Probabilistic Graphical Markov Model Learning: An Adaptive Strategy

Elva Diaz[1], Eunice Ponce-de-Leon[1], Pedro Larrañaga[2], and Concha Bielza[2]

[1] Computer Science Department, Autonomous University of Aguascalientes, Ave. Universidad 940, C.P. 20100 Aguascalientes, Mexico
elva.diaz@itesm.mx,
eponce@correo.uaa.mx
[2] Department of Artificial Intelligence
Polytechnic University of Madrid 28660 Boadilla del Monte, Madrid, Spain
pedro.larranaga@fi.upm.es,
mcbielza@fi.upm.es

Abstract. In this paper an adaptive strategy to learn graphical Markov models is proposed to construct two algorithms. A statistical model complexity index ($SMCI$) is defined and used to classify models in complexity classes, sparse, medium and dense. The first step of both algorithms is to fit a tree using the Chow and Liu algorithm. The second step begins calculating $SMCI$ and using it to evaluate an index ($EMUBI$) to predict the edges to add to the model. The first algorithm adds the predicted edges and stop, and the second, decides to add an edge when the fitting improves. The two algorithms are compared by an experimental design using models of different complexity classes. The samples to test the models are generated by a random sampler (MSRS). For the sparse class both algorithms obtain always the correct model. For the other two classes, efficiency of the algorithms is sensible to complexity.

Keywords: Graphical Markov Model, Chow and Liu Algorithm, Model Learning, Entropy, Complexity.

1 Introduction

The problem of learning probabilistic graphical Markov models has two challenging tasks: learning the structure, and learning the parameters. Learning the structure involves a combinatorial search through the space of the graphical model class. Learning the parameters involves calculating marginal distributions that is a $\#P$ - complete problem [10]. For the case of the tree structures the well known Chow and Liu (CL) algorithm [4] is a polynomial-time solution. Chow and Liu constructed a polynomial-time algorithm to learn an approximate tree to a binary sample data. The CL algorithm obtains the maximum weight spanning tree using the Kruskal algorithm [11] and the mutual information values for the random variables. Chow and Liu showed that the maximum-weight spanning tree also maximizes the likelihood only in the class of tree model distributions. For

A. Hernández Aguirre et al. (Eds.): MICAI 2009, LNAI 5845, pp. 225–236, 2009.

suitably decomposable and sparse graphs, the junction tree algorithm provides a practical solution to the general problem of computing likelihoods. Unfortunately, many graphical models of practical interest are not decomposable and suitably sparse [10].

The learning problem can be defined as an optimization problem. To define an optimization problem, it is necessary to specify both a scoring function to be optimized, and a constraint set (the space of solutions) over which the optimization takes place. The space of solutions is the set of all probability distributions, characterized by a set of parameters Θ that must be estimated. The constraint set can be defined by a graph G, and in this case the graphical Markov model is represented by $M = (P(x, \Theta), G)$, where x is a vector of realization of random variables with joint probability distribution $P(x, \Theta)$, and G is an undirected graph representing the topological structure.

In this paper two algorithms are presented, that perform a graphical Markov model learning using the following adaptive strategy. In the first step, both algorithms fit a tree to the data using CL algorithm. In the second step, both algorithms calculate the edge missing upper bound index ($EMUBI$) that is used to estimate the maximum number of edges that can be added to the tree. This number of edges depends on the statistical model complexity index ($SMCI$) that adapts the algorithm performance to the information contained in the data sample. In the third step, the two algorithms differ. The first algorithm adds the edges proposed based on $EMUBI$ and stop. The second algorithm decides to add an edge when the fitting improves, and stop when has tested all the proposed edges.

The content of the paper is the following. In section 2 the complexity, the entropy and the model structure are defined and described. In section 3, the statistical model complexity index ($SMCI$) is defined and used to classified models in complexity classes. A graph similarity index (GSI) is defined too. In section 4 the two algorithms are constructed based on the indexes defined in section 3. In section 5 an experimental design is described. Samples generated by a random sampler introduced in [8] are employed. The results are presented and analyzed in the same section. In section 6 the conclusion and future work are presented. The appendix contains the algorithms performance description tables.

2 Complexity, Entropy and Model Structure

Learning from data could be seen as discovering constraints capable of leaving the noise in observed data out. The strength of the constraints can be encoded using a model structure. The model structure characterizes the constraints over the data, and the complexity of a model could be assessed thought the amount of information expressed by the model with respect to the totality of information contained in the data. Searching for a short program might be a promising way to discriminate a model from noise.

2.1 Entropy and Assessing Complexity

Most of the complexity measures are conceptually based on the Kolmogorov-Chaitin KC-complexity [14] that is defined as the length *K(m)* of the shortest binary program p that produces a binary string m

$$K(m) = \min\{l(p) : m = C_{UT}(p)\} \tag{1}$$

where $l(p)$ is the length of the program p , $C_{UT}(p)$ is the implementation of the program p in a Universal Turing Machine. As is known the KC-complexity is not computable [14].

It is possible to approximately assess the complexity using the entropy H, as was proved in [16] cited by [1]. Let $\mathcal{M} = \{m_i\}$ be a set of N random binary strings, and let $P(m = m_i) = p(m_i)$, so, the expected value can be approximated as follows,

$$E(K(m)) \approx -H(m) = \sum_{i=1}^{N} p(m_i) \log p(m_i) \tag{2}$$

2.2 Log-Linear Models and Graphical Representation

To represent the interactions between discrete variables using log-linear models, let $V = \{v_1, v_2, ..., v_m\}$ be a finite set of classification criteria. For each $v_i \in V$ let I_{v_i} be the set of levels of the criterion v_i. The set of cells in the contingency table is the set $I = \prod_{v_i \in V} I_{v_i}$, and a particular cell is denoted by $i = (i_{v_i}, v_i \in V)$. Let O be an object characterized by the classification criteria V. If a set of n objects is classified according to the criteria, let $n(i)$ be the number of objects in the cell i. For $a \subseteq V$ let $n(i_a)$ be the number of objects in the marginal cell $i_a = (i_{v_i}, v_i \in a)$ obtained as the sum of the cell $n(i)$ for all such i that agree with i_a on the coordinates corresponding to a. Let $p(i)$ denote a parameter assigned to the cell i, let $p = (p(i), i \in I)$ be the parameter set, similarly $p_a(i_a)$ denote the parameter set for the marginal cell i_a and $p_a = (p_a(i_a), i_a \in I_a)$.

If a sample of n independent observations of individuals classified by the criteria V is taken, the classification counts of the n objects has a multinomial distribution:

$$P\{N(i) = n(i), i \in I\} = \binom{n}{n(1)\ n(2)\ ...\ n(2^n)} \Pi_{i \in I} p(i)^{n(i)} \tag{3}$$

Definition 1. *The* general log-linear interaction model M *is defined by two types of restrictions. First, the logarithm of* $p(i)$ *is expanded as,*

$$\log p(i) = \sum_{a \subset V} \xi_a(i_a) \ , \tag{4}$$

where ξ_a *are functions of* i *that only depend on* i *via the coordinates in* a*, i.e., through* i_a*. This class of functions* $\xi_a(i_a)$ *is known as the class of interactions. To*

have a one-to-one correspondence between the system of functions $\{\xi_a, a \subset V\}$ *and the parameter set* p*, let introduce the second group of constraints [9], [7]*

$$\forall\, b \subset a : \sum_{\{i_a : i_c = i_b\}} \xi_a(i_c) \equiv 0 \text{ for all } i_b \ . \tag{5}$$

These restrictions (5) mean that the sum over each interaction equals zero.

Definition 2. *A* hierarchical log-linear model M_H *is a log-linear interaction model where the functions* ξ_a *satisfies the following property:*

$$if\ \xi_a = 0 \text{ and } b \supseteq a \text{ then } \xi_b = 0 \ . \tag{6}$$

A hierarchical log-linear interaction model can be specified via the so-called generating class.

Definition 3. *A* generating class of a hierarchical log-linear model *is a set* $\mathcal{E} = \{V; E_1, E_2, ..., E_k\}$ *of pair-wise incomparable (w.r.t. inclusion) subsets of* V *to be interpreted as the maximal sets of permissible interactions, i.e., [9]*

$$\xi_a \equiv 0 \text{ iff there is no } E_i \in \mathcal{E} \text{ with } a \subseteq E_i \ . \tag{7}$$

Let denote a graph G by its maximal cliques $G = \{V; g_1, g_2, ..., g_k\}$ where for all $g_i \subseteq V$, they are pairwise incomparable (w.r.t. inclusion).

Definition 4. *If the subsets of the generating class* $\mathcal{E}$ *of a hierarchical log-linear model are in a one to one correspondence to the maximal cliques of a graph, the model receives the name of* graphical Markov model *and is denoted by* M_G *[13].*

From now on, the graphical Markov model will be denoted by M to simplify the notation.

3 Information and Model Complexity

Let $X = (X_1, X_2, ..., X_v)$ be a binary random vector with a multinomial distribution. Let $P(x)$ be a probability distributions of the random vector X. Let take a sample of size n, and let $L(\widehat{m}_n^M \mid x)$ be the likelihood statistic for the sample where $\widehat{m}_n^M = \left\{\widehat{m}_i^M\right\}$ is the *maximum likelihood estimate* assuming that the true model is M.

Definition 5. *The* K-L divergence from the probability model P to the data x *is given by the Kullback-Leibler information [2]:*

$$G^2(M, x) = \log(L(\widehat{m}_n^M(x)) = -2 \sum_{i=1}^{2^v} x_i \log_2(\frac{\widehat{m}_i^M}{x_i}) \ . \tag{8}$$

The K-L divergence is also known as relative entropy, and can be interpreted as the amount of information in the sample x not explained by the model M, or the deviation from the model M to the data x. To obtain the $\widehat{m}_n^M$ the IPS algorithm can be employed. This algorithm is exponential, but it converges to the parameters of the model M [5][13].

3.1 Statistical Model Complexity Index (*SMCI*)

Using a short program might be a promising way to discriminate a model from noise (see 2.1). But, how much from the information contained in the sample must be saved in the model? A strategy can be to capture first the ease to catch information, and later, gain as much as possible, but not so much as to include the noise as part of the model. Given the number of nodes of a connected graph, the graph with the minimum number of edges is a tree. It is known that a graphical model that can be exactly learned in polynomial time is one whose interaction structure is a tree [4]. These two properties sustain the idea that a graphical model complexity must be a minimum for a model represented by a tree, because a tree can be learned by a polynomial time algorithm.

It is known that the uniform distribution has the maximum entropy and at the same time gives no information about the interaction structure of a model [15]. The model representing the uniform distribution is denoted by M_0, and the model represented by a tree is denoted by M_T. If a sample is generated by a model M containing more edges than a tree, the information about the model M contained in this sample and not explained, when the model structure is approximated by a tree, may be assessed by the following index.

Definition 6. *Let x be a sample generated by a model M. The* statistical model complexity index ($SMCI$) of the model M *is defined by the quantitative expression:*

$$SMCI(M, x \mid M_T) = \frac{G^2(M_T, x) - G^2(M, x)}{G^2(M_0, x)} \,. \tag{9}$$

This is the amount of information contained in the sample x and not explained by the model M with respect to the information not explained by the tree model M_T. This $SMCI$ index is standardized by the information not explained by the uniform model M_0. This last information is the total amount of information contained in the sample, because the uniform model does not contain any information.

Proposition 1. *$SMCI(M, x \mid M_T)$ has the following properties:*

(a) $0 \leq SMCI(M, x \mid M_T) \leq 1$.

(b) If a sample is generated by a tree model M_T, the value of the complexity index $SMCI(M, x \mid M_T)$ is equal zero.

(c) If a sample x is generated by a model M with the same number of vertices but with more edges than a tree model, then the complexity index $SMCI(M, x \mid M_T)$ is greater than zero.

3.2 Sparse, Medium and Dense Model Classes

The model complexity index is used to define three types of model class complexities. The idea is that a model to describe a data sample must be as complex as the information contained in the data.

Definition 7. *Let M be a model and x be a sample generated by M, when the model complexity index $SMCI(M, x \mid M_T) \leq 0.33$, M is classified in the*

sparse class type, *when* $0.33 \leq SMCI(M, x \mid M_T) \leq 0.66$, *M is classified in the* medium class type *and when* $SMCI(M, x \mid M_T) \geq 0.66$, *M is classified in the* dense class type.

Theorem 1. *Let x be a sample generated by a graphical model M. Let* $\{M_i\}$ *be a succession of graphical models, such that* M_{i+1} *has one edge more than* M_i, *then*

$$\lim_{M_i \longrightarrow M} SMCI(M_i, x \mid M_T) = \frac{G^2(M_T, x)}{G^2(M_0, x)} \,. \tag{10}$$

Proof. Substituting the definition 6 in the equation (10)

$$\begin{aligned}
\lim_{M_i \longrightarrow M} SMCI(M_i, x \mid M_T) &= \lim_{M_i \longrightarrow M} \frac{G^2(M_T, x) - G^2(M_i, x)}{G^2(M_0, x)} \,, \\
&= \lim_{M_i \longrightarrow M} \frac{G^2(M_T, x)}{G^2(M_0, x)} - \lim_{M_i \longrightarrow M} \frac{G^2(M_i, x)}{G^2(M_0, x)} \,, \\
&= \frac{G^2(M_T, x)}{G^2(M_0, x)} - \lim_{M_i \longrightarrow M} \frac{G^2(M_i, x)}{G^2(M_0, x)} \,, \\
&= \frac{G^2(M_T, x)}{G^2(M_0, x)} - \frac{1}{G^2(M_0, x)} \lim_{M_i \longrightarrow M} G^2(M_i, x) \,.
\end{aligned}$$

Let M_S be the saturated model, then $G^2(M_S, x) = 0 = L$ such that L is the under bound. Every monotone decreasing bounded sequence is convergent [12], then

$$\forall\, \epsilon \geq 0 \text{ and } M_j \in \{M_i\} \text{ such that } \forall\, i \geq j, \left|G^2(M_i, x) - L\right| < \epsilon \,, \tag{11}$$

then

$$\lim_{M_i \longrightarrow M} G^2(M_i, x) = 0 \text{ and the theorem is proved.} \tag{12}$$

□

Let denotes

$$\frac{G^2(M_i, x)}{G^2(M_0, x)} = IP(M_i, x) \,. \tag{13}$$

$IP(M_i, x)$ is the information proportion contained in the data sample x and not explained by the model M_i. This will be used in the Algorithm 3 later.

3.3 Graphs Similarity Index (*GSI*)

Definition 8. *Given two graphs* G_1 *and* G_2 *a* graph similarity index $GSI(G_1, G_2)$ *between* G_1 *and* G_2 *is given by the number of common edges divided by the number of edges in the union of the two graphs [6]. Denote by* E_1 *the set of edges from* G_1 *and by* E_2 *the set of edges from* G_2, *then the similarity index is given by:*

$$GSI(G_1, G_2) = \frac{|E_1 \cap E_2|}{|E_1 \cup E_2|} \,, \tag{14}$$

where $|C|$ *denotes the number of elements in the set C.*

4 Graphical Markov Model Learning: An Adaptive Strategy

In this section two adaptive algorithms are introduced to learn a model from a sample data. The name adaptive for these algorithms is because their design allows that the performance depends on the complexity of the model to fit. As a first part of the algorithms a maximum expansion tree is fitted to the data using the mutual information as a measure of interaction between variables taken two by two and the Kruskal algorithm [4] is used to fit a maximum spanning tree. A prediction indicator $EMUBI$ will be defined, calculated and used, to predict the number of edges that must be added to the tree to attain an approximation to the correct model.

4.1 Edge Missing Upper Bound Index (*EMUBI*)

The most important step in both algorithms consists of a prediction of the number of edges that must be added to the maximum spanning tree to fit the model M to the data sample.

Definition 9. *Let G be the graph of the model M, and let v be the number of nodes, let $MNE(v)$ be the maximum number of edges. The* edge missing upper bound index $(EMUBI)$ *is defined by:*

$$EMUBI(M, x \mid M_T) = (MNE(v) - v + 1)SMCI(M, x \mid M_T)\,. \tag{15}$$

If the data sample were generated from a tree model, the edge missing upper bound index would be almost zero. The maximum number of edges that could be added to model is proportional to the amount of information contained in the sample x and not explained by the tree model. $EMUBI$ will be used to predict the number of missing edges.

4.2 Chow and Liu Maximum Spanning Tree

The CL algorithm (See Algorithm 1) obtains the maximum weight spanning tree using the Kruskal algorithm [11] and the mutual information values for the random variables.

The mutual information measure $I_{X_iX_j}$ for all $X_i, X_j \in X$ are defined as:

$$I_{X_iX_j} = I(X_i, X_j) = \sum_{x_i,x_j} P(x_i, x_j) \log \frac{P(x_i, x_j)}{P(x_i)P(x_j)}\,. \tag{16}$$

4.3 Extended Tree Adaptively Learning Algorithm (ETreeAL)

In this section the CL algorithm already described is extended. A number of edges is added based on $EMUBI$. The number of edges added, try to adapt the complexity of the learning model to the data complexity, as described in the Algorithm 2. This algorithm has a parameter τ that play a regularization role to avoid overfitting the model to the sample.

Algorithm 1. CL algorithm

Input: Distribution P over the random vector $X = (X_1, X_2, ..., X_v)$
1. Compute marginal distributions P_{X_i} , $P_{X_iX_j}$, for all $X_i, X_j \in X$
2. Compute mutual information values $I_{X_iX_j}$ for all $X_i, X_j \in X$
3. Order the values from high to low (w.r.t.) mutual information
4. Obtain the maximum weight spanning tree $M_T(CL)$ by the Kruskal algorithm [11]
Output: The maximum spanning tree $M_T(CL)$

Algorithm 2. The extended tree adaptively learning algorithm (ETreeAL)

Input: Distribution P over the random vector $X = (X_1, X_2, ..., X_v)$
1. Call Algorithm 1: CL Algorithm
2. Calculate the edge missing upper bound prediction index ($EMUBI(M, x \mid M_T(CL))$)
3. Add to $M_T(CL)$, τ per cent from missing edges in the order of the mutual information values
Output: Extended $M_{EXT}(CL)$

4.4 Global Fitting Extended Tree Adaptive Learning Algorithm (GETreeAL)

In this section the extended CL algorithm already described is enriched with a global fitting assessing, using the Kullback-Leibler divergence at each step to decide if each new edge (taken in order from the list) is added or not, after the maximum spanning tree is obtained. So the adaptation to the complexity proceed step by step, as can be seen in the Algorithm 3.

Algorithm 3. The global fitting extended tree algorithm (GETreeAL)

Input: Distribution P over the random vector $X = (X_1, X_2, ..., X_v)$
1. Call Algorithm 1: CL Algorithm
2. Calculate the $G^2(M_0, x)$ K-L divergence
3. Calculate the edge missing upper bound prediction index $EMUBI(M, x \mid M_T(CL))$
Repeat
    4. Select an edge in the order of the mutual information values
    5. $E \leftarrow E - 1$
    6. Add to the already obtained graph G_{i-1}
    7. Calculate the $G^2(M_i, x)$ K-L divergence
    8. **If** $G^2(M_i, x) < G^2(M_{i-1}, x)$ retain M_i
       Else retain M_{i-1}
Until $IP(M_i, x) < \epsilon$ for some $\epsilon > 0$
Output: Global extended $M_{GEXT}(CL)$

5 Experiment Design, Samples and Analysis

To perform a comparative factorial experiment of the two presented algorithms, samples are generated using the random sampler algorithm (MSRS) introduced

and described in [8]. Samples of models of 16, 18 and 20 vertices and complexities sparse, medium and dense are generated (See Table 1, Appendix). A total of 9 models are used to perform the experiment. Each combination of factors is replicated 6 times for a total of 54 samples that are used with each algorithm. For ETreeAL algorithm the parameter $\tau = 0.67$. The result variables are the running time and the measure of similarity. A simple covariance analysis is performed for each result variable. The following results are obtained.

The algorithms and the experimental design were implemented in ANSI C. The result tables are in the appendix. For sparse models both algorithms obtain exactly the original models as solutions, because in this case only the CL algorithm is employed. So, they are taken out of the analysis. A covariance analysis was performed where the covariables are complexity of the model (sparse, medium, dense), and number of nodes (16, 18, 20) of the models, and the factor is the algorithm type (ETreeAL, GETreeAL). The response variables are similarity and running time. The factor "Algorithm", and the covariables "Complexity" and "Number of nodes" are statistically significant for the response variables "Run time" (in seconds) and "Similarity" (See Similarity Index, Definition 8) (See Tables 2-3, Appendix). The run time of ETreeAL is approximately 10 times faster than the run time for the GETreeAL (See Table 4, Appendix). The ETreeAL algorithm has a better performance in the case of the similarity index for dense models than medium models. For Example, for 20 nodes ETreeAL obtains complexity 3, similarity .672 and complexity 2, similarity .436 and this relationship is reversed for GETreeAL algorithm. For Example, for 20 nodes GETreeAL obtains complexity 3, similarity .992 and complexity 2, similarity 1.000 (See Table 4, Appendix).

6 Conclusions and Future Work

In this paper it is showed that it is possible to know in advance how many edges are to be added to a model to capture the information necessary to have a model whose complexity represents the data sample. Detecting the model complexity before deciding how many edges to add, permits an adaptive strategy helping the algorithms to be as efficient as possible depending on the model complexity. The learning step of the GETreeAL needs the sample evaluation at each step, to decide if an edge is added or not. It can be concluded that the ETreeAL algorithm, can be used for the sparse models class. The ETreeAL algorithm outperforms 10 times the GETreeAL in running time but GETreeAL outperforms ETreeAL 44 per cent in similarity index. The GETreeAL algorithm can be scaled for more than 20 nodes substituting the IPS algorithm by an aproximate estimate of the $\widehat{m}_n^M$.

Acknowledgments. This work is supported by Project PII 08-3, Autonomous University of Aguascalientes, Aguascalientes, Mexico. The reviewers of the paper are greatly acknowledged for their constructive comments.

References

1. Adami, C.N., Cerf, J.: Physical Complexity of Symbolic Sequences. Physica D 137, 62–69 (2000)
2. Akaike, H.: A New Look at the Statistical Model Identification. IEEE Transactions on Automatic Control 19, 716–723 (1974)
3. Chickering, M.C., Heckerman, D., Meck, C.: Large-Sample Learning of Bayesian Network is NP-Hard. Journal of Machine Learning Research 5, 1287–1330 (2004)
4. Chow, C.K., Liu, W.: Approximating Discrete Probability Distributions with Dependency Trees. IEEE Trans. Inf. Theory IT-14(3), 462–467 (1968)
5. Deming, W.E., Stephan, F.F.: On a Least Squares Adjustment of a Sampled Frequency Table When the Expected Marginal Totals are Known. Ann. of Math. Static. 11, 427–444 (1940)
6. Diaz, E.: Metaheurísticas Híbridas para el Aprendizaje de Modelos Gráficos Markovianos y Aplicaciones. Tesis para optar por el Grado de Doctor en Ciencias de la Computación. Universidad Autónoma de Aguascalientes, Ags., Mexico (2008)
7. Diaz, E., Ponce de Leon, E.: Discrete Markov model selection by a genetic algorithm. In: Sossa-Azuela, J.H., Aguilar-IbaÑÉz, C., Alvarado-Mentado, M., Gelbukh, A. (eds.) Avances en Ciencias de la Computación e Ingeniería de Cómputo, Mexico, vol. 2, pp. 315–324 (2002)
8. Diaz, E., Ponce de Leon, E.: Markov Structure Random Sampler (MSRS) algorithm from unrestricted discrete graphic Markov models. In: Gelbukh, A., Reyes, C.A. (eds.) Proceedings of the Fifth Mexican International Conference on Artificial Intelligence, pp. 199–206. IEEE Computer Society, Mexico (2006)
9. Haberman, S.J.: The Analysis of Frequency Data. The University of Chicago Press (1974)
10. Koller, D., Freedman, N., Getoor, L., Taskar, B.: Graphical models in a nutshell in Introduction to Statistical Relational Learning. In: Getoor, L., Taskar, B. (eds.) Stanford (2007)
11. Kruskal, J.B.: On the Shortest Spanning Tree of a Graph and the Traveling Salesman Problem. Proc. Amer. Math. Soc. 7, 48–50 (1956)
12. Kuratowski, K.: Introduction to Calculus. Pergamon Press, Warsaw (1961)
13. Lauritzen, S.L.: Graphical models. Oxford University Press, USA (1996)
14. Li, M., Vitanyi, P.M.B.: An Introduction to Kolmogorov Complexity and its Applications. Springer, Heidelberg (1993)
15. MacKay, J.C.: Information Theory Inference and Learning Algorithms. Cambridge Press (2003)
16. Zvonkin, A.K., Levin, L.A.: The Complexity of Finite Objects and the Development of the Concepts of Information and Randomness by Means of the Theory of Algorithms. Russ. Math. Surv. 256, 83–124 (1970)

Appendix. Algorithm Performance Description Tables

Table 1. Graphical Markov model structures

models	cliques	edges	#cliques
medium 16	ABCD,BCDE,CDEF,DEFG,EFGH, FGHI,GHIJ,HIJK, ILJK, JKLM,KLMN,LMNO,MNOP,ANOP	45	14
dense 16	ABCDEF,BCDEFG,CDEFGH, DEFGHI,EFGHIJ, FGHIJK, GHIJKL,HIJKLM,IJKLMN, JKLMNO,KLMNOP,ALMNOP	70	12
medium 18	ABCD,BCDE,CDEF,DEFG, EFGH,FGHI,GHIJ, HIJK, IJKL,JKLM,KLMN,LMNO, MNOP, NOPQ,OPQR,APQR, ABQR	53	17
dense 18	ABCDEF,CDEFGH,EFGHIJ GHIJKL, IJKLMN,KLMNOP MNOPQR,ABOPQR	77	8
medium 20	ABCD,BCDE,CDEF,DEFG,EFGH,FGHI,GHIJ, HIJK, IJKL,JKLM,KLMN,LMNO,MNOP, NOPQ,OPQR, PQRS,QRST,ARST,ABST	59	19
dense 20	ABCDEF,BCDEFG,CDEFGH,DEFGHI, EFGHIJ,FGHIJK,GHIJKL,HIJKLM, IJKLMN, JKLMNO,KLMNOP,LMNOPQ, MNOPQR,NOPQRS,OPQRST APQRST,ABQRST	94	17

Table 2. Analysis of Variance Table, Dependent Variable= RUN TIME, R Squared = .470 (Adjusted R Squared = .447)

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	2939636.191	3	979878.730	20.130	.000
Intercept	1651429.970	1	1651429.970	33.927	.000
NODES	1503464.143	1	1503464.143	30.887	.000
COMPLEX	439566.851	1	439566.851	9.030	.004
ALGORITHM	996605.197	1	996605.197	20.474	.000
Error	3309992.351	68	48676.358		
Total	7746170.175	72			
Corrected Total	6249628.542	71			

Table 3. Analysis of Variance Table: Dependent Variable = SIMILARITY, R Squared = .770 (Adjusted R Squared = .760)

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	1.910	3	.637	75.932	.000
Intercept	.690	1	.690	82.317	.000
NODES	.172	1	.172	20.545	.000
COMPLEX	.100	1	.100	11.960	.001
ALGORITHM	1.637	1	1.637	195.290	.000
Error	.570	68	8.384E-03		
Total	52.862	72			
Corrected Total	2.480	71			

Table 4. Means: Run time and similarity by algorithm, nodes number and complexity classes

			Mean	Mean
ALGORITHM	NODES	COMPLEX	RUN TIME	SIMILARITY
ETreeAL	16	2	1.104	.730
ETreeAL	16	3	2.281	.890
ETreeAL	18	2	7.089	.656
ETreeAL	18	3	12.495	.730
ETreeAL	20	2	39.029	.436
ETreeAL	20	3	97.123	.672
GETreeAL	16	2	7.600	.979
GETreeAL	16	3	29.357	.980
GETreeAL	18	2	59.679	.994
GETreeAL	18	3	154.259	.979
GETreeAL	20	2	281.714	1.000
GETreeAL	20	3	1038.322	.992