that with high mutation probability, GAEM needs more number of iterations to converge whereas with low mutation probability, few iterations are sufficient to converge.

5.7 Discussion

Distortion such as fisheye views can increase the ability to keep context visible. Multi-focus approaches improve visual analysis by allowing comparison of different parts of a network, providing analysts with a collect-review-dispense analytical capability. The NetEyes generates dis-

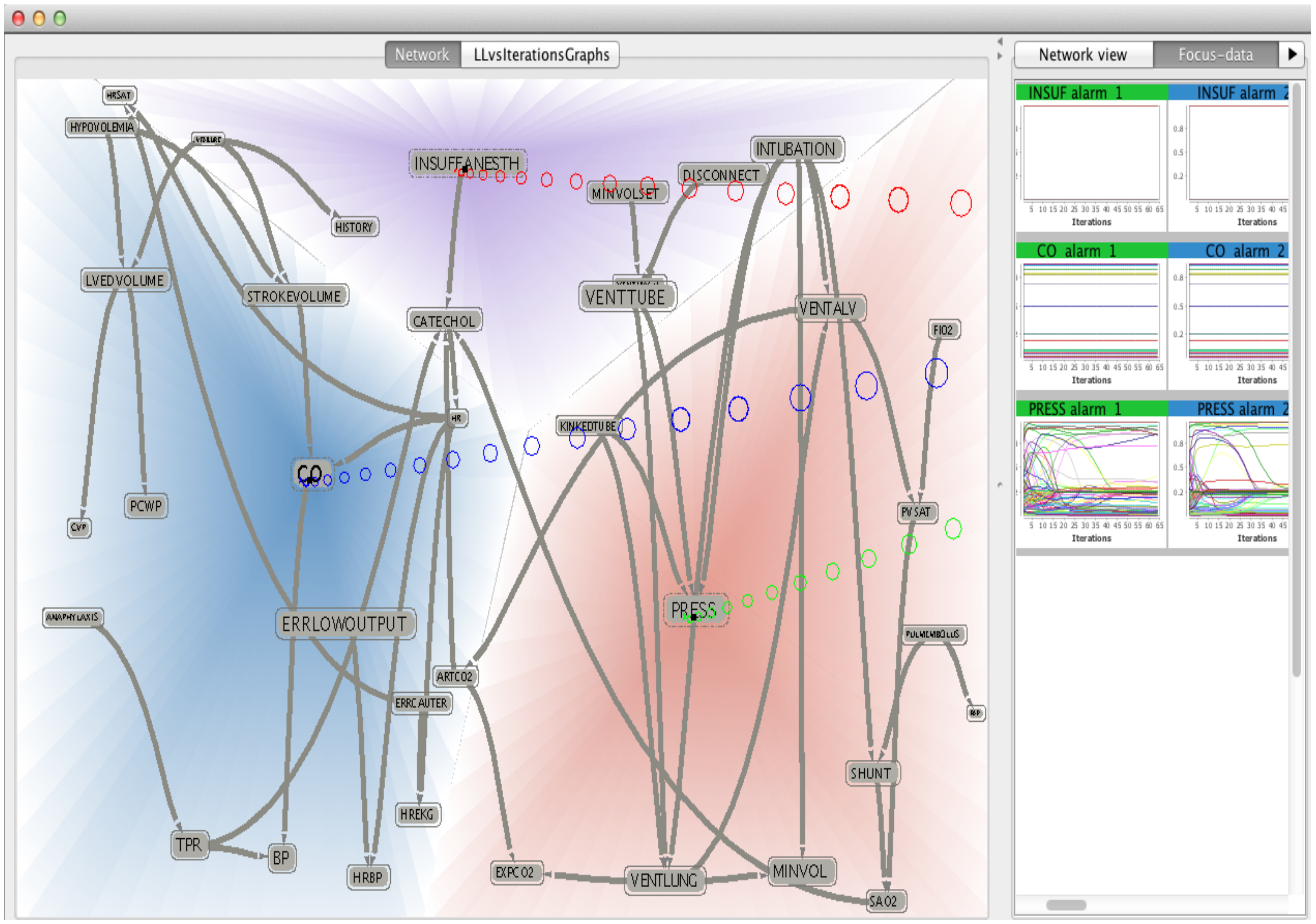


Figure 5.12: In alarm BN, 3 nodes are selected and zoomed. Progress of CPT values for 65 iterations during traditional EM learning are shown as time-series graphs on the right. Bubbles connect the nodes in the network view to the CPTs on the right.

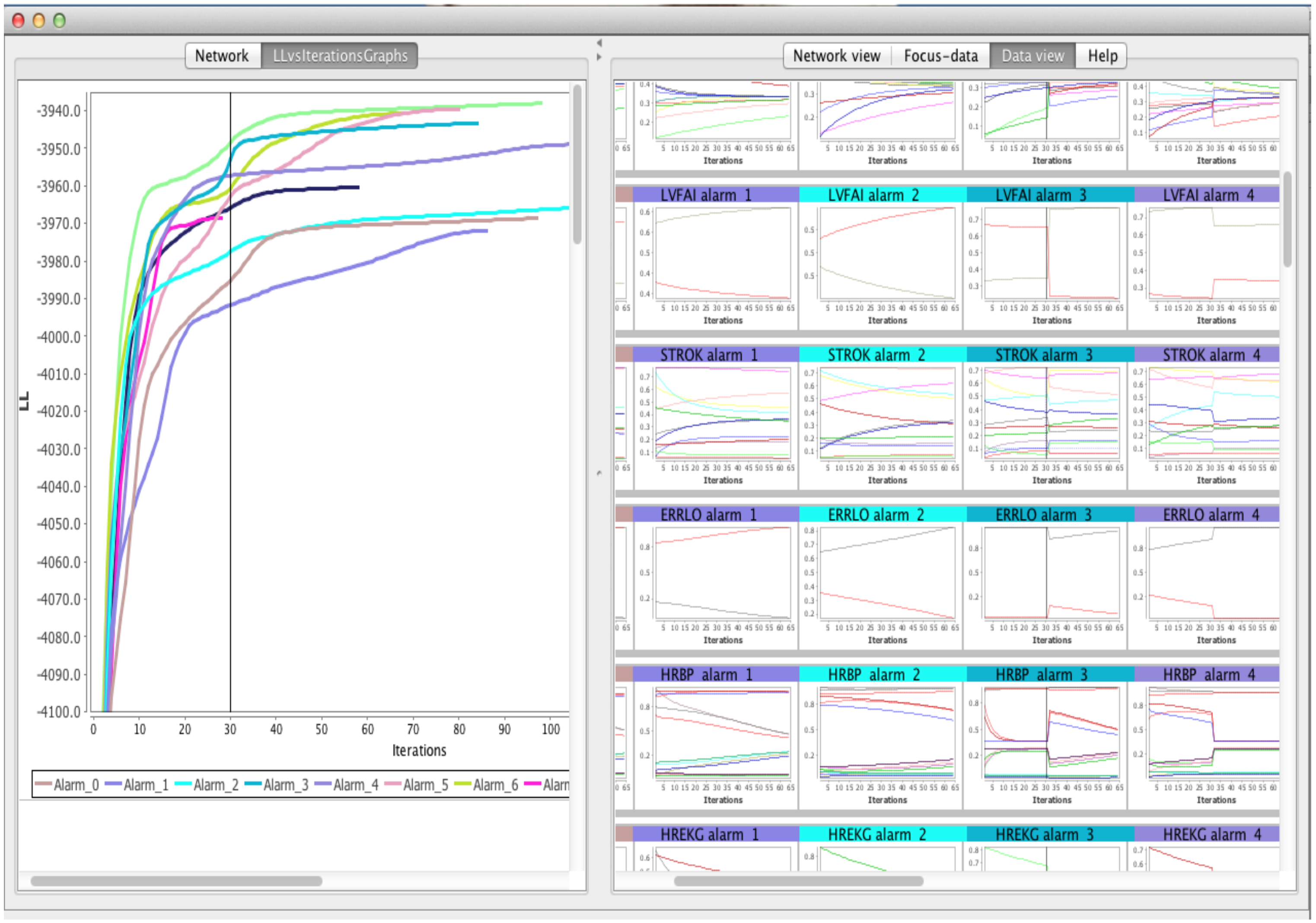


Figure 5.13: Results of traditional EM learning for 10 EM runs is shown. Progress of log likelihood (LL) values for each iteration is shown on the left. Changes in CPT values for each iteration is shown on the right. For EM run 3 (alarm 3), traceline is used to compare LL values (window on the left) and CPT values (window on the right) at EM iteration 30.

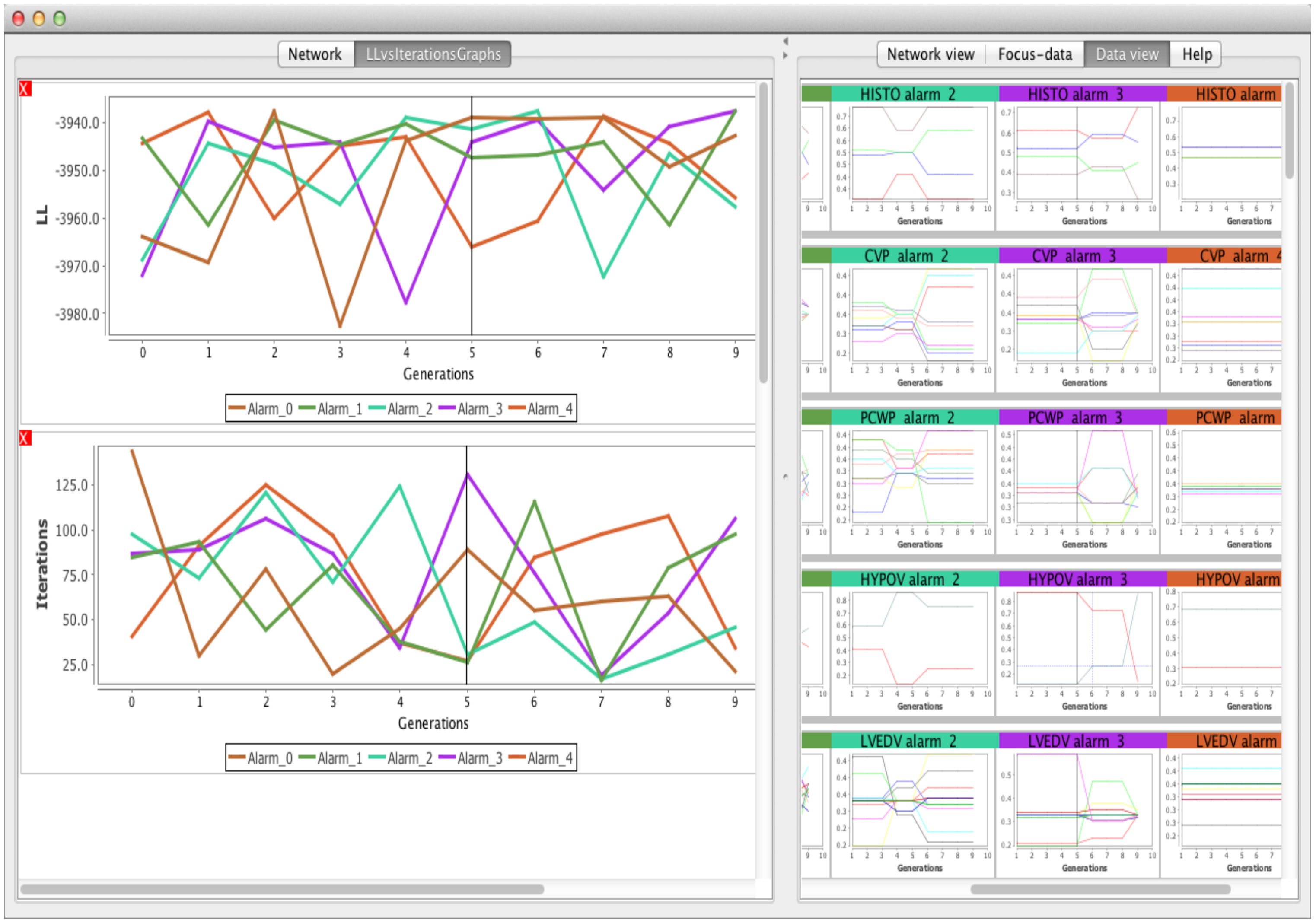


Figure 5.14: Results of GAEM learning for 5 EM runs is shown. Change in log likelihood (LL) values and iterations for each generation is shown on the left. Changes in CPT values for each generation is shown on the right. For EM run 3 (alarm 3), there is an increase in LL from generation 4 to 5. Traceline at generation 5 is used to compare the change in LL values (window on the left) and the change in CPT values (window on the right) for EM run 3. The mutation probability is $p_m = 0.9$ and crossover probability is $p_c = 0.5$. Due to a high mutation probability, more iterations are needed for EM to converge.

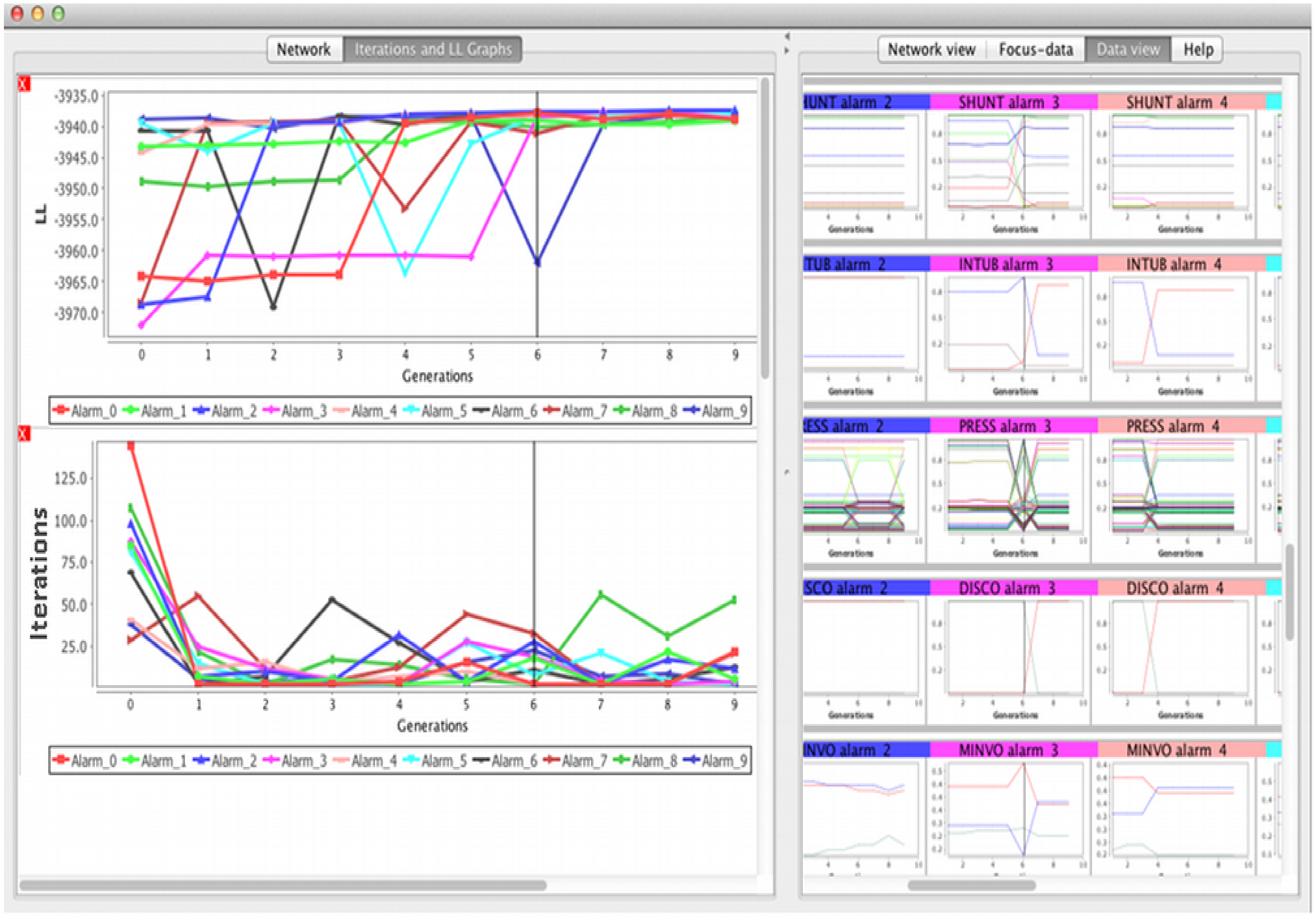


Figure 5.15: Results of GAEM learning for 5 EM runs is shown. Change in log likelihood (LL) values and iterations for each generation is shown on the left. Changes in CPT values for each generation is shown on the right.The mutation probability is $p_m = 0.05$ and crossover probability is $p_c = 0.5$. Due to a low mutation probability, only few iterations are needed for EM to converge.

tortion boundaries which can reduce global distortion, reducing visual comparison challenges. In particular, our technique allows an analyst to interactively bring important parts of a network 'forward' by selectively zooming in, to be compared both structurally in the network and in a multi-window semantic display [16]. Our system can be used to create over a dozen focus partitions.

This system gives simultaneous multi-focus and multi-window zooming capability that enables improved interactive visual exploration of Bayesian networks. In case of a failure in an electrical circuit, the user may want to find the faulty component. For this, a deeper analysis of each of the component is required. Using this multi-focus technique, similar components like voltage or current sensors can be zoomed. Using the multi-window technique, their internal readings can be studied. The multiple windows which contain detailed data can be floating or aggregated in a side panel. The side panel is designed to align and compare internal readings of multiple selected nodes. If there is any sudden drop or rise in the readings in any of the components, then they can be diagnosed further.During traditional EM learning, the progress of solution quality and changes in CPT values for EM iteration can be studied in detail. We found that the CPTs some of the nodes converged much earlier during EM learning. In the case of GAEM learning, the user can study the progress of the probability values, solution quality, and iterations during each generation of GA. The analysis can help the user to better understand the impact of genetic operators during EM learning.

The multi-focus with the multi-window views shown in this chapter promises to improve network analysis. We have used our technique in electrical networks (Figure 5.9), Bayesian networks (Figure 5.10) and even on social networks. We showed how the network view, the CPTs (shown in detail window) and the solution quality graphs can be useful for understanding the traditional EM and GAEM. This chapter pushes for adding techniques to the arsenal of ways to allow users to better view and analyze large networks. Analytics and reasoning are being done on increasingly complex datasets. This chapter demonstrates improvements towards and calls for future work on systems that integrates scalable user interactivity into comparing parts and internal semantics of large-scale networks.

Chapter 6

Conclusion and Future Work

The EM algorithm is a popular method for parameter estimation in Bayesian networks in the presence of incomplete data. In real world scenarios, incomplete data can arise due to security constraints or system fault. The algorithm has several limitations, such as the possibility of getting stuck in local optima, slow convergence, lack of understanding of the progress of CPT values during learning, and computational expense. The application of stochastic and visualization techniques provides a possible way to solve the local optima and slow convergence problems and improve the understanding of EM behavior.

6.1 Summary of Contributions

In this thesis, we have integrated genetic algorithm and age-layered methods for the problem of parameter estimation when data is incomplete. We proposed two novel techniques, the GAEM (Genetic Algorithm based Expectation Maximization) and ALEM (Age Layered Expectation Maximization) algorithms for solving the local optima and slow convergence problems in EM. We propose an intuitive user interface integrated with multi-focus technique (NetEyes) for aiding in the analysis of the progress of EM runs during BN learning.

6.1.1 Genetic Algorithm for Expectation Maximization (GAEM)

A hybrid genetic algorithm for expectation maximization is proposed by combining the local search property of EM with the global search property of GA. With the help of the genetic operators, GA helps to widen the search. On the other hand, EM helps to orient the search and does a hill climbing. We found that a small initial population such as 2 or 4 individuals were sufficient to produce superior quality solutions compared to the traditional EM. This is because the GAEM technique uses the already converged EM individuals as parents for the next generation. The random perturbations introduced by GA helps to improve these converged EM individuals further during the generations and reach better solution quality. For our experiments, we split the Bayesian networks in to hard and easy search spaces based on their distribution of log likelihood values. We experimented on two difficult Bayesian networks, the Alarm BN representing the hard search space and the Carstarts BN representing the easy search space, for varying sample sizes and hidden variable configurations. GAEM was able to produce speed up in all cases.

6.1.2 Age-Layered Expectation Maximization (ALEM)

We proposed an age layered paradigm for speeding up EM algorithm. In this technique, ‘age’ is defined as the number of iterations an EM individual has undergone. Each layer has the age limit or the maximum number of iterations for a layer. Whenever an EM individual reaches a maximum iteration for a layer, it is moved to the layer above it by doing a log likelihood comparison check. Thus, we group similarly aged individuals in layers. The key idea is that log likelihood comparisons between similarly aged EM individuals are a reliable indicator for comparisons at convergence. The EM individual that fail this check will be culled. Thus, we save the computations cost that could have been wasted on this poorly performing EM individual. We assume that the probability of a poorly performing EM individual during initial iterations, becoming a high quality EM individual at later iterations is low. Each layer has the minimum runs parameter, which is the number of EM individuals, a layer can accommodate without any comparison. This parameter can be viewed as a tradeoff between computation efficiency and solution quality. If the minimum runs parameter is a larger value, more EM individuals can be inserted without the comparison check which means we do not get a chance to cull. We show that ALEM can significantly decrease the average number of iterations on four difficult Bayesian networks but at the same time achieve a higher solution quality, or gets very close, in all instances.

6.1.3 Network Visualization with Multi-focus Technique (NetEyes)

We present Neteyes, an interactive software, where the user can compare and analyze a set of nodes simultaneously. The user interface is integrated with a multi-focus technique that allows a user to select nodes from different parts of the network and zoom in on them. The selected node's details can be studied in the focus window. Neteyes software is designed to meet carefully selected set of design goals. The software also provides a detailed window wherein a thorough analysis regarding all nodes details can be done. For example, CPTs of all the nodes can be viewed as time series graphs to show the change in probability values during EM learning.

6.2 Future Work

In this thesis, we investigate genetic algorithm based techniques to mitigate the slow convergence and local maxima problem of the EM algorithm. We focused on the problem of parameter estimation for Bayesian networks in the presence of incomplete data. But the above strategies can potentially be applied to any problems that involve EM learning.

As future work, the EM algorithm can be integrated with other population-based approaches such as evolutionary strategies (ES) or differential evolution (DE) [22]. The performance of GAEM can then be compared with other evolutionary techniques.

A higher speed-up can probably be achieved by deploying GAEM in a GPU environment. The configuration parameters of GAEM can be auto-tuned based on the progress of the EM individuals.

The ALEM technique can be enhanced by integrating it with robust methods to tune the minimum runs parameter, as well as error bounds on using ALEM versus traditional EM. We are also intrigued by the relationship between the nature of the stopping criterion used and the convergence of EM, and more exploration can be done in this area.

NetEye software can be used in visualizing any networks such as social networks and computer networks. The software currently supports visualization for hundreds of nodes which can be improved with the help of powerful image rendering techniques. The multi-focus zooming algorithm can also be improved to support faster zooming actions on large collections of nodes. Novel visualization techniques can perhaps be integrated more closely with the machine learning operations in order to even better understand the behavior of EM algorithm.

Bibliography

[1] Stig K. Andersen, Kristian G. Olesen, Finn Verner Jensen, and Frank Jensen. HUGIN - A Shell for Building Bayesian Belief Universes for Expert Systems. In *Proc. of IJCAI'89*, pages 1080–1085, Detroit, MI, USA, Aug 1989. 5.1, 5.6.2

[2] Franz Aurenhammer. Voronoi diagrams a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, September 1991. 5.5, 5.5.2

[3] Michael Balzer, Oliver Deussen, and Claus Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the 2005 ACM symposium on Software visualization*, SoftVis '05, New York, NY, USA, 2005. ACM. 5.5

[4] Lyn Bartram, Albert Ho, John Dill, and Frank Henigman. The continuous zoom: a constrained fisheye technique for viewing and navigating large information spaces. In *Proc ACM symposium on User interface and software technology*, UIST '95, New York, NY, USA, 1995. ACM. 5.4.1

[5] Aniruddha Basak, Irina Brinster, Xianheng Ma, and Ole J Mengshoel. Accelerating bayesian network parameter learning using hadoop and mapreduce. In *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pages 101–108. ACM, 2012. 3.2.1

[6] Aniruddha Basak, Irina Brinster, and Ole J Mengshoel. Mapreduce for Bayesian network parameter learning using the EM algorithm. *Proc. of Big Learning: Algorithms, Systems and Tools*, 2012. 3.2.1

[7] V. Batagelj, W. Didimo, G. Liotta, P. Palladino, and M. Patrignani. Visual analysis of large graphs using (x,y)-clustering and hybrid visualizations. In *Pacific Visualization Symposium (PacificVis), 2010 IEEE*, march 2010. 5.4.2

[8] E. Bauer, D. Koller, and Y. Singer. Update rules for parameter estimation in Bayesian networks. In *Proc. Thirteenth Annual Conference on Uncertainty in AI (UAI)*, pages 3–13, 1997. 4.1

[9] I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. Technical report, Knowledge Systems, AI Laboratory, 1989. 3.5.1, 4.4.1

[10] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. Extending distortion viewing from 2d to 3d. *IEEE Comput. Graph. Appl.*, 17(4):42–51, July 1997. 5.2

[11] M. Sheelagh T. Carpendale, M. Sheelagh, T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. 3-dimensional pliable surfaces: For the effective presentation of visual information. In *In Proc. of UIST'95*. ACM, 1995. 5.4.3

[12] Chad Carson, Serge Belongie, Hayit Greenspan, and Jitendra Malik. Blobworld: Image segmentation using expectation-maximization and its application to image querying. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:1026–1038, 1999. 4.3.1

[13] D. M. Chickering. Learning equivalence classes of Bayesian Network structures. *Journal of Machine Learning Research*, 2, 2002. 3.2.1

[14] Andy Cockburn, Amy Karlson, and Benjamin B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv.*, 41(1):2:1–2:31, January 2009. 5.4.1

[15] G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992. 3.2.1

[16] M. Cossalter, O. J. Mengshoel, and T. Selker. Visualizing and understanding large-scale Bayesian networks. In *The AAAI-11 Workshop on Scalable Integration of Analytics and Visualization*, pages 12–21, 2011. 5.5, 5.6.1, 5.7

[17] B. Delyon, M. Lavielle, and E. Moulines. Convergence of a stochastic approximation version of the EM algorithm. *Annals of Statistics*, 27(1):94–128, 1999. 1.3, 3.2.1, 3.2.2

[18] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete

data via the EM algorithm. *Journal of the Royal Statistical Society. Series B*, 39(1):1–38, 1977. 1.2, 2.3.1, 2.3.2, 4.1

[19] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977. 2.3

[20] C. B. Do and S. Batzoglou. What is the expectation maximization algorithm? *Nature Biotech.*, 26:897–899, 2008. 2.3.2

[21] Marek J. Druzdzel. SMILE: Structural Modeling, Inference, and Learning Engine and GeNIe: A Development Environment for Graphical Decision-Theoretic Models. In *Proc. of AAAI'99*, pages 902–903, Orlando, FL, USA, Jul 1999. 5.1, 5.6.2

[22] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*, volume 53. Springer, 2003. 6.2

[23] Tarek A El-Mihoub, Adrian A Hopgood, Lars Nolle, and Alan Battersby. Hybrid genetic algorithms: A review. *Engineering Letters*, 13(2):124–137, 2006. 2.4.1

[24] G. Elidan, M. Ninio, N. Friedman, and D. Schuurmans. Data perturbation for escaping local maxima in learning. In *Eighteenth National Conference on Artificial Intelligence*, pages 132–139, 2002. 3.2.2, 4.2

[25] T. Elmihoub, A. A. Hopgood, L. Nolle, and A. Battersby. Performance of hybrid genetic algorithms incorporating local search. In *Proc. of 18th European Simulation Multiconference*, pages 154–160, 2004. 3.5.3

[26] N. Elmqvist, Y. Riche, N. Henry-Riche, and J.-D. Fekete. Melange: Space folding for visual exploration. *Visualization and Computer Graphics, IEEE Transactions on*, 16(3):468 –483, may-june 2010. 5.4.3

[27] F. P. Espinoza, B. S. Minsker, and D. E. Goldberg. Performance evaluation and population reduction for a self adaptive hybrid genetic algorithm (sahga). In *Proc. of the 2003 International Conference on Genetic and Evolutionary Computation: PartI*, GECCO'03, pages 922–933, 2003. 3.5.3

[28] Ronald Aylmer Fisher. On the mathematical foundations of theoretical statistics. *Philo-*

sophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character, 222:309–368, 1922. 1.1

[29] Arno Formella and Jorg Keller. Generalized fisheye views of graphs. In *Proceedings of the Symposium on Graph Drawing*, GD '95, pages 242–253, London, UK, 1996. Springer-Verlag. 5.4.1

[30] S Fortune. A sweepline algorithm for voronoi diagrams. In *Proceedings of the second annual symposium on Computational geometry*, SCG '86, New York, NY, USA, 1986. ACM. 5.3

[31] G. W. Furnas. Generalized fisheye views. *SIGCHI*, 17, April 1986. 5.1, 5.4.1, 5.5.1

[32] George W. Furnas. A fisheye follow-up: further reflections on focus + context. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 999–1008, New York, NY, USA, 2006. ACM. 5.5.1

[33] Emden R. Gansner, Yehuda Koren, and Stephen C. North. Topological fisheye views for visualizing large graphs. *IEEE Transactions on Visualization and Computer Graphics*, 11:457–468, July 2005. 5.4.1

[34] S. Ghani, N.H. Riche, and N. Elmqvist. Dynamic insets for context-aware graph navigation. In *Computer Graphics Forum*, volume 30, pages 861–870. Wiley Online Library, 2011. 5.4.1

[35] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. (document), 2.4

[36] D. Heckerman, J. S. Breese, and K. Rommelse. Decision-theoretic troubleshooting. *Communications of the ACM*, 38(3):49–57, 1995. 3.5.1, 4.4.1

[37] Jeffrey Heer, Stuart K. Card, and James A. Landay. Prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '05, New York, NY, USA, 2005. ACM. 5.6

[38] Ivan Herman, Guy Melancon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transcations on Visualization and Computer*

Graphics, 6(1):24–43, 2000. 5.5

[39] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975. 3.2.1

[40] G. S. Hornby. Alps: The age-layered population structure for reducing the problem of premature convergence. In *Proc. of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 815–822, New York, NY, USA, 2006. ACM. 1.4, 4.2

[41] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The lumière project: Bayesian user modeling for inferring the goals and needs of software users. In *Proc. of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 256–265, 1998. 3.5.1, 4.4.1

[42] Edmundo Bonilla Huerta, Batrice Duval, and Jin kao Hao. A hybrid ga/svm approach for gene selection and classification of microarray data. In *EvoWorkshops 2006, LNCS 3907*, pages 34–44. Springer, 2006. 2.4.2

[43] S. H. Jacobson and E. Ycesan. Analyzing the Performance of Generalized Hill Climbing Algorithms. *Journal of Heuristics*, 10:387–405, 2004. 1.3, 4.2

[44] M. Jamshidian and R. I. Jennrich. Acceleration of the EM algorithm by using quasi-Newton methods. *Journal of the Royal Statistical Society. Series B (Methodological)*, 59(3):pp. 569–587, 1997. 3.2.1, 3.2.2

[45] Mortaza Jamshidian and Robert I. Jennrich. Conjugate gradient acceleration of the EM algorithm. *Journal of the American Statistical Association*, 88(421):pp. 221–228, 1993. 1.3

[46] W. Jank. The EM algorithm, its stochastic implementation and global optimization: Some challenges and opportunities for OR. *Perspectives in Operations Research*, pages 367–392, 2006. 1.3, 3.2.1, 3.2.2, 4.2

[47] J.Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988. 2.1, 5.1

[48] Karlis Kaugars, Juris Reinfelds, and Alvis Brazma. A simple algorithm for drawing large

graphs on small screens. In *Proc. of the DIMACS International Workshop on Graph Drawing*, GD '94, London, UK, 1995. Springer-Verlag. 5.4.3

[49] T. Alan Keahey, Dirk Van Gucht, T. Alan Keahey, Nels Gustaf Jerde, and Thomas Elbert Keahey. Nonlinear magnification. Technical report, transformations,Proceedings of the IEEE Symposium on Information Visualization, IEEE Visualization, 1997. 5.4.3

[50] J. F. C. Kingman. *Poisson processes*, volume 3 of *Oxford Studies in Probability*. The Clarendon Press Oxford University Press, New York, 1993. Oxford Science Publications. 4.3.2

[51] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. 2.2

[52] Natalio Krasnogor and James Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005. (document), 2.4

[53] K. Krishna and M. Narasimha Murty. Genetic K-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(3):433–439, Jun 1999. 2.4.2, 3.4

[54] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951. 3.3.3

[55] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proc. SIGCHI Human factors in computing systems*, CHI '95, 1995. 5.4.1

[56] P. Larrañaga, H. Karshenas, C. Bielza, and R. Santana. A review on evolutionary algorithms in Bayesian network learning and inference tasks. *Inf. Sci.*, 233:109–125, June 2013. 3.2.1

[57] P. Larrañaga, C. M. H. Kuijpers, R. H. Murga, and Y. Yurramendi. Learning Bayesian network structures by searching for the best ordering with genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 26(4):487–493, Jul 1996. 3.2.1, 3.2.2

[58] K. B. Laskey and J. W. Myers. Population Markov Chain Monte Carlo. *Machine Learning*, 2003. 3.2.1, 3.2.2

[59] M. Lavielle and E. Moulines. A simulated annealing version of the EM algorithm for non-Gaussian deconvolution. *Statistics and Computing*, 7(4):229–236, 1997. 4.2

[60] T.Y. Lee. Coherent time-varying graph drawing with multifocus+ context interaction. *IEEE Transactions on Visualization and Computer Graphics*, 18(8), 2012. 5.4.1

[61] M. J. Lindstrom and D. M. Bates. Newton-Raphson and EM Algorithms for Linear Mixed-Effects Models for Repeated-Measures Data. *Journal of the American Statistical Association*, 83(404):pp. 1014–1022, 1988. 2

[62] F. G. Lobo and D. E. Goldberg. Decision making in a hybrid genetic algorithm. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 121–125, Apr 1997. 2.4.2

[63] S. W. Mahfoud. Crowding and preselection revisited. In *PPSN 2*, pages 27–36, Amsterdam, 1992. North-Holland. 3.3.3

[64] Geoffrey McLachlan and David Peel. *Finite Mixture Models*. John Wiley & Sons, Inc., 2005. 4.2

[65] X. L. Meng and D. B. Rubin. Maximum likelihood estimation via the ECM algorithm: A general framework. *Biometrika*, 80(2):pp. 267–278, 1993. 1.3, 3.2.1, 3.2.2

[66] Xiao L. Meng and David van Dyk. The EM Algorithm–An Old Folk-Song Sung to a Fast New Tune. *Journal of the Royal Statistical Society. Series B (Methodological)*, 59, 1997. 2.3.2

[67] O. J. Mengshoel and D. E. Goldberg. Probabilistic crowding: Deterministic crowding with probabilisitic replacement. In *GECCO-1999*, volume I, pages 409–416, Orlando, FL, 1999. Morgan Kaufmann Publishers, San Francisco, CA. 3.3.3

[68] O. J. Mengshoel, D. C. Wilkins, and D. Roth. Initialization and restart in stochastic local search: Computing a most probable explanation in Bayesian networks. *IEEE Trans. on Knowledge and Data Engineering*, 23(2):235–247, 2011. 1.3, 4.2

[69] Ole J. Mengshoel, Mark Chavira, Keith Cascio, Scott Poll, Adnan Darwiche, and Serdar

Uckun. Probabilistic Model-Based Diagnosis: An Electrical Power System Case Study. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 40(5):874–885, 2010. 1.1, 5.1, 5.6.1

[70] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. (document), 2.2

[71] Baback Moghaddam, Tony Jebara, and Alex Pentland. Bayesian face recognition. *Pattern Recognition*, 33(11):1771 – 1782, 2000. 1.1

[72] J. M. Mooij. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11:2169–2173, August 2010. 3.5.1, 4.4.1

[73] Robert I. Jennrich Mortaza Jamshidian. Conjugate gradient acceleration of the em algorithm. *Journal of the American Statistical Association*, 88(421):221–228, 1993. 3.3.3

[74] Tamara Munzner, François Guimbretière, Serdar Tasiran, Li Zhang, and Yunhong Zhou. TreeJuxtaposer: scalable tree comparison using Focus+Context with guaranteed visibility. *ACM Trans. Graph.*, 22(3):453–462, July 2003. 5.4.2

[75] J. W. Myers, K. B. Laskey, and K. A. DeJong. Learning Bayesian networks from incomplete data using evolutionary algorithms. GECCO'99, pages 458–465, 1999. 3.2.1, 3.2.2

[76] PK Nanda, D Patra, and A Pradhan. Unsupervised Image Segmentation using Tabu Search and Hidden Markov Random Field Model and Hidden Markov Random Field Model. 2004. 4.2

[77] Netica. by Norsys Software Corp., 1998. 5.6.2

[78] S. K. Ng and G. J. McLachlan. On the choice of the number of blocks with the incremental EM algorithm for the fitting of normal mixtures. *Statistics and Computing*, 13(1):45–55, 2003. 4.2

[79] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using em. *Mach. Learn.*, 39(2-3):103–134, May 2000. 1.2

[80] D. Nikovski. Constructing bayesian networks for medical diagnosis from incomplete and

partially correct statistics. *IEEE Trans. on Knowledge and Data Engineering*, 2000. 1.1

[81] Luis E. O. and Leslie P. K. Accelerating em: An empirical study. In *In Proc. of the fifteenth annual conference on uncertainty in artifical intelligence (UAI)*, pages 512–521. Morgan Kaufmann, 1999. 1.3

[82] Il-Seok Oh, Jin-Seon Lee, and Byung-Ro Moon. Hybrid genetic algorithms for feature selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1424–1437, Nov 2004. 2.4.2

[83] Agnieszka Onisko, Marek J. Druzdzel, and Hanna Wasyluk. A probabilistic causal model for diagnosis of liver disorders. In *In Proc. of the Seventh International Symposium on Intelligent Information Systems (IIS–98*, pages 379–387, 1998. 3.5.1, 4.4.1

[84] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988. 2.1

[85] F. Pernkopf and D. Bouchaffra. Genetic-based EM algorithm for learning Gaussian mixture models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(8):1344–1348, August 2005. 1.3, 3.2.1, 3.2.2

[86] C. Plaisant, J. Grosjean, and B.B. Bederson. Spacetree: supporting exploration in large node link tree, design evolution and empirical evaluation. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, 2002. 5.4.2

[87] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc. of the IEEE*, 77(2):257–286, 1989. 2.3.2, 4.1

[88] E. B. Reed and O. J. Mengshoel. Scaling Bayesian network parameter learning with expectation maximization using MapReduce. *Proc. of Big Learning: Algorithms, Systems and Tools*, 2012. 3.2.1

[89] Erik Reed and Ole J Mengshoel. Bayesian network parameter learning using EM with parameter sharing. In *BMA@ UAI*, pages 48–59. Citeseer, 2014. 4.2

[90] Tobias Reinhard, Silvio Meier, and Martin Glinz. An improved fisheye zoom algorithm for visualizing and editing hierarchical models. In *Workshop on Requirements Engineering Visualization*, REV '07, 2007. 5.2, 5.4.1

[91] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Trans. on Neural Networks*, 1994. 3.4, 3.4

[92] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995. (document), 2, 2.1

[93] Ruslan Salakhutdinov, Sam Roweis, and Zoubin Ghahramani. Optimization with EM and expectation-conjugate-gradient. In *Proceedings, Intl. Conf. on Machine Learning (ICML)*, pages 672–679, 2003. 1.3

[94] A. Saluja, P. Sundararajan, and O. J. Mengshoel. Age-layered expectation maximization for parameter learning in Bayesian networks. *AISTATS-2012*, pages 984–992, 2012. 1.4, 3.2.1, 3.3.3, 3.5.1

[95] Manojit Sarkar and Marc H. Brown. Graphical fisheye views of graphs. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '92, pages 83–91, New York, NY, USA, 1992. ACM. 5.4.1

[96] Manojit Sarkar, Scott S. Snibbe, Oren J. Tversky, and Steven P. Reiss. Stretching the rubber sheet: a metaphor for viewing large layouts on small screens. In *Proceedings of the 6th annual ACM symposium on User interface software and technology*, UIST '93, pages 81–91, New York, NY, USA, 1993. ACM. 5.4.1

[97] Doug Schaffer, Zhengping Zuo, Saul Greenberg, Lyn Bartram, John Dill, Shelli Dubs, and Mark Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Trans. Comput.-Hum. Interact.*, 3, 1996. 5.4.1

[98] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, pages 336–, Washington, DC, USA, 1996. IEEE Computer Society. 5.2

[99] M. A. D. Storey, F. Fracchia, and H. Müller. Customizing a Fisheye View Algorithm to Preserve the Mental Map. *Journal of Visual Languages and Computing*, (10):245–267, 1999. 5.2

[100] Priya Krishnan Sundararajan, Ole Mengshoel, and Ted Selker. Multi-fisheye for interactive visualization of large graphs. In *The AAAI-11 Workshop on Scalable Integration of*

Analytics and Visualization, 2011. 1.4, 5.5, 5.5.2

[101] Priya Krishnan Sundarararajan, Ole J. Mengshoel, and Ted Selker. Multi-focus and multi-window techniques for interactive network exploration. volume 8654, pages 86540–15, 2013. 1.4, 5

[102] B. Thiesson, C. Meek, and D. Heckerman. Accelerating EM for large databases. *Machine Learning*, 45:279–299, 2001. 3.2.1, 3.2.2, 4.2

[103] Bo Thiesson. *Accelerated quantification of Bayesian networks with incomplete data*. University of Aalborg, Institute for Electronic Systems, Department of Mathematics and Computer Science, 1995. 3.3.3

[104] Ying Tu and Han-Wei Shen. Balloon focus: a seamless multi-focus+context method for treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 14:1157–1164, November 2008. 5.4.2

[105] Y. Wang and N.L. Zhang. Severity of local maxima for the EM algorithm: experiences with hierarchical latent class models. In *Proc. of the 3rd Workshop on Probabilistic Graphical Models*, 2006. 4.2

[106] Yi Zhang, Wei Xu, and Jamie Callan. Exact maximum likelihood estimation for word mixtures. In *Text Learning Workshop in International Conference on Machine Learning (ICML)*, 2002. 4.3.1

[107] Z. Zhang, B. T. Dai, and A. K. H. Tung. Estimating local optimums in EM algorithm over Gaussian mixture model. *ICML '08*, pages 1240–1247, 2008. 4.2

[108] Q. Zhao, V. Hautamäki, I. Kärkkäinen, and P. Fränti. Random swap EM algorithm for Gaussian mixture models. *Pattern Recognition Letters*, (16):2120–2126, 2012. 3.2.1, 3.2.2

[109] K. Zhu, H. Song, L. Liu, J. Gao, and G. Cheng. Hybrid genetic algorithm for cloud computing applications. In *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*, pages 182–187, Dec 2011. 2.4.2