Using Tarjan's Red Rule for Fast Dependency Tree Construction

Dan Pelleg and Andrew Moore

February 2002
CMU-CS-02-116

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We focus on the problem of efficient learning of dependency trees. It is well-known that given the pairwise mutual information coefficients, a minimum-weight spanning tree algorithm solves this problem exactly and in polynomial time. However, for large data-sets it is the construction of the correlation matrix that dominates the running time. We have developed a new spanning-tree algorithm which is capable of exploiting partial knowledge about edge weights. The partial knowledge we maintain is a probabilistic confidence interval on the coefficients, which we derive by examining just a small sample of the data. The algorithm is able to flag the need to shrink an interval, which translates to inspection of more data for the particular attribute pair. Experimental results show significant improvement in running time, without loss in accuracy of the generated trees. Interestingly, our spanning-tree algorithm is based solely on Tarjan's red-edge rule, which is generally considered a guaranteed recipe for bad performance.

This research was supported by NSF Grant #ACI0121671

Keywords: Machine learning, Bayes' networks, dependency trees, Hoeffding races, scalable data-mining

1. Introduction

Bayes' nets are widely used for data modeling. However, the problem of constructing Bayes' nets from data remains a hard one, requiring search in a super-exponential space of possible graph structures. Despite recent advances (Friedman et al., 1999), learning network structure from big data sets demands huge computational resources. We therefore turn to a simpler model, which is easier to compute while still being expressive enough to be useful. Namely, we look at dependency trees, which are belief networks that satisfy the additional constraint that each node has at most one parent. In this simple case it has been shown (Chow & Liu, 1968) that finding the tree that maximizes the data likelihood is equivalent to finding a minimum-weight spanning tree in the attribute graph, where edge weights are derived from the mutual information of the corresponding attribute pairs.

It is our intent to eventually apply the technology introduced in this paper to the full problem of Bayes Net structure search.

Once the weight matrix is constructed, executing a minimum spanning tree (MST) algorithm is fast. The time-consuming part is the population of the weight matrix, which takes time linear in the number of records in the given dataset and quadratic in the number of attributes. This becomes expensive when considering datasets with hundreds of thousands of records and hundreds of attributes.

To overcome this problem, we propose a new way of interleaving the spanning tree construction with the operations needed to compute the mutual information coefficients. We develop a new spanning-tree algorithm, based solely on Tarjan's (1983) red-edge rule. This algorithm is capable of using partial knowledge about edge weights and of signaling the need for more accurate information regarding a particular edge. The partial information we maintain is in the form of probabilistic confidence intervals on the edge weights; an interval is derived by looking at a sub-sample of the data for a particular attribute pair. Whenever the algorithm signals that a currently-known interval is too wide, we inspect more data records in order to shrink it. Once the interval is small enough, we may be able to prove that the corresponding edge is *not* a part of the tree. Whenever such an edge can be eliminated without looking at the full data-set, the work associated with the remainder of the data is saved. This is where performance is potentially gained.

We have implemented the algorithm for numeric and categorical data and tested it on real and synthetic data-sets containing hundreds of attributes and millions of records. We show experimental results of up to thirty-fold speed improvements over the traditional algorithm. In many cases the resulting tree is identical to the one produced by the naive algorithm. When it is not, they are of near-identical quality.

Use of probabilistic bounds to direct structure-search appears in (Maron & Moore, 1994) for classification and in (Moore & Lee, 1994) for model selection. In a sequence of papers, Domingos et al. have demonstrated the usefulness of this technique for decision trees (Domingos & Hulten, 2000), K-means clustering (Domingos & Hulten, 2001a), and mixtures-of-Gaussians EM (Domingos & Hulten, 2001b). In the context of dependency trees, Meila (1999a) discusses the discrete case that frequently comes up in text-mining applications, where the attributes are sparse in the sense that only a small fraction of them is true for any record. In this case it is possible to exploit the sparseness and accelerate the Chow-Liu algorithm.

Throughout the paper we use the following notation. The number of data records is R, the number of attributes n. When x is an attribute, x_i is the value it takes for the i-th record. We denote by ρ_{xy} the correlation coefficient between attributes x and y, and omit the subscript when it is clear from the context. H_x is the entropy of an attribute or an attribute set x.

2. A Slow Minimum-Spanning Tree Algorithm

We begin by describing our MST algorithm[1]. Although in its given form it can be applied to any graph, it is asymptotically slower than established algorithms (as predicted in Tarjan (1983) for all algorithms in its class). We then proceed to describe its use in the case where some edge weights are known not exactly, but rather only to lie within a given interval. In Section 4 we will show how this property of the algorithm interacts with the data-scanning step to produce an efficient dependency-tree algorithm.

In the following discussion we assume we are given a complete graph with n nodes, and the task is to find a tree connecting all of its nodes such that the total tree weight (defined to be the sum of the weights of its edges) is minimized. This problem has been extremely well studied and numerous efficient algorithms for it exist.

[1] To be precise, we will use it as a *maximum* spanning tree algorithm. The two are interchangeable, requiring just a reversal of the edge weight comparison operator.

We start with a rule to eliminate edges from consideration for the output tree. Following Tarjan (1983), we state the so-called "red-edge" rule:

Theorem 1: The heaviest edge in any cycle in the graph is not part of the minimum spanning tree.

Traditionally, MST algorithms use this rule in conjunction with a greedy "blue-edge" rule, which chooses edges for inclusion in the tree. In contrast, we will repeatedly use the red-edge rule until all but $n-1$ edges have been eliminated. The proof this results in a minimum-spanning tree follows from Tarjan (1983).

Let E be the original set of edges. Denote by L the set of edges that have already been eliminated, and let $\bar{L} = E \setminus L$. As a way to guide our search for edges to eliminate we maintain the following invariant:

Invariant 2: At any point there is a spanning tree T, which is composed of edges in $\bar{L}$.

In each step, we arbitrarily choose some edge e in $\bar{L} \setminus T$ and try to eliminate it using the red-edge rule. Let P be the path in T between e's endpoints. The cycle we will apply the red-edge rule to will be composed of e and P. It is clear we only need to compare e with the heaviest edge in P. If e is heavier, we can eliminate it by the red-edge rule. However, if it is lighter, then we can eliminate the tree edge by the same rule. We do so and add e to the tree to preserve Invariant 2. The algorithm, which we call Minimum Incremental Spanning Tree (MIST), is listed in Figure 1.

The MIST algorithm can be applied directly to a graph where the edge weights are known exactly. And like many other MST algorithms, it can also be used in the case where just the relative order of the edge weights is given. Now imagine a different setup, where edge weights are not given, and instead an oracle exists, who knows the exact values of the edge weights. When asked about the relative order of two edges, it may either respond with the correct answer, or it may give an inconclusive answer. Furthermore, a constant fee is charged for each query. In this setup, MIST is still suited for finding a spanning tree while minimizing the number of queries issued. In step 2, we go to the oracle to determine the order. If the answer is conclusive, the algorithm proceeds as described. Otherwise, it just ignores the "if" clause altogether and iterates (possibly with a different edge e).

For the moment, this setup may seem contrived, but in Section 4, we go back to the MIST algorithm and put it in a context very similar to the one described here.

1. $T \leftarrow$ an arbitrary spanning set of $n-1$ edges.

   $L \leftarrow$ empty set.

2. While $|\bar{L}| > n-1$ do:

   Pick an arbitrary edge $e \in \bar{L} \setminus T$.
   Let e' be the heaviest edge on the path in T between the endpoints of e.
   If e is heavier than e':
   $L \leftarrow L \cup \{e\}$
   otherwise:
   $T \leftarrow T \cup \{e\} \setminus \{e'\}$
   $L \leftarrow L \cup \{e'\}$

3. Output T.

Figure 1. The MIST algorithm. At each step of the iteration, T contains the current "draft" tree. L contains the set of edges that have been proven to *not* be in the MST and so $\bar{L}$ contains the set of edges that still have some chance of being in the MST. T never contains an edge in L.

3. Probabilistic Bounds on Mutual Information

We now concentrate once again on the specific problem of determining the mutual information between a pair of attributes. We show how to compute it given the complete data, and how to derive probabilistic confidence intervals for it, given just a sample of the data.

As shown in Reza (1994), the mutual information for numeric attributes X and Y is:

$$I(X;Y) = -\frac{1}{2}\ln(1-\rho^2)$$

where the correlation coefficient $\rho = \rho_{XY} =$

$$\frac{\sum_{i=1}^{R}\left((x_i-\bar{x})(y_i-\bar{y})\right)}{\hat{\sigma}_X^2\hat{\sigma}_Y^2}$$

with $\bar{x}, \bar{y}, \hat{\sigma}_X^2$ and $\hat{\sigma}_Y^2$ being the sample means and variances for attributes X and Y.

Since the log function is monotonic, $I(X;Y)$ must be monotonic in $|\rho|$. This is a sufficient condition for the use of $|\rho|$ as the edge weight in a MST algorithm. Consequently, the sample correlation can be used in a straightforward manner when the complete data is available. Now consider the case where just a sample of the data has been observed.

Let x and y be two data attributes. We are trying to estimate $\sum_{i=1}^{R} x_i \cdot y_i$ given the partial sum $\sum_{i=1}^{r} x_i \cdot y_i$

for some $r < R$. To derive a confidence interval, we use the Central Limit Theorem [2]. It states that given samples of the random variable Z (where for our purposes $Z_i = x_i \cdot y_i$), the sum $\sum_i Z_i$ can be approximated by a Normal distribution with mean and variance closely related to the distribution mean and variance. Furthermore, for large samples, the sample mean and variance can be substituted for the unknown distribution parameters. Note in particular that the central limit theorem *does not require us to make any assumption about the Gaussianity of Z*. We thus can derive a two-sided confidence interval for $\sum_i Z_i = \sum_i x_i \cdot y_i$ with probability $1 - \delta$ for some user-specified δ ,typically 1%. Given this interval, computing an interval for ρ is straightforward.

In the case of categorical data, we follow Meila (1999b) and write:

$$\begin{aligned} I(X;Y) &= H_X + H_Y - H_{XY} \\ &= \frac{1}{R}[-\mathrm{zlogz}(N_X) - \mathrm{zlogz}(N - N_X) \\ &- \mathrm{zlogz}(N_Y) - \mathrm{zlogz}(N - N_Y) \\ &+ \mathrm{zlogz}(N_{XY}) + \mathrm{zlogz}(N_X - N_{XY}) \\ &+ \mathrm{zlogz}(N_Y - N_{XY}) \\ &+ \mathrm{zlogz}(R - N_X - N_Y + N_{XY}) \\ &+ \mathrm{zlogz}(R)] \end{aligned}$$

where $\mathrm{zlogz}(z)$ is shorthand for $z \log z$ and N_z denotes the number of times an attribute or a set of attributes are observed all true. As before, N_{XY} is the quantity we are deriving a probabilistic estimate for, which we do from the counts in a sample and application of the CLT. We then evaluate $I(X;Y)$ at the endpoints. We also evaluate the function's minimum, if it happens to fall within the interval, and determine minimum and maximum values for $I(X;Y)$.

4. The Full Algorithm

As we argued, the MIST algorithm is capable of using partial information about edge weights. We have also shown how to derive confidence intervals on edge weights. We now combine the two and give an efficient dependency-tree algorithm.

We largely follow the MIST algorithm as listed in Figure 1. We initialize the tree T in the following heuristic way: first we take a small sub-sample of the data, and derive point estimates for the edge weights from it. Then we feed the point estimates to a MST algorithm and obtain a tree T. For each edge in T, we compute the exact correlation coefficient from the full data.

Throughout the algorithm we maintain the following modification of Invariant 2:

Invariant 3: At any point there is a spanning tree T, which is composed of edges in $\bar{L}$, and the tree edges have correlation coefficients computed from the full data.

This way, finding the heaviest edge on the tree path P is straightforward. When the time comes to compare a non-tree edge e and a tree edge e', we have the exact value for e', and a confidence interval for e. If the value for e' lies outside of the interval for e, we can make a decision immediately. The amount of work we save is related to how much data was used in computing the confidence interval — the rest of the data for this attribute-pair can be ignored. Now, if the weight of e' lies *within* the interval, we have a situation similar to the inconclusive oracle answers in Section 2. The price we need to pay here is looking at more data to shrink the confidence interval. Currently we double the sample size used to compute the sufficient statistics. After doing so we try to compare e and e' again (since we can do this at no additional cost). If we fail to eliminate one of the edges we iterate, possibly choosing a different edge this time.

Therefore, the sooner we can eliminate an edge, the more work we save. Also, since tree edges always have their weights computed exactly, we get no savings for edges that go into the tree (either at the initial step, or later as they eliminate tree edges), only to be eliminated by a better edge in a subsequent step. The theoretical best we can hope for is to choose the right tree edges from the sample, and to eliminate all other edges based on intervals supported by this sample. In this case the total amount of work we do is proportional to $R(n-1)$ (we assume R is sufficiently larger than n to neglect the work related to finding tree paths and so on). Compare this to $O(Rn^2)$ required by algorithms that need the full weight matrix in advance. The following section presents experimental results and compares how close we get to this theoretical limit.

Another heuristic we employ goes as follows. Consider the comparison of the path-heaviest edge to an estimate of a candidate edge. The candidate edge's confidence interval may be very small, and yet still include the point that is the heavy edge's weight. We may be able to reduce the amount of work by pretending the interval is narrower than it really is. We therefore trim the interval by a constant, parameter-

[2] One can use the weaker Hoeffding bound instead, and our implementation supports it as well, although Hoeffding is generally much less powerful.

Output: A data-record as a vector $\{X(0) \ldots X(n-1)\}$ of attributes.

- $X(0)$ is a value drawn from $N(0,1)$.
- $X(10i)$ is a value drawn from $N(X(10(i-1)),1)$.
- $X(10k+j)$ for $j \neq 0$ is drawn from $N(X(10k+j-1),j)$.

Figure 2. The data-generation algorithm

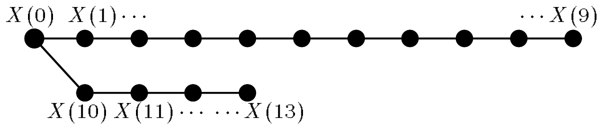

Figure 3. Structure of the generated data for 14 attributes.

ized by the user as ϵ. On further inspection it turns out we only need to trim the upper boundary of the interval. If we trim the lower boundary, and it then turns out the point estimate is below it, the candidate edge becomes a tree edge. To maintain Invariant 3 we would then need to scan the full data for this new edge. Therefore any decision leading to this outcome is better delayed until it is absolutely necessary (or else proven redundant).

This use of δ and ϵ is analogous to their use in "Probably Approximately Correct" analysis: on each decision, with high probability $(1-\delta)$ we will make at worst a small mistake (ϵ).

5. Experimental Results

In the following description of experiments, we vary different parameters for the data and the algorithm. Unless otherwise specified, these are the default values for the parameters. We set δ to 1% and ϵ to 0. The initial sample size is 5000 records. There are $100,000$ records and 100 attributes. The data is numeric. The data-generation process for the synthetic sets is as in Figure 2. The correct dependency-tree for this process is described in Figure 3. In the categorical case, the network is identical, but parent-child relationships are as follows. The root is true with probability 0.5. For the other nodes, the probability of them being true given that their parent is true is $0.5+c$ for some constant c, and the probability of them being true given that their parent is false is $0.5-c$. By setting the "coupling" parameter c to 0 we get a completely random data, while a value of 0.5 generates a highly-structured, noiseless data-set.

In nearly all of the 615 synthetic experiments in which both ϵ and δ were set to their default values, the output tree was identical to the one generated by the naive algorithm which first computes the full correlation matrix. Only in 16 experiments did this not happen. In all of these runs the weight of the output tree was at least 99.99% the weight of the "naive" tree, and the log-likelihood was nearly identical. Table 1 and Figures 8 and 10 present the quality of the output trees in the remaining experiments.

To construct the correlation matrix from the full data, each of the R records needs to be considered for each of the $\binom{n}{2}$ attribute pairs. We evaluate the performance of our algorithm by adding the number of records that were scanned for all the attribute-pairs, and dividing the total by $R\binom{n}{2}$. We call this number the "edge usage" of our algorithm. The closer it is to zero, the more efficient our sampling is, while a value of one means the same amount of work as for the naive algorithm.

We first demonstrate the speed of our algorithm as compared with the full $O(Rn^2)$ scan. As expected, both algorithms scale linearly with the number of points (data not shown). Figure 4 suggests that for medium-sized datasets the speedup may not be that large (a little under four), but as the number of records grows the scale factor flattens out at about 0.04, meaning a 25-fold speed-up. As expected, the plot for the edge usage versus the number of points looks the same as the one that shows the relative running time (data not shown).

The speed-up improves with the number of attributes as well. See Figure 5. Note how the rate of improvement slows down as the number of attributes grows. This agrees with the theoretical analysis, which predicts that MIST takes $O(1/n)$ as much time as the original algorithm.

Our next experiment examines the sensitivity of our algorithm to noisy data. Data was generated in the usual way, except that some fraction of the records had completely random values in all attributes. As shown in Figure 6, when ϵ is 0, data-usage is kept below 15% of maximum, similar to the performance with noiseless data, as long as the noise level is below 30%. With ϵ set to 0.01, this is true for all noise levels up to 80%.

Recall that the δ parameter controls how loose the confidence intervals are. The bigger it is, the higher the chance that a wrong decision about a tree-edge inclusion or exclusion will be made. Figure 7 shows

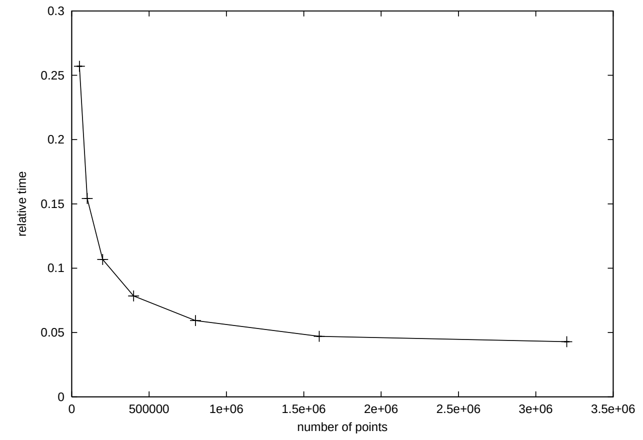

Figure 4. Running time, divided by the running time of the naive algorithm for the same data, as a function of the number of points. In a total of 90 runs reported here, only three did not produce the same output as the naive algorithm.

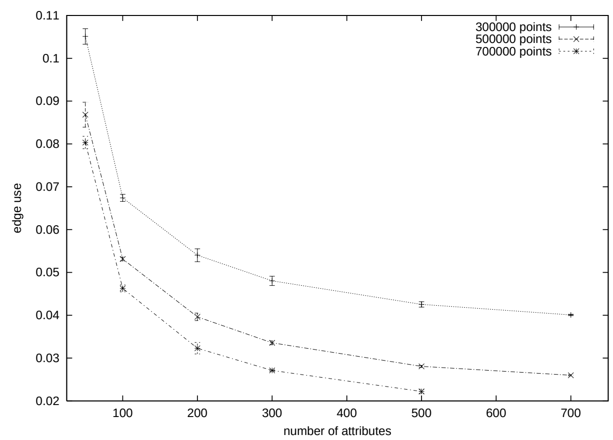

Figure 5. Edge usage (indicative of relative running time) as a function of the number of attributes. In a total of 138 runs reported here, twelve did not produce the same output as the naive algorithm.

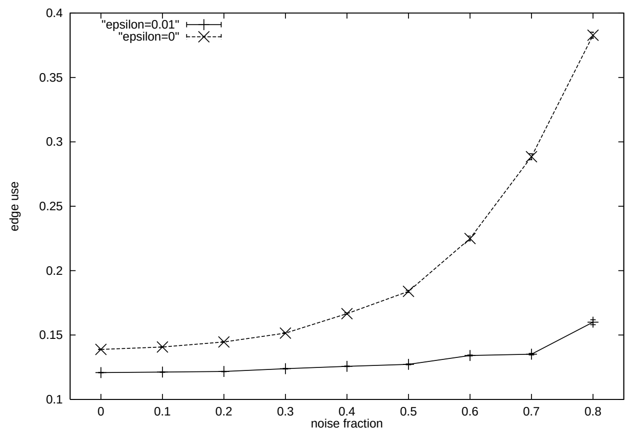

Figure 6. Edge usage as a function of noise. In a total of 180 runs reported here for the case $\epsilon = 0$, only two did not produce the same output as the naive algorithm.

Table 1. Number of runs, out of 30, that ended in a tree different than the full tree for $\epsilon = 0$.

δ	Number of runs
1%	1
2%	1
3%	0
4%	0
5%	0
6%	2
7%	2
8%	2
9%	3
10%	3

the effect of δ on the running time. When ϵ is 0, it appears that higher values of δ do not improve the running time significantly, while increasing the chance of deviation from the output of the full algorithm (see Table 1 for the number of runs that ended in trees that are different than reported by the full algorithm). However, when ϵ was set to 0.05, an improvement in running-time can be traded for some decrease in the quality of the output (Figure 8). For this case none of the 30 runs in any of the 10 values for δ resulted in the same identical tree as with the full algorithm.

We continue to examine the effect the ϵ parameter has on performance. Recall that it controls a heuristic that may decrease the edge usage, but may also lead to the wrong edges being included in the tree. See Figure 9 for the effect on running time (or, equivalently, on the the number of data-cells scanned). We see that changes in ϵ can dramatically improve performance,

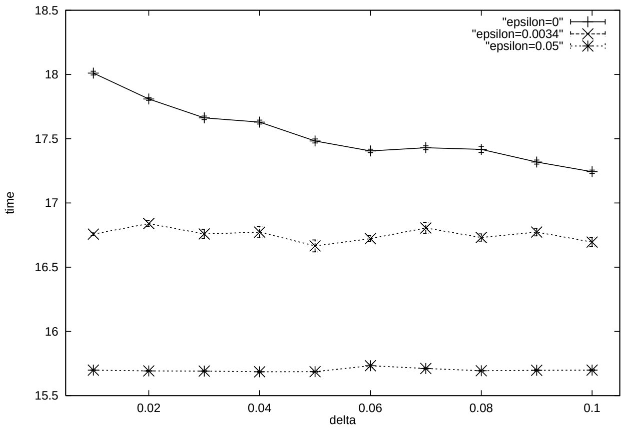

Figure 7. Running time, in seconds, as a function of δ and ϵ.

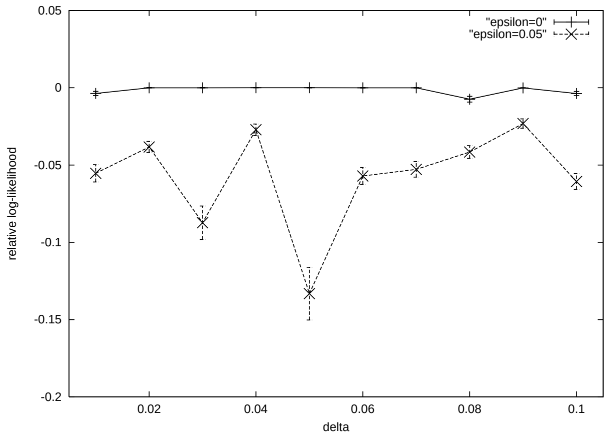

Figure 8. Difference in log-likelihood (divided by the number of records) of the generated trees, as a function of δ and ϵ. Baseline log-likelihoods were in the order of 3.5×10^6.

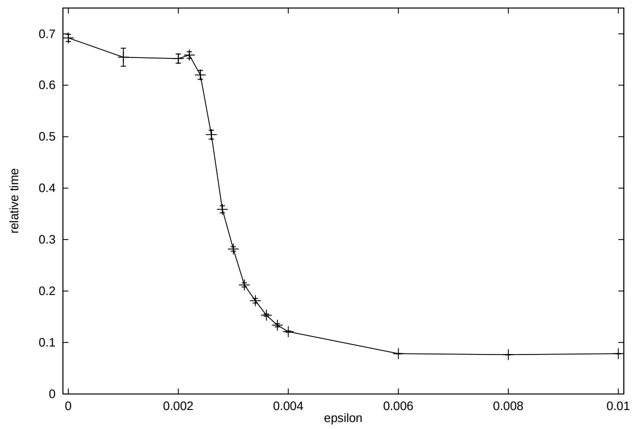

Figure 9. Edge-usage as a function of ϵ. The data is categorical, with the "coupling" parameter c set to 0.04.

down from 70% to about 10% on this data-set, with a sharp drop in the $0.002 — 0.004$ range. The interesting question is, how badly is the output quality affected by this heuristic. To answer this we have plotted the data from the same experiments, but now with the X-axis being the relative log-likelihood of the output (Figure 10). The worst log-likelihood ratio is about 0.05, and it seems that with careful selection of ϵ it is possible to enjoy most of the time savings while sacrificing very little accuracy. For this particular data-set this "sweet-spot" is approximately at $\epsilon = 0.0028$.

To test our algorithm on real-life data, we analyzed data derived from astronomical observations taken in the Sloan Digital Sky Survey. The 21 numeric attributes are the results of various computations on an image of a sky object. Overall, there are about 2.4 million records. The naive algorithm runs in 143 seconds on a 667-MHz Compaq Alpha. See Table 2 for the relative run-times for the fast algorithm. It runs in about half the time, with virtually the same output: in the one case where the generated tree was different, the difference was in two edges, with no significant change in log-likelihood or total tree weight. Extrapolating from the results for synthetic data to data sets of this size, we expect the speedup to increase as we add more attributes.

We then turned this data into a *second-order* dataset to provide an example of a dataset with many attributes and many records. We first discretized all of the attributes. Then we added all pairwise conjunctions of these attributes. There were 23 original attributes $X_1 \ldots X_{23}$ to which were added $\binom{23}{2}$ additional attribute $A_{i,j}$ where $A_{i,j} = X_i \wedge X_j$.

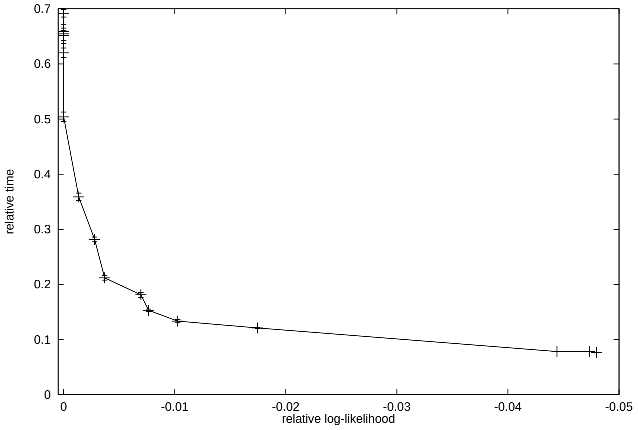

Figure 10. Relative log-likelihood vs. relative time, as a function of ϵ. This is data from the same experiments as plotted in Figure 9.

Table 2. Run-times, relative to the naive algorithm, on real data.

ϵ	δ	Run-Time	Same result?
0.01	0	55%	$\surd$
0.01	0.05	45%	$\surd$
0.01	0.1	43%	$\times$
0.05	0.05	45%	$\surd$

After doing that for all attributes and removing attributes which take on constant values we were left with 148 attributes and the original 2.4 million records. The naive algorithm constructs a tree for this set in 6.6 hours, while the fast algorithm (with default settings) takes about 21 minutes, meaning a speedup of 19. The tree generated by the fast algorithm weights 99.89% of the naive tree, and the difference in log-likelihoods is 1.26×10^5, or about 0.05 per record.

6. Conclusion and Future Work

We have presented an algorithm that implements a "probably approximately correct" approach to dependency-tree construction for numeric and categorical data. Experiments in sets with millions of records and hundreds of attributes show it is capable of processing massive data-sets up to thirty times faster than the naive algorithm, with no appreciable loss in the quality of the output.

Currently, the running time grows linearly with the number of attributes and records. While this is a significant improvement over the naive algorithm, we would like to reduce the data-usage even further. For this we need to be able to use probabilistic bounds for the tree edges as well. We plan on modifying the algorithm to accommodate these intervals when a non-tree edge is compared to a tree edge.

Another issue we would like to tackle is disk access. One advantage the naive algorithm has is that it is easily executed with a single sequential scan of the data file. We will explore the ways in which this behaviour can be attained or approximated by our algorithm.

While we have presented formulas for both numeric and categorical data, we do not allow both types of attributes to be present in a single network. To the best of our knowledge, there is no theoretical framework in which such mutual-information coefficients can be defined.

7. Acknowledgements

We would like to thank Mihai Budiu, Scott Davies, Danny Sleator and Larry Wasserman for helpful discussions, and Andy Connolly for providing access to data.

References

Chow, C. K., & Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory, 14*, 462–467.

Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-00)* (pp. 71–80). N. Y.: ACM Press.

Domingos, P., & Hulten, G. (2001a). A general method for scaling up machine learning algorithms and its application to clustering. *Proceeding of the 17th International Conference on Machine Learning.* San Francisco, CA: Morgan Kaufmann.

Domingos, P., & Hulten, G. (2001b). Learning from infinite data in finite time. *Proceedings of the 14th Neural Information Processing Systems (NIPS-2001).* Vancouver, British Columbia, Canada.

Friedman, N., Nachman, I., & Peér, D. (1999). Learning bayesian network structure from massive datasets: The "sparse candidate" algorithm. *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)* (pp. 206–215). Stockholm, Sweden.

Maron, O., & Moore, A. W. (1994). Hoeffding races: Accelerating model selection search for classification and function approximation. *Advances in Neural Information Processing Systems* (pp. 59–66). Denver, Colorado: Morgan Kaufmann.

Meila, M. (1999a). An accelerated Chow and Liu algorithm: fitting tree distributions to high dimensional sparse data. *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99)*. Bled, Slovenia.

Meila, M. (1999b). *Learning with mixtures of trees*. Doctoral dissertation, Massachusetts Institute of Technology.

Moore, A. W., & Lee, M. S. (1994). Efficient algorithms for minimizing cross validation error. *Proceedings of the 11th International Conference on Machine Learning (ICML-94)* (pp. 190–198). Morgan Kaufmann.

Reza, F. (1994). *An introduction to information teory*. New York: Dover Publications. pp. 282–283.

Tarjan, R. E. (1983). *Data structures and network algorithms*, vol. 44 of *CBMS-NSF Reg. Conf. Ser. Appl. Math.* SIAM.