8-2008

Empirical Study on Greedy Algorithms for Learning Decomposable Markov Networks

Kyle Jonah Nunez

Empirical Study on Greedy Algorithms for Learning Decomposable Markov Networks

By

Kyle Nunez

Senior Honors Thesis Submitted

in partial fulfillment

of the requirements for

Upper Division Honors Distinction

Advisor: Dr. Jianhua Chen

Committee Members Dr. Bogdan Oporowski and Dr. Evangelos Triantaphyllou

August 2008

Table of Contents

Abstract

Storing discrete probability distributions is memory-expensive, but approximating these distributions using decomposable Markov networks with a limited treewidth effectively reduces this expense. Finding an optimal Markov network for this problem is NP-hard and, as a result, there exists a need for efficient algorithms that may not yield an optimal solution, like greedy algorithms. Three such greedy algorithms are discussed and empirically compared in this thesis: Malvestuto's Hyperedge Greedy Algorithm [8], HGA; an application of the Forward Selection Algorithm [4], FSA-K; and the Edgewise Greedy Algorithm [5], EGA. These algorithms vary in approximation accuracy and efficiency. Each of these algorithms were implemented into a computer program and experiments were conducted to compare the accuracy and speed of the algorithms to one another under varying conditions. The experiments show that (1) all algorithms produce models with similar accuracy; (2) the EGA is the most efficient; (3) the HGA produces on average a model with higher accuracy if the treewidth is large and a model with lower accuracy if the treewidth is small in comparison to the other algorithms, but is the least efficient; (4) the EGA coupled with a local search procedure produces models with the best accuracy and requires 50 percent more execution time than the EGA without the local search procedure.

1. Introduction

A Markov network is a graphical model of a joint probability distribution. It consists of nodes—the random variables—and undirected edges which represent conditional dependencies between the random variables. The algorithms discussed in this thesis deal with decomposable Markov networks (DMNs) of n discrete random variables, which have an attribute that allows probabilities to be easily attained from it (that is, requires fewer calculations). A k-clique is a complete subgraph consisting of k nodes. A maximal clique of order k is a k-clique that is not contained in any other k^*-clique, for some $k^* > k$. A DMN with a treewidth of k is a graphical model such that the graph is chordal (defined in Chapter 2) and the largest order of the maximal cliques is $k + 1$.

The greedy algorithms discussed in this thesis are designed to construct a DMN of with treewidth k that approximates a probability distribution, which will be referred to as P^*. The distribution attained from the DMN that approximates P^* will be referred to as P_μ. The algorithms achieve this by finding dependencies among variables; a graph with edges between variables that are most dependent upon each other is ideal for capturing a probability distribution. The number of edges in the graph is proportional to the accuracy of P_μ. The goal, however, is to limit the number of edges allowed in the graph, which reduces the data required to maintain the graph; consequently, a DMN constructed by one of the algorithms is limited in number of edges by the size of the predetermined treewidth. Using a high treewidth will produce a P_μ that accurately approximates P^*, and using a low treewidth will produce P_μ that less accurately approximates P^*. There are many P_μ's that can be produced from P^*—each of which vary in its accuracy—and it is each algorithm's objective to produce a P_μ that approximates P^* accurately and efficiently (requiring minimal computational time).

Greedy Algorithms for learning Markov networks are applicable whenever a probability distribution needs to be maintained and the conservation of memory is important. For example, if P^* is a binary probability table of 20 variables, there will be 2^{20} (that is, 1,048,576) vectors in the table; whereas, for a treewidth of 5 P_μ will require a total of $(20 - 5) * 2^5$ (that is, 480) vectors to be stored in a table. In addition, given a large set of training data these algorithms can construct a DMN to define a probability distribution for the set. For instance, a DMN can be constructed from the US Census and be used to find the probability that a person with certain attributes (say, age greater than 50 and married) earns a high income; rather than producing an enormous probability table, a condensed distribution can be attained.

Computer vision, natural language processing, and information retrieval are examples of studies that use statistical classification, in which an object is placed into some class based on the properties of the object. For each class, a probability distribution can maintained so that the probability that an object belongs to this class can be calculated. For some object, the class that yields the highest probability is likely to be the appropriate class for the object. DMNs with limited treewidth are applicable in maintaining these multiple probability distributions for each class.

This thesis will explain each algorithm as well as provide results from experiments that tested the accuracy and efficiency of each algorithm. Chapter 2 introduces definitions and theorems necessary to understand the algorithms. Chapters 3, 4, and 5 explain each of the algorithms respectively: the HGA, the

FSA-K, and the EGA. The design of the experiments is discussed in Chapter 6, and the results of the experiments are discussed in Chapter 7. Chapter 8 discusses future work and concludes the thesis.

2. Definitions

2.1 Chordal Graph, Clique Graph, and Junction Tree

Definition: A *graph* [7] G is an ordered pair $G := (V, E)$ where V is a set of vertices and E is a set of edges.

In this thesis, only undirected edges are mentioned, which are denoted as an unordered pair of two vertices.

Definition: A *chordal graph* [9] is a graph such that every cycle of length at least four has a chord, which is an edge between two nodes that are not adjacent in the cycle.

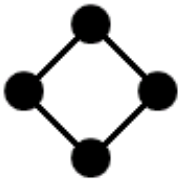


Figure 2.1: A graph that is not chordal

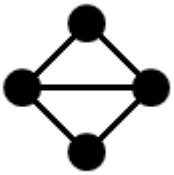


Figure 2.2: A graph that is chordal

Definition: A *minimal separator* of two vertices v_a and v_b of a graph G is the smallest set of vertices (not including v_a or v_b) such that, once removed from G, there exists no path from v_a to v_b (or from v_b to v_a).

Definition: Let $G = (V, E)$ be a chordal graph. The *clique graph* [5] of G, denoted $C(G) = (V_c, E_c, \mu)$, is the weighted graph defined by :

1. The vertex set V_c is the set of maximal cliques of G.
2. An edge (C_1, C_2) belongs to E_c if and only if the intersection $C_1 \cap C_2$ in G is a minimal a,b-separator for each $a \in C_1 \backslash C_2$ and each $b \in C_1 \backslash C_2$.
3. Each Edge $(C_i, C_j) \in E_c$ is given the weight $\mu(C_i, C_j) = |(C_i \cap C_j)|$.

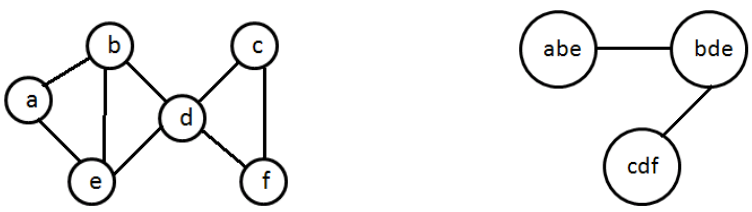


Figure 2.3: A chordal graph and its corresponding clique graph

A clique graph has nodes that represent the maximal cliques of the chordal graph and have edges between the two nodes if (2) is satisfied. The weight is not important in this thesis because all edges in the clique graph that the algorithms produce will have the same weight. The models that the algorithms produce are *edge-maximal*—the addition of any new edges to the chordal graph will increase its tree-width, k. Thus, the weight of every edge in the clique graph is identical, k.

Definition: Let $G = (V, E)$ be a chordal graph. A *junction tree* [5] of G is a maximal spanning tree of the clique graph of G.

Because all weights of the edges are the same in the models produced by the algorithms, any spanning tree can be a junction tree.

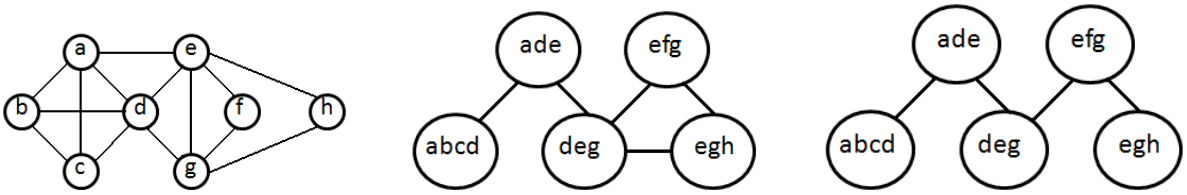


Figure 2.4: (left) a chordal graph G, (middle) a clique graph of G, and (right) a junction tree of G.

Figure 2.4 shows a possible junction tree derived from the clique graph of G. The edge (deg,egh) is removed from the clique graph to form this junction tree, but the edges (efg,egh) or (deg,efg) could have been removed instead to form a different junction tree. Each of these three edges corresponds to the same minimal separator {e,g}, and any of the possible junction tree created from the clique graph of G would have two edges that correspond to this separator. Thus, the set of separators in the junction tree is a multiset—{e,g} would appear twice in the set.

2.2 Probability Distribution Modeled by a DMN

Let the chordal graph G be a DMN for probability distribution $p_{\boldsymbol{X}}$, where $\boldsymbol{X} = \{X_1, X_2, \ldots, X_n\}$. Let $T = (V_c, E_t)$ be a junction tree for G and let $\boldsymbol{S}$ denote the multiset $\{C_i \cap C_j \mid (C_i, C_j) \in E_t\}$.

Theorem 2.1: The joint probability distribution and the marginal distribution satisfy

$$p_{\boldsymbol{X}} \cdot \prod_{S \in \boldsymbol{S}} p_S = \prod_{C \in V_c} p_C$$

Proof: This result is due to [11], and in this form the proof appears in [3], Theorem 4.3.2.

With this factorization a probability distribution does not need to be in the form of a large probability table consisting of all possible vectors. If the probability distribution of each clique is known then one can compute both the probability distribution of each separator (notice that separators are subsets of a clique, thus its probability distribution can be attained from the probability distribution of a clique) and

the large probability distribution, p_X. Using the example in the introduction—the probability distribution consists of 20 random variables and the treewidth of the DMN is 5—15 probability tables with 32 entries each has all the information one large probability table would have (which has 2^{20} entries). The probability distributions of the cliques could be stored as multiple probability tables.

If there are no independent variables, then this factorization is not applicable, because the DMN would be a complete graph—it will have one clique and no separators. This factorization, however, can still be applied if the probability distribution, P^*, is approximated. A DMN of treewidth k can be constructed to model an approximation of P^*, but constructing an optimal DMN of treewidth k is an NP-hard problem. Thus, finding an optimal DMN of treewidth k would not be feasible. Greedy algorithms can efficiently produce a DMN of treewidth k that models an accurate, but seldom optimal, approximation of P^*, which will be referred to as P_μ. P_μ can be extracted from the DMN by

$$p_\mu = \prod_{C \epsilon V_c} p_C \Big/ \prod_{S \epsilon \boldsymbol{S}} p_S$$

where V_c is the set of vertices in the associated junction tree of the DMN and $\boldsymbol{S}$ is the set of separators, as described in theorem 2.1. Now, P^* would require significantly less data to maintain in the form of P_μ, because P_μ utilizes the factorization in theorem 2.1 more effectively—how effectively depends on the choice of k.

2.3 Entropy

The greedy algorithms in this thesis must choose edges so that the resulting DMN will be able to accurately approximate a probability distribution. In determining which edge to choose, the algorithms calculate entropy. Entropy is a measurement of surprise; it measures the uncertainty associated with a random variable. The following definitions deal with the entropy of a random vector.

Definition: For some discrete random variable X, the *entropy* [12] of X is defined by

$$H(X) = -\sum_x p(x) log_2 p(x)$$

where the sum is over all possible values x of X.

Definition: For some discrete random vector X, the *conditional entropy* [12] is defined by

$$H(Y|X) = -\sum_x \sum_y p(y,x) log_2 p(y|x)$$

where the double sum takes all possible values x of X and y of Y. The chain rule for entropy is defined by

$$H(Y,X) = H(X) + H(Y|X)$$

Conditional entropy is the uncertainty that an event Y occurs given an even X has already occurred.

Definition: The *K-L divergence* [6] of probability distributions P^* and P_μ over $\mathbf{X}$ is defined

$$D_{KL}(P^*||P_\mu) = \sum_x P^*(x)\log\left(\frac{P^*(x)}{P_\mu(x)}\right)$$

where the sum is over all possible vectors x of $\boldsymbol{X}$.

The difference between the approximated probability distribution attained from the DMN and the real probability distribution is measured by K-L divergence. In this study, a DMN (and the corresponding P_μ) is an optimal solution to the approximation problem if the DMN minimizes the K-L divergence.

The K-L divergence can be given equivalently in terms of entropy: $D_{KL}(P^*||P_\mu) = H_{P_\mu}(X) - H(X)$, [10]. $H_{P_\mu}(X)$ denotes the entropy of the DMN created to approximate P^*. Thus, by reducing the entropy of the DMN, the K-L divergence is reduced as well (the value of $H(X)$ is not affected by the structure of the DMN).

3. The Hyperedge Greedy Algorithm

The Hyperedge Greedy Algorithm constructs a DMN of treewidth k by iteratively adding hyperedges of order $k+1$, and each addition results in the formation of a new maximal clique of order $k+1$. Thus, adding a hyperedge is equivalent to adding a maximal clique to the DMN. Because each maximal clique will be of the same order, the model is said to be *uniform*. In considering which clique to add to the model, the clique that yields a model with the least entropy and results in another uniform model is selected. After a few terms have been defined, the HGA will be explained more in depth.

The following paragraph is from [5]. Let T be a junction tree of a chordal graph G. Choose a vertex of T to be a root, thus directing all edges in T. This induces a partial order of the maximal cliques of G by $C_i \leq C_j$ if there is a directed path from C_i to C_j. A numbering C_1, C_2,..., C_s of the maximal cliques that is compatible with this partial order is a *running intersection order* of the maximal cliques. Following Malvestuto, for a given uniform model with maximal cliques $C_1, C_2 \dots C_m$, a maximal clique C_h, $h > 1$, is called elementary (with respect to this ordering) if the set $C_h \backslash (C_1 \cup C_2 \dots C_{h-1})$ is a singleton. A chordal graph is called *elementary* if it has a rooted junction tree such that every maximal clique, except the root, is elementary.

Theorem 3.1: For a given model M the entropy of the model is equivalent to:

$$H(M) = \sum_{c \in C} H(c) - \sum_{s \in S} H(s)$$

where C is the set of maximal cliques and S is the multi-set of separators in the junction tree of the model [8].

At each iteration of the HGA a new maximal clique of order $k+1$ is added to the DMN. The HGA chooses the clique that minimizes $H(c) - H(s)$ because this is a term in the equation from theorem 3.1. It is apparent that the difference $H(c) - H(s)$ is the increase in entropy due to adding the clique c. Recall that reducing the entropy of the model is equivalent to reducing the K-L divergence of P^* and P_μ, and, therefore, improves the approximation accuracy of P_μ.

Given a joint probability distribution for discrete random variables $\{X_1, X_2, X_3, \dots, X_n\}$ we can describe Malvestuto's Hyperedge Greedy Algorithm. Let G be a chordal graph. The following steps are taken from [5].

Step 1: Calculate the marginal entropy $H(C_1)$ of all subsets of order $k+1$ from the set $\{X_1, X_2, X_3, \dots, X_n\}$, and choose the subset with the least marginal entropy.

Step 2: Suppose cliques $C_1, \dots, C_h$ have already been chosen. Consider all subsets C of $\{X_1, \dots, X_n\}$ of order k+1 such that the chordal graph with maximal cliques $C_1, \dots, C_h, C$ is elementary. In the associated rooted junction tree of G_c, let C_0 denote the parent of C and put $S_c = C_0 \cap C$. From these eligible subsets C, choose one that minimizes the quantity $H(C) - H(S_C)$.

Step 3: Stop when all the X_i appear in some clique. By the definition of elementary, this will occur after $n-(k+1)+1=n-k$ cliques of order $k+1$ have been selected.

When the algorithm is complete a DMN with treewidth k is constructed as well as a junction tree which is necessary for application of Theorem 2.1.

The cliques have an order of $k+1$ and all possible cliques from a set of n nodes must be produced. This is equivalent to finding all subsets of size $k+1$ from a set of size n. For each clique produced the entropy, $H(c)$, must be computed which calculates the entropy between $k+1$ variables. Therefore, the complexity of the HGA is $O\left(\binom{n}{k+1}E\right)$ where E is the complexity in computing the entropy between $k+1$ variables.

3.1 An example using the HGA

The following example will demonstrate how the HGA constructs a DMN of treewidth 2 using the probability table in figure 3.1. Starting from the furthest left column, the columns will represent the variables A, B, C, and D.

0	0	0	0	0.029411
0	0	0	1	0.117647
0	0	1	0	0.014705
0	0	1	1	0.058823
0	1	0	0	0.058823
0	1	0	1	0.117647
0	1	1	0	0.117647
0	1	1	1	0.117647

1	0	0	0	0.014705
1	0	0	1	0.058823
1	0	1	0	0.014705
1	0	1	1	0.117647
1	1	0	0	0.058823
1	1	0	1	0.058823
1	1	1	0	0.029411
1	1	1	1	0.014705

Figure 3.1: A discrete probability table for five binary variables

The first step of the HGA is to calculate the entropy between all subsets of size 3 among the 4 variables.

Cliques	Entropy
{A,B,C}	2.841252
{A,B,D}	2.750213
{A,C,D}	2.852867
{B,C,D}	2.829196

Figure 3.2: Cross entropy for cliques of order 3

{A,B,D} has the least entropy among the other sets. It will be our first clique and the root of the junction tree. The remaining three sets are elementary and are eligible to be the next clique in the

graph. Notice that each of the remaining cliques contains C, which is correct because C is the only variable that is not already in the graph. Now we must compute the separators between each potential clique and its parent ({A,B,D} is the parent of each).

Cliques	H(C)	Separator	H(S)	H(C)-H(S)
{A,B,C}	2.841252	{A,B}	1.902668	0.944584
{A,C,D}	2.852867	{A,D}	1.871342	0.981328
{B,C,D}	2.829196	{B,D}	1.838256	0.99094

Figure 3.3: Entropy of potential cliques and their separators

The clique that minimizes $H(C) - H(S)$ is {A,B,C} and, so, it is selected to be the next, and final, clique in the junction tree. Each variable now appears in the chordal graph; therefore, the algorithm is complete. The resulting chordal graph and junction tree is displayed in figure 4.4.

Figure 3.4: The resulting chordal graph and junction tree from the example

The K-L divergence of this model is $H(\{A,B,D\}) + H(\{A,B,C\}) - H(\{A,B\})$ which equals 0.00908.

4. The Treewidth k Forward Selection Algorithm

Both the FSA-K and the EGA use the Forward Selection Algorithm (FSA) as a subroutine. The Forward Selection algorithm constructs a DMN with no restriction on treewidth. The FSA-K is an application of the FSA for learning Markov networks of treewidth *k*; the FSA-K constrains the FSA so that the DMN being constructed never exceeds a determined treewidth.

The FSA adds one edge at a time to a chordal graph so that the resulting graph is chordal as well. A clique graph is maintained in addition to the chordal graph, and the status of the clique graph determines which edges upon addition maintain chordality. After each edge is added, the clique graph is updated. Of all eligible edges, the edge that maximizes the difference of entropy between the current DMN and a new DMN that includes the edge is selected. When the process is complete, there will be $n - k$ cliques of order $k + 1$ in the chordal graph. A junction tree is extracted from the clique graph—by removing edges so that the properties from definition of a junction tree are satisfied—and is used to formulate the joint probability distribution. The FSA can be broken down into three phases in each iteration: determining which edges are eligible, choosing an edge, and updating the clique graph.

4.1 Eligibility of an Edge for Addition

The clique graph is used to determine which edges upon addition maintain chordality of a graph.

Lemma 4.1: Let $G = (V, E)$ be a chordal graph in which vertices a and b are not adjacent, and let G' be obtained from G by adding the edge (a, b), Then G' is chordal if and only if there exist maximal cliques C_a and C_b of G such that $a \in C_a, b \in C_b$, and (C_a, C_b) is an edge in the clique graph of G (i.e., $S_{ab} = C_a \cap C_b$ is a minimal a, b separator) [4].

Figure 4.1: A chordal graph and its corresponding clique graph

In figure 4.1, the edge (B,D) is eligible for addition—it maintains chordality of the graph—because cliques {B,C} and {C,D} contain B and C, respectively, and these cliques share an edge in the clique graph. Thus, by Lemma 4.1, the resulting graph after addition of this edge will be chordal. Similarly, edge (A,C) is eligible. Edge (A,D) is not eligible, because there is no pair of cliques, to which one contains A and the

other contains D, that share an edge in the clique graph. By observing figure 4.1, the addition of this edge would form a cycle of length 4 that has no chords, thus the resulting graph would not be chodral.

The FSA-K requires that eligible edges for addition to the chordal graph do not create a clique with clique of order greater than k+1. The following corollaries from [4] help determine which edges satisfy this criteria, and those edges that do satisfy this criteria are said to be k-admissible (defined below).

Corollary 4.1: Under the conditions of Lemma 4.1, $C_{ab} = \{a, b\} \cup (C_a \cap C_b)$ is the only new maximal clique of G'.

Corollary 4.2: Let G be a chordal graph of treewidth k and suppose the chordal graph G' is obtained from G by adding the edge (a, b). Then G' has treewidth k if and only if the maximal clique C_{ab} has order at most $k + 1$.

Proof: The result follows from Corollary 4.1 and the fact that the treewidth of a chordal graph is one less than the largest order of a maximal clique [1].

Definition: Let G be a chordal graph and let a and b be vertices that are not adjacent in G. We say that the edge (a, b) is k-*admissible* [5] if the graph G' obtained by adding (a, b) to G is chordal of treewidth at most k.

The FSA-K allows only edges that are k-admissible to be eligible for addition to the chordal graph. This is the modification to the FSA that allows the creation of a DMN of treewidth k.

4.2 Selection Criterion

Theorem 4.2: If the chordal graph G' is obtained from the chordal graph G by adding the edge (v_a, v_b), then the difference in entropy between the two DMNs, $H(G) - H(G')$, equals

$$-H(S_{ab}) + H(v_a, S_{ab}) + H(v_b, S_{ab}) - H(v_a, v_b, S_{ab})$$

where S_{ab} is the minimal separator of v_a and v_b in G [4].

Definition: Given two random variables X and Y, *mutual information* of X and Y is defined by

$$\mathrm{I(X;Y)} = \sum_{\mathrm{x}} \sum_{\mathrm{y}} \mathrm{p(x,y)} \log \frac{\mathrm{p(x,y)}}{\mathrm{p(x)p(y)}}$$

where the double sum is over all possible values x of X and y of Y. In addition conditional mutual information can be defined as $\mathrm{I(X;Y)} = \mathrm{H(X)} + \mathrm{H(Y)} - \mathrm{H(X,Y)}$; see [2], Theorem 2.4.1. Note that $I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$, [5].

Definition: *Conditional mutual information* [2] of a variables X and Y given Z, denoted by $I(X;Y|Z)$ is defined by

$$I(X;Y|Z) = H(X|Z) - H(X|Y,Z)$$

Theorem 4.2 can be expressed in terms of conditional mutual information.

Corollary 4.3: With assumptions as in Theorem 4.2, the difference in entropy equals $I(v_a; v_b|S_{ab})$.

Proof: By the chain rule for entropy ([2], Theorem 2.5.1), we have

$$H(v_a, S_{ab}) = H(S_{ab}) + H(v_a|S_{ab})$$

$$H(v_b, S_{ab}) = H(S_{ab}) + H(v_b|S_{ab})$$

$$H(v_a, v_b, S_{ab}) = H(S_{ab}) + H(v_b|S_{ab}) + H(v_a|v_b, S_{ab})$$

Replacing these the entropies on the right in the equation from Theorem 4.2 with the entropies on the left yields $H(v_a|S_{ab}) - H(v_a|v_b, S_{ab})$ which equals $I(v_a; v_b|S_{ab})$ [5]. ■

Of the eligible edges, the edge with vertices v_a and v_b that gives the high value of $I(v_a; v_b|S_{ab})$ is chosen as the next edge to be added to the graph. Mutual information measures how dependent two variables are to each other. A high mutual information implies that the state of one variable is dependent on the another and/or vice versa. It is ideal that two variables that share a high mutual information share an edge between their corresponding nodes in the DMN. Remember, an ideal DMN approximates a probability distribution by determining which variables are independent to others. If two variables x, y are independent then $p(x, y) = p(x)p(y)$; thus, a probability distribution table does not need to be maintained for $p(x, y)$, only for $p(x)$ and $p(y)$. This is the underlying tactic that enables a DMN to condense and approximate a probability distribution. According to Corollary 4.3 an edge between two vertices with high mutual information conditioned by their minimal separator contributes to a model with less entropy. If two variables share a high mutual information given all the neighboring variables shared by each of them, then placing an edge between these two variables is important in capturing the dependency that exists between the two variables.

4.3 Updating the Clique Graph

In the FSA (and therefore the FSA-K) when an edge is added, the clique graph must be updated. The steps below provide an efficient way to update the clique graph. Following the notation from [4], let $G = (V, E)$ be the original chordal graph and let CG_G be the clique graph, let the new edge that is added be (v_a, v_b), the corresponding edge in the clique graph be (C_a, C_b), and let S_{ab} = $C_a \cap C_b$. The new model and clique graphs will be G' and $CG_{G'}$.

A new maximal clique will be created as a result of adding (v_a, v_b). This clique, C_{ab}, is equal to $S_{ab} + v_a + v_b$. The clique graph must be updated to include this change. This clique will share an edge with both C_a and C_b in CG_g.

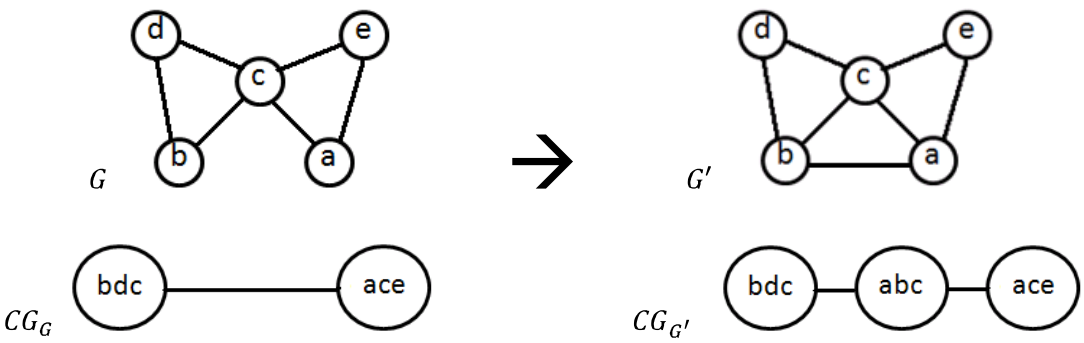


Figure 4.2: Example of the change in a chordal graph and its clique graph after addition of an edge

In figure 4.2 v_a is a, v_b is b, C_a is the clique {a,c,e}, C_b is the clique {b,c,e}, and S_{ab} is the set {c}. The addition of the edge (a, b) creates a new clique, {a,b,c}.

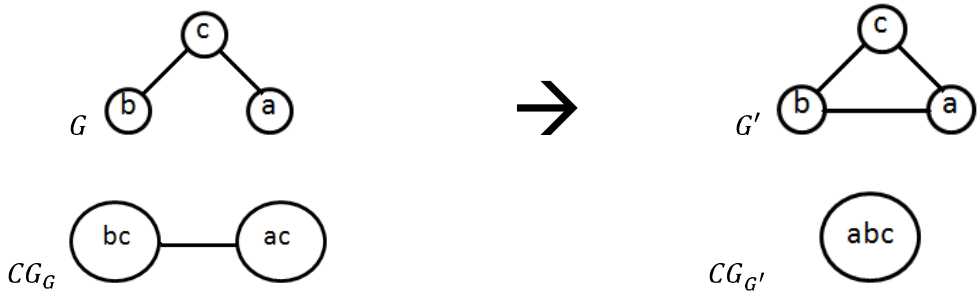


Figure 4.3: Another example of an edge of the change of a chordal graph and its clique graph after the addition of an edge

In figure 4.3 the new clique is a superset of the initial two clique, thus the intial two cliques are no longer maximal cliques and do not exist in the clique graph. Both 4.2 and 4.3 only show a partial structure of a clique graph. Upon addition of an edge, the relationship between C_a, C_b, C_{ab} and other cliques must be updated because some edges between other cliques may no longer satisfy Theorem 4.1. The following steps are taken from [4]. In congruence with these steps figure 4.4 will be a used as an example. Note that v_a is a, v_b is b, C_a is the clique {a,c,e}, C_b is the clique {b,c,d}, S_{ab} is the set {c}, and C_{ab} is the clique {a,b,c}.

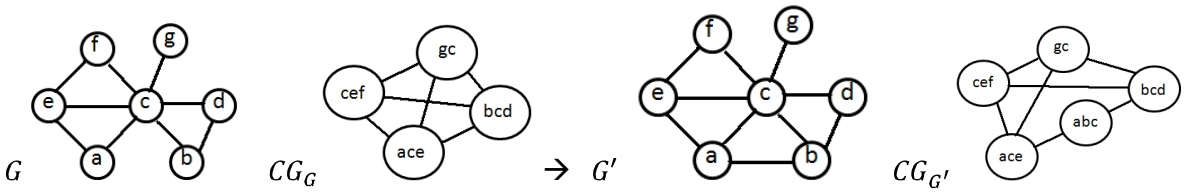
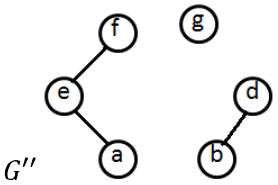


Figure 4.4: An example of a chodral graph and clique graph after the addition of the edge (a,b) and before updating the clique graph

1. Let $G'' = G - S_{ab}$. Find all nodes that are connected to v_a and to v_b in G'' and maintain this information in two arrays indexed by the vertices of G.



	a	b	c	d	e	f	g
a	False	False	False	False	True	True	False
b	False	False	False	True	False	False	False

Figure 4.5: An example of G'' and arrays maintaining which vertices a and b are connected to in G''

2. **Deciding whether to keep an edge** $(C_1, C_2) \in CG_{G'}$: Let $S_{12} = C_1 \cap C_2$. If $S_{12} \neq S_{ab}$, then keep this edge. Otherwise, consider the graph $G - S_{12}$ $(= G'')$. If v_a is connected to $C_1 \backslash S_{12}$ in this graph and v_b is connected to $C_2 \backslash S_{12}$ or *vice versa*, do not keep this edge in $CG_{G'}$. Otherwise, keep it. This check can be performed in $O(1)$ time using the arrays constructed in Step 1. (In figure 4.4, the edge in G'' (cef,bcd) must be removed).

3. **Adding edges involving C_{ab}**:

   1. For every maximal clique C' such that $(C', C_a) \in CG_{G'}$ (after execution of the above step), add an edge (C', C_{ab}) to CG_G, if $(C' \cap C_a) \subset S_{ab} + v_a$. (From figure 4.4 the edge (gc,abc) should be added to G'').
   2. For every maximal clique C' such that $C' \cap C_{ab} = S_{ab} + v_a$, explicitly check if this edge should be added to the clique graph. This requires checking whether v_b is separated from $C' \backslash S_{ab} + v_a$ by $S_{ab} + v_a$, which can be done by looking up the arrays computed in Step 1.
   3. Repeat the above two steps for C_b and v_b instead of C_a and v_a.

4. Remove C_a (C_b) if it is contained in C_{ab}

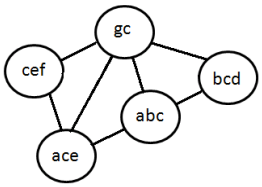


Figure 4.6: The resulting clique graph from the example after updating

4.4 The FSA-K Algorithm

Input: The joint (empirical) probability distribution of n discrete random variables $X_1, X_2, X_3, \dots, X_n$ and a positive integer $k < n$.

Output: A DMN (chordal graph) with n vertices and treewidth k.

Initialize: Let G be the graph with vertices $X_1, X_2, X_3, \dots, X_n$ and no edges.

Step 1: Use the FSA (modified to take into account the order of the potential new clique) to enumerate all k-admissible edges. If there are no k-admissible edges, stop.

Step 2: For each k-admissible edge (X_i, X_j), compute $I(X_i; X_j | S_{ij})$ and add an edge with the maximum value.

Step 3: Perform the necessary updating in the FSA and go to Step 1.

4.5 Complexity Analysis

According to [4], the algorithm consists of two components at each step: determining all edges that can be added to the model and determining which of these edges to add to the model. Determining all edges that can be added to the model consist of updating the clique graph and finding those edges that satisfy Lemma 4.1. Determining which edges satisfy Lemma 4.1 requires $O(n^2)$ time; for every node a search must be made from the arrays computed in the first step of updating the clique graph. Updating the clique graph also requires $O(n^2)$ time. Step 1 can be done in $O(n^2)$ time; for each node, except v_a and v_b, a search which is linear in n is performed to determine its connectivity among other nodes in G''. Similarly, Step 3 can be done in $O(n^2)$ time. Step 2, however, determines if a two sets are equal ($O(n)$ time) for each edge in the clique graph (number of edges $\in O(n^2)$) which seems to be a total of $O(n^3)$ time. Deshpand-Garofalakis-Jordin (2001) provide an efficient alternative—by storing all separators and the edges that point to these separators in a data structure—that reduces the complexity to $O(n^2)$. Thus, determining all edges that can be added to the model can be done in $O(n^2)$ time. The

time required in determining which of the eligible edges to add to the model is dependent on the number of entropies calculated.

Theorem 4.3: The number of new entropies that need to be computed after adding the edge (v_a, v_b) to the underlying model graph (*i.e.* after performing one forward selection step) is at most $2(n - n_a) + 2(n - n_b)$, where n_a and n_b are the number of neighbors of v_a and v_b, respectively [4].

Let E be the complexity of calculating entropy, then theorem 4.3 implies that the complexity in determining which edge to add to the model is $O(nE)$. Thus, the complexity at each step of the FSA is $O(n^2 + nE)$.

The model produced by the FSA-K is edge-maximal. Therefore the separator between every clique is of size k. The number of edges shared by each clique is the number of edges in a separator, $\binom{k}{2}$, and the number of edges in a clique is $\binom{k+1}{2}$. There are a total of $n - k$ cliques and $n - k - 1$ separators. Therefore the total number of edges is equal to $\binom{k+1}{2}(n-k) - \binom{k}{2}(n-k-1)$ which can be reduced to $nk - \frac{(k+1)k}{2}$. Because $k < n \rightarrow k^2 < nk$, the complexity of the number of edges in a model is $O(nk)$. Therefore, the total complexity of the FSA is $O((n^2 + nE)(nk)) = O(kn^3 + kn^2E)$. The FSA-K adds a restriction at each iteration that requires $O(n^2)$ time: determining if S_{ab} of C_a and C_b is of size less than k+1 for all eligible vertices, v_a and v_b. Therefore, the FSA-K has the same complexity as the FSA.

4.6 An example using the FSA-k

The following example uses the same probability table as in Chapter 3 (Figure 3.1) to produce a model of treewidth 2. Starting with a graph containing no edges, an edge whose vertices possess the highest mutual information is selected.

Variables X,Y	I(X,Y)
A,B	0.030523
A,C	0.000047
A,D	0.000625
B,C	0.000013
B,D	0.069207
C,D	0.001960

Figure 4.7: Mutual information among pairs of variables

The pair {A,B} has the highest mutual information in figure 4.7, so the first edge in the graph will be (A,B) and {A,B} will be the first clique of order 2. By the definition of a clique graph, an edge will be shared between the cliques of order 1 and every other clique because, although the separator is an empty set,

a vertice of a clique of order 1 is disjoint to all other vertices in the graph. In addition, by using the converse of lemma 4.1, if the addition of a new edge between two vertices results in another chordal graph then there must be an edge between the cliques that contain those vertices. It is clear that adding an edge between two nodes that are not connected (share no path from one to the other) results in a chordal graph.

Figure 4.8: The chordal graph (left) and clique graph (right) after the first iteration of the FSA-K

The first iteration of the algorithm is complete. All remaining edges are still eligible.

Variables X,Y	I(X,Y)
A,C	0.000047
A,D	0.000625
B,C	0.000013
B,D	0.069207
C,D	0.001960

Figure 4.9: The mutual information among pairs of variables

The edge (B,D) is the next edge to add because B and D share the highest information. There are no separators that are not the empty set, so there is no need to compute conditional mutual information yet.

Figure 4.10: The chordal graph (left) and clique graph (right) after the second iteration

The second iteration is complete. All edges are still eligible, but the conditional mutual information must be calculated for A and D given B because B is the separator between the cliques {A,B} and {B,D}.

$I(A; D|B)$ is 0.006368. Because this value is greater than any other mutual information than in Figure 4.7, the next edge added is (A,D).

Figure 4.11: The chordal graph (left) and clique graph (right) after the third iteration

The only three possible edges left are (A,C),(B,C), and (A,D), and each of these is eligible. The mutual information between these edges can be found in figure 4.9. The pair {C,D} has the highest mutual information, and, so, it will be the next edge.

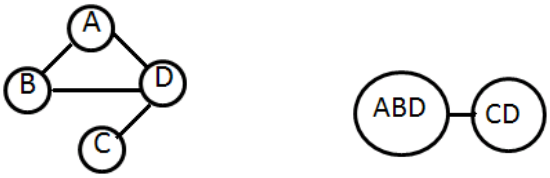


Figure 4.12: The chordal graph (left) and clique graph (right) after the fourth iteration

Two possible edges remain that can be added to the graph and both are eligible: (A,C) and (B,C). C belongs to the clique {C,D}, so the minimal separator of C and A, and C and B is {D}. Hence, $I(A; C|D)$ and $I(B; C|D)$ must be computed. Their values are 0.015891 and 0.006476, respectively. Therefore, the edge (A,C) should be added.

Figure 4.13: The chordal graph (left) and clique graph (right) after the fifth iteration

There are no k-admissible edges left; the addition of the edge (B,C) creates a clique that exceeds the limit (a clique of order $k+2$). Therefore, the algorithm is complete. The junction tree created from this DMN is equivalent to the clique graph. The K-L divergence of this model is 0.038844.

5. The Edgewise Greedy Algorithm

The Edgewise Greedy Algorithm is similar to the FSA-K; in addition to using the Forward Selection algorithm as a subroutine and constraining the treewidth of the DMN, it recursively constructs a DMN with treewidth $r + 1$ from a DMN with treewidth r until r equals k. In other words, for increasing values of r (starting at 1) the EGA does not allow a DMN to have a clique of size $r + 2$ until the DMN is an edge-maximal DMN of treewidth r—that is, until there no more r-admissible edges.

The EGA first constructs a tree, which has a treewidth of 1. When there are no available edges to add that do not increase the treewidth of the DMN, the algorithm constructs a tree with treewidth of 2 and so forth. Recall that the FSA-K only restricts edges that exceed the treewidth, k. The FSA-K may have maximal cliques of order 2 as well as maximal cliques of order 4; whereas, in the EGA the difference between the largest maximal clique's order and the smallest maximal clique's order is at most one. The following algorithm is taken from [5].

Input: The joint (empirical) probability distribution of n discrete random variables $X_1, X_2, X_3, \dots, X_n$ and a positive integer $k < n$.

Output: A DMN (chordal graph) with n vertices and treewidth k.

Initialize: Let G be the graph with vertices $X_1, X_2, X_3, \dots, X_n$ and no edges. Put $r = 1$.

Step 1: Use the FSA (modified to take into account the order of the potential new clique) to enumerate all r-admissible edges. If there are no r-admissible edges, proceed to Step 4.

Step 2: For each r-admissible edge (X_i, X_j), compute $I(X_i; X_j | S_{ij})$ and add an edge with the maximum value.

Step 3: Perform the necessary updating in the FSA and go to Step 1.

Step 4: If $r = k$, stop; else, increment r by 1 and proceed to Step 1.

The EGA incorporates no new procedure to the FSA-K, and, so, it is apparent that it shares the same complexity as the FSA-K, $O(kn^3 + kn^2E)$.

5.1 The Local Search Procedure

In addition to the procedure above, the EGA may be coupled with a local search procedure that attempts to reduce the entropy of the produced model by exchanging edges between some vertices. The procedure will reiterate until no new edges can be exchanged that result in a reduction of the model's entropy. The remainder of the chapter describing the local search procedure is taken from [5].

A k-tree is a graph defined recursively by the following axioms:

(A1) The complete graph on $k+1$ vertices is a k-tree.

(A2) If G is a k-tree and C is any k-clique of G, then the graph obtained by adding a new vertex x and adding all edges between x and the vertices in C is again a k-tree.

Figure 5.1: An example of (A2), C is the set $\{y, w, v\}$

The local search procedure requires that the chordal graph be edge-maximal—that is, the addition of any new edge would result in an increase in treewidth. The EGA does, indeed, produce models that are edge-maximal. The local search procedure will be defined as follows. Let T_n^k denote the set of all k-trees on n vertices. Assume $n > k+1$. Let $G \in T_n^k$. A simplicial vertex is one whose neighbors form a clique, or, equivalently, a vertex of degree k. By a theorem of Dirac (Lemma 2.9 of Lauritzen (1996)), G has at least two non-adjacent simplicial vertices. Let x be a simplicial vertex of G and let W denote the set of neighbors of x in G. The vertices in W form a k-clique in G because G is a k-tree. Put $W' = \{y \in V(G) | \{y\} \cup W \text{ is a } (k+1)\text{-clique in } G\}$.

Let $y \in W'$. Notice that $W = (\{x\} \cup W) \cap (\{y\} \cup W)$ is a minimal separator of x and y. It follows from Lemma (1 from the paper) that the graph $G^* = G + (x, y)$ obtained from G by added the edge (x, y), is chordal. Let $w \in W$. Then the graph G' = $G^* - (x, w)$, obtained by deleting the edge (x, w), is chordal by Lemma 2.19 of Lauritzen (1996) since $\{x, y\} \cup W$ is the only maximal clique of G^* that contains the edge xw. It follows from Rose (1974) that G'$\in T_n^k$. The Figure 5.2 shows an example of this type of exchanging of edges.

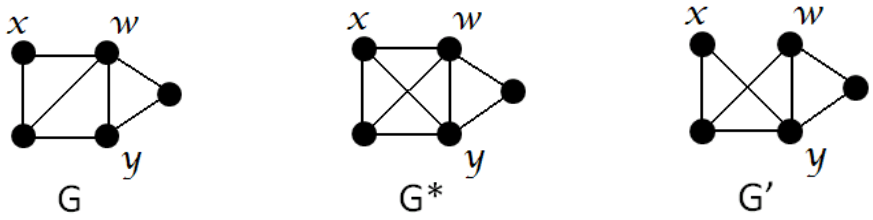


Figure 5.2: An example of an edge being exchanged between two 2-trees

The entropy of the new model produced after the local search can be calculated by applying Corollary 2.1 twice: once for calculating the difference of entropy between G and G^* and a second time for calculating the difference of entropy between G^* and G'.

Theorem 5.1: Let G be the output of the EGA, and assume $n > k+1$. Let X_i be a vertex of degree k in G. Let W denote the set of neighbors of X_i in G. Let $X_j \in W$ and let $X_l \neq X_i$ be a common neighbor of

the k neighbors in X_i. Let G' be the DMN obtained from G by adding the edge (X_i, X_l) and deleting the edge (X_i, X_j). Then $H(G') = H(G) - I(X_i; X_l|W) + I\left(X_i; X_j \middle| \left(W \backslash \{X_j\}\right) \cup \{X_l\}\right)$, [4].

G' will have less entropy than G if $I(X_i; X_l|W) > I\left(X_i; X_j \middle| \left(W \backslash \{X_j\}\right) \cup \{X_l\}\right)$. If G is the DMN produced by the EGA, then the local search procedure is informally described as follows:

Step 1: Identify all vertices of G of degree k. (These are the simplicial vertices.)

Step 2: Choose a vertex X_i of degree k.

Step 3: Identify W, the set of k neighbors of X_i, and W', the set of vertices other than X_i that are adjacent to all vertices in W.

Step 4: Choose $X_j \in W$ and $X_l \in W'$.

Step 5: Compute $I(X_i; X_l|W) - I\left(X_i; X_j \middle| \left(W \backslash \{X_j\}\right) \cup \{X_l\}\right)$. If this quantity is positive, then replace G by the DMN G' obtained by adding the edge (X_i, X_j), and go to step 1; else, repeat steps 3 and 4 until all pairs in $W \times W'$ have been checked.

Step 6: Go to Step 2 until all vertices of degree k have been used.

5.2 An example using the EGA

The following example uses the same probability table as in Chapter 3 (Figure 3.1), which is also used in the FSA-K example. The first two iterations are exactly the same as the first two iterations of the FSA-K: the edge {A,B} is selected among all other pairs of variables because A and B share the highest mutual information and the edge {B,D} is selected next. The status of the current graph is displayed in figure 4.8.

Our current r-value is 1 so the only edges eligible are the r-admissibles edges: {(A,C),(B,C),(C,D)}. Notice that (A,D), which was selected as the next edge in the FSA-K, is not r-admissible because it will form a clique of order 3, which is greater than r+1. C and D share the highest mutual information (see figure 4.7) among those edges listed, and, so, the pair (C,D) will be the next edge added to the model.

Figure 5.3: the chordal graph (left) and clique graph (right) after iteration 3.

After the third iteration, there are no r-admissible edges; thus, r must be incremented—to 2. Now cliques of order 3 may be created as a result of adding an edge. Those edges that are now r-admissible are (B,C), (A,D), and (A,C), but notice that (A,C) is not eligible for addition because it will result in graph that is not chordal. This can be seen from the clique graph in figure 5.3: the cliques AB and CD do not share an edge, and, so, by lemma 4.1 the addition of edge (A,C) would not result in a chordal graph. The computations $I(B;C|D)$ and $I(A;D|B)$ must be calculated because D is the sepator between cliques BD and CD, and B is the separator between AB and BD. The resulting values are 0.00648 and 0.006368, respectively. Therefore, (B,C) is the next edge added to the model.

Figure 5.4: The chordal graph (left) and clique graph (right) after iteration 4.

With the addition of the edge (B,C), the edge (A,C) is now eligible. In addition, the edge (A,D) is still eligible as well. Computing $I(A,C|B)$ yields 0.060778, which is greater than $I(A,D|B)$ as given above. The edge (A,C) is added to the model.

Figure 5.5: The chordal graph (left) and clique graph (right) after iteration 5.

Adding the edge (A,D) will increment the models treewidth and exceed r. Therefore r must be incremented; however, r equals k, thus the algorithm is finished. The resulting DMN is indeed an edge-maximal model of treewidth 2. The K-L divergence of this model is 0.007655, which happens to be the least among the three algorithms.

The local search procedure may produce a model with lesser K-L divergence. The simplicial vertices of the final graph as displayed in figure 5.5 are A and D (they are each of degree 2). It is apparent that (A,D) is the only edge that may be added if another edge is removed. Let X_i be A. Then, W is {B,C}, and X_l is D. First, let X_j be B. $I(A;D|\{B,C\}) - I(A;B|\{C,D\})$ equals -0.074785. This number is not positive and, therefore, will not result in a model with lower K-L divergence. Now, let X_j be C. $I(A;D|\{A,B\}) - I(A;C|\{B,D\})$ equals -0.054411. Again, this number is not positive and will not result

in a better model. Now, let X_i be D. W remains as {B,C}. X_l is A. Let X_j be B. Then, $I(A; D|\{B, C\}) - I(D; B|\{C, A\})$ equals -0.059199, which is not positive. Let X_j be C. Then, $I(A; D|\{B, C\}) - I(D; C|\{B, A\})$ equals -0.002054; the resulting model would be identical to the model produced by the HGA. There are no edge exchanges that result in a model with lesser K-L divergence. The local search procedure did not improve the accuracy of the EGA for this particular example.

There are 6 different DMNs of treewidth 2 that can be created from 4 variables—must choose 2 cliques of order 3 out of a possible 4. 3 of them were produced by the algorithms, and 3 more were considered during the local search procedure. Of all 6 possible DMNs, the EGA produced the *optimal*.

6. Experiment Design

A Java program was written to implement each algorithm and experiments were conducted to compare the performance of each algorithm from data attained by the program. The experiments compared two attributes of each algorithm: accuracy and efficiency. Accuracy was measured by K-L divergence and efficiency was measured by the execution time of the program while the algorithm was executing. Each algorithm has two inputs: the number of variables to be in the joint probability distribution, n, and the treewidth of the DMN, k. The algorithm creates a DMN as an approximation to a randomly generated joint probability distribution of n random variables. It returns either the execution time, or computes and returns the K-L divergence. The experiments that measured K-L divergence and the experiments that measured execution time differ slightly.

The experiments that measure the accuracy of each algorithm compute an average K-L divergence for different treewidths. These experiments hold n constant and use k as the independent variable. These experiments were conducted as follows. Let i be the number of iterations the algorithms execute before an average is computed. The program will generate a random joint probability distribution P^* of binary variables. Using this distribution each algorithm will construct a DMN. A new joint probability distribution, P_μ,is attained from the DMN and the KL-divergence, $D_{KL}(P^*||P_\mu)$, is calculated. The process of creating random joint probability distributions, constructing a DMN for each algorithm, and computing K-L divergence continues for i iterations. Next, the average K-L divergence for each algorithm is computed. Once the average K-L divergence is computed for each k, the experiment is complete.

The experiments that measure the efficiency of each algorithm compute the average execution time of each algorithm for different values of n. These experiments were conducted as follows. The algorithm to test is selected. Let i be the number of iterations the algorithm executes before an average is computed. The program will generate a random joint probability distribution of binary variables and the algorithm will construct a DMN from this distribution. The execution time of the algorithm will be recorded (Using java.util.Calendar class). This occurs for i iterations and, then, an average execution time for the algorithm is computed. Once the average execution time is computed for each value of n, another algorithm is selected. When all necessary algorithms are selected, the experiment is complete.

A joint probability distribution is generated by constructing a table of all possible vectors and assigning each one a weight. Only binary variables will be in the distribution, thus there will be 2^n vectors for n random variables. Let r_i be a pseudo-random number between 1 and 4 that corresponds to a vector, X_i. The weight of X_i is determined by computing 2^{r_i}. The weights of all vectors are normalized and a joint probability distribution is created as a result.

7. Experiment Results

7.1 Comparison of Accuracy between the EGA, FSA-K, and the HGA

The experiments were conducted on a PC running Windows XP and using Eclipse as an environment. Eclipse was the only application running. The graphs in this section display the results of different experiments conducted to compare either the accuracy or efficiency of the algorithms. The algorithm EGA refers to the Edgewise Greedy Algorithm without the local search procedure, and the EGA+L refers to the Edgewise Greedy Algorithm with the local search procedure. Comparing these two algorithms provide information on the effects of the local search.

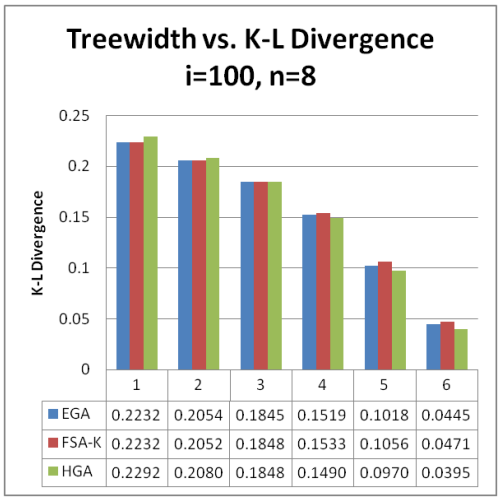


Figure 7.1

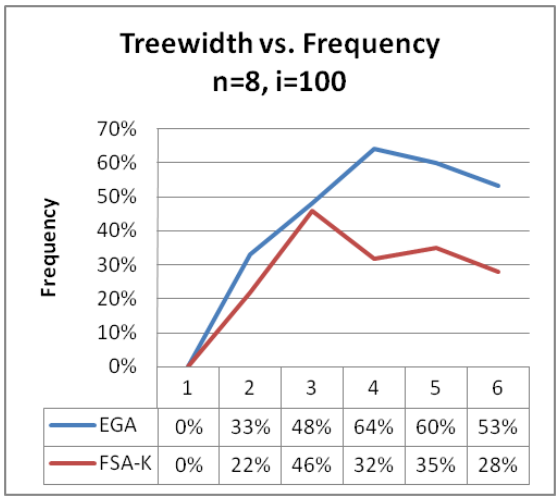


Figure 7.2

Figure 7.1 compares the accuracy of EGA, FSA-K, and HGA. EGA and FSA-K have similar accuracies, but notice that EGA performs slightly better than FSA-K overall. HGA has less accuracy for treewidths 1 and 2 in comparison to the other two algorithms, but has greater accuracy for treewidths 3, 4, and 5. Figure 7.2 shows the frequency that either the EGA or the FSA produced a model with better accuracy than the other. The advantage of EGA in comparison to FSA-K is more apparent in this figure.

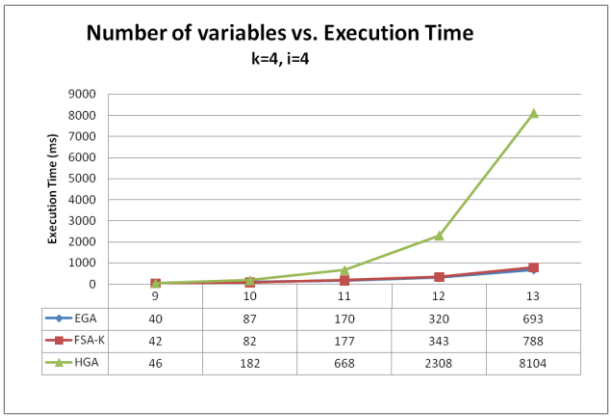


Figure 7.3

7.2 Comparison of Efficiency Between EGA, FSA-K, and HGA

Figure 7.3 compares the efficiency of three algorithms. The data shows that the EGA executes the fastest on average. The FSA-K requires a similar amount of time, and the HGA performs the slowest. The EGA adds a restriction at each step of the FSA: a new edge cannot be added if it creates a clique with a size greater than the current limit. This restriction reduces the number of edges eligible for addition to the graph, and, as a result may reduce the number of entropies to calculate. The restriction, however, also increases the execution time. The experiment indicates that the calculations for restricting edges cost less execution time than the calculations for considering those edges that would not have been eligible. This a reason that the EGA required less execution time than the FSA-K. HGA's execution time is growing at a much faster rate than the other two algorithms. In the HGA, the entropy of all subsets of size $k+1$ from a set of size n must be calculated; thus, its execution time is expected to grow at a faster rate as n increases than the execution time of the EGA and FSA-K. Each algorithm's execution time grew at an exponential rate because the program requires each algorithm to calculate entropy from a probability table whose size is doubling for every increase in n. If the empirical distribution P^* is given in the form of a training data set instead of a probability distribution table, then this exponential growth with n would not be observed.

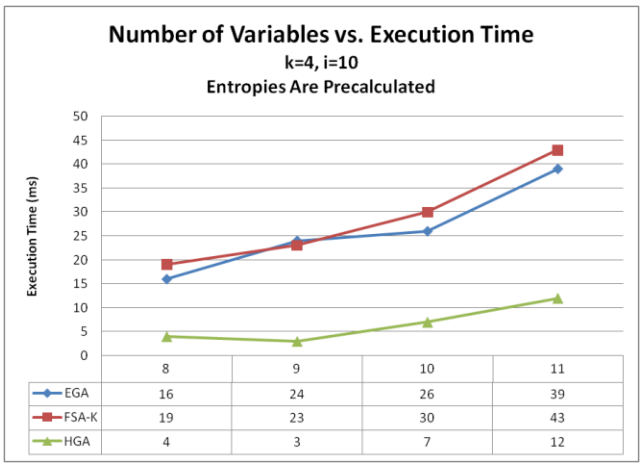


Figure 7.4

Figure 7.4 shows the how execution time is affected by increases in n when entropies between all subsets of variables are calculated before each algorithm executes. Comparing this data with that of Figure 7.3 shows that the bulk of the execution time is spent calculating entropies between variables. Figure 7.4 shows that the HGA is very efficient when entropy is precalculated; however, the rate at which the execution time of the HGA increases with increments in n may be greater than the EGA and FSA-K. Figure 4, however, does not sufficiently capture whether or not this is true. Comparing execution time of the algorithms when $n = 9$ and $n = 10$, the EGA and the FSA-K increase execution time by about 50 percent and the HGA doubles it; similarly, when $n = 10$ and $n = 11$ the EGA and FSA-K increase by about 50 percent and the HGA doubles. Data for values when $n > 11$ would provide valuable insight about the algorithms' efficiency when entropies are precalculated, but the computational capabilities of the PC used in the experiments were limited. From our knowledge of the complexity of each algorithm, however, we can predict whether or not HGA will exceed the computation time for greater values of n. Because entropies are precalculated, E is constant. Moreover, the value of k is fixed at 4. Therefore, the complexity of the algorithms EGA, FSA, and HGA would be $O(n^3)$, $O(n^3)$, and $O\left(\binom{n}{5}\right)$, respectively. $O\left(\binom{n}{5}\right)$ is equivalent to $O(n^5)$, and, thus, the HGA is expected grow at a faster rate with increase in n even when entropies are precalculated. Figure 7.4 also shows that the EGA was slightly more efficient than the FSA. The analysis of Figure 7.3 reasoned that the EGA was more efficient because it calculated fewer entropies. Here, however, entropies are precalculated and the EGA is still more efficient than the FSA. The execution times are too close to make any conclusions between the two except that the restriction that the EGA adds to the FSA-K is almost negligible in execution time—since the EGA appears to be at least equivalent in efficiency as shown by Figure 7.4.

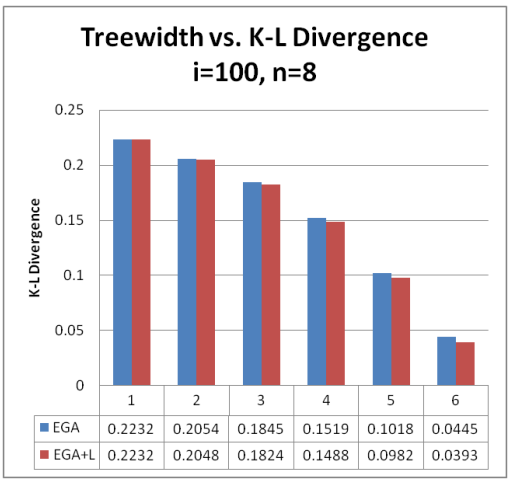


Figure 7.5

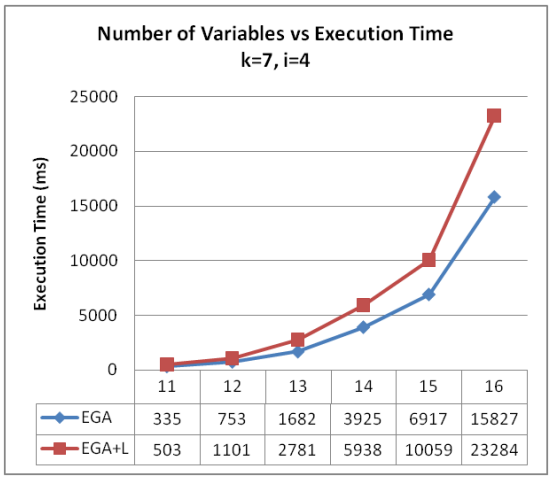


Figure 7.6

7.3 Analysis of the Local Search Procedure

Figure 7.5 compares the efficiency of Edgewise Greedy Algorithm without the local search procedure, EGA, and the Edgewise Greedy Algorithm with the local search procedure, EGA+L. The data shows that the local search procedure improves the accuracy of the EGA, and that it is more effective for higher treewidths than for lower treewidths. This increase in accuracy comes with a cost in efficiency. Figure 7.6 shows that the EGA performs slower. The rate, however, at which both algorithms' execution time grows with increases in n appears to be about the same: the local search procedure appears to increase the execution time of the EGA on average by 50 percent.

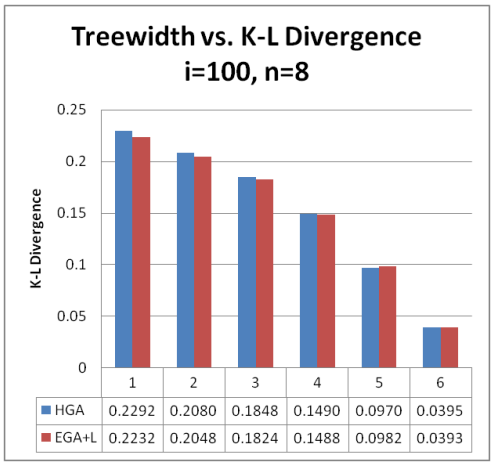


Figure 7.7

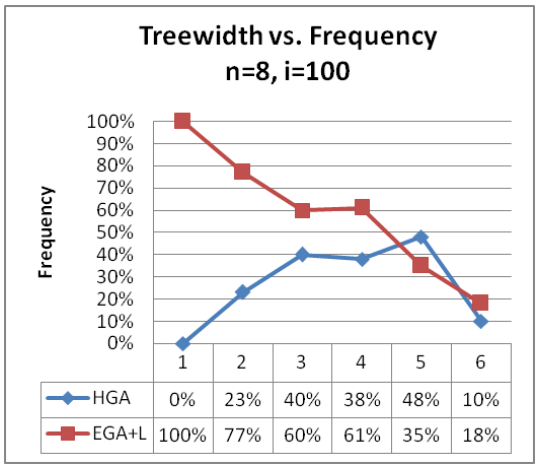


Figure 7.8

7.4 Comparison of the EGA+L and HGA

Figure 7.6 compares the accuracy of the EGA+L and the HGA. The accuracy of the two appear very similar, but the table shows that EGA+L performs slightly better. The advantage of EGA+L is conveyed in Figure 7.7 which shows the frequency that each algorithm generated a DMN with better accuracy than the other. It is clear that the EGA+L will more likely generate a model with better accuracy especially when the treewidth is low. In addition, the EGA+L requires less time to execute on average than the HGA; Figure 7.8 shows that the EGA+L requires about 50 percent more execution time than EGA, and Figure 7.3 shows that HGA is significantly less efficient than EGA. The experiments show that the EGA+L is both more accurate and more efficient than the HGA.

8. Conclusions and Future Work

Experiments were conducted that compared three greedy algorithms for learning Markov networks of treewidth k: the Edgewise Greedy Algorithm, the FSA-K—an applications of the Forward Selection algorithm—and Malvestuto's Hyperedge Greedy Algorithm. The data shows the following. All three algorithms produce models with similar approximation accuracies. The EGA is the most efficient and the HGA is significantly the least efficient. The FSA and EGA have similar approximation accuracies and have similar efficiency, but the EGA performs better in both aspects. The HGA has the best accuracy on average for treewidths that are higher than one-half of n in comparison to the other two, but has the least accuracy on average for treewidths that are lower than one-half of n. Experiments were conducted that measured the accuracy and the efficiency of the EGA coupled with a local search procedure, EGA+L. The data shows that the local search procedure improves the accuracy of the EGA—the improvement is greater for higher treewidths than for lower treewidths—but at the cost of increasing the execution time of the algorithm by 50 percent. When comparing the EGA+L to the HGA, the data shows that the EGA+L is both more efficient and more accurate.

The experiments conducted were limited by the computational power of the PC running the program. Producing an optimal model of a Markov network of treewidth k that minimized K-L divergence was not an option because of this limitation. How often an algorithm produces an optimal model and how close an algorithm comes to an optimal model are two measurements that would provide valuable insight on the accuracy of each algorithm. Conducting experiments that used more variables in the probability distribution would provide better data for comparing both the accuracy and efficiency of the algorithms. In addition, constructing a three-dimensional graph from data that calculated execution time or accuracy from two independent variables—n and k—would give insight on which treewidths are ideal in terms of accuracy or efficiency in relation to n. Any of these ideas would be manageable with the use of more sophisticated technology.

The probability distributions of the experiments were all generated the same way as described in Chapter 6. There are other ways of constructing probability tables, which may affect how each algorithm performs in terms of accuracy. Moreover, comparing the algorithms using probability distributions from real world applications would be more meaningful.

References

[1] H.L. Bodlaender, T. Kloks, and D. Kratsch (1995). Treewidth and pathwidth of permutation graphs. *SIAM Journal on Discrete Mathematics* **8**(4), 606-616.

[2] T. M. Cover and J. A. Thomas (1991). *Elements of Information Theory*. New York: John Wiley & Sons.

[3] G. T. Dean (2002). Graphical Models for high dimensional density estimation. B. S. Honours thesis, Australian National University. Citeseer.ist.psu.edu/562940.html

[4] A. Deshpande, M. Garofalakis, and M. I. Jordan (2001). Efficient stepwise selection in decomposable models. In Proceedings *of the 17th Annual Conference on Uncertainty in Artificial Intelligence* (UAI-01), 128-135. San Francisco, CA: Morgan Kaufmann.

[5] G. Ding, R. Lax, J. Chen, P. Chen, and B. Marx (2007). Comparison of Greedy Strategies for Learning Markov Networks of Tree Width k. In *Proceedings of the 2007 Int. Conference on Machine Learning: Models, Techonologies, and Applications* (MLATA07), Las Vegas, 294-300.

[6] M. Hazewinkel, "Kullback–Leibler information," in The Encyclopedia of Mathematics. [Online]. Available: Encyclopedia of Mathematics, http://eom.springer.de/. [Accessed: July 21, 2008].

[7] V. P. Kozyrev, "Graph" in The Encyclopedia of Mathematics. [Online]. Available: Encyclopedia of Mathematics, http://eom.springer.de/. [Accessed: July 21, 2008].

[8] F. Malvestuto (1991). Approximating discrete probability distributions with decomposable models. *IEEE Transactions on Systems, Man, and Cybernetics* **21**(5), 1287-1294.

[9] J. Naor, M. Naor, and A. Schaffer (1987). Fast parallel algorithms for chordal graphs. *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, New York, 355-364.

[10] K. Nunez, G. Ding, R. Lax, J. Chen, P. Chen, and B. Marx (2008). Empirical Comparison of Greedy Strategies for Learning Markov Networks of Treewidth k. *Submitted*.

[11] J. Pearl (1988). *Probabilistic Reasoning in Intelligent Systems*. San Mateo, CA: Morgan Kaufmann.

[12] S. Ross (2006). *A First Course in Probability*. Upper Saddle River: Pearson Education, Inc.