Constructor: A System for the Induction of Probabilistic Models

Robert M. Fung and Stuart L. Crawford
Advanced Decision Systems
1500 Plymouth Street
Mountain View, CA 94043-1230

Abstract

The probabilistic network technology is a knowledge-based technique which focuses on reasoning under uncertainty. Because of its well defined semantics and solid theoretical foundations, the technology is finding increasing application in fields such as medical diagnosis, machine vision, military situation assessment, petroleum exploration, and information retrieval. However, like other knowledge-based techniques, acquiring the qualitative and quantitative information needed to build these networks can be highly labor-intensive.

CONSTRUCTOR integrates techniques and concepts from probabilistic networks, artificial intelligence, and statistics in order to induce Markov networks (*i.e.,* undirected probabilistic networks). The resulting networks are useful both qualitatively for concept organization and quantitatively for the assessment of new data.

The primary goal of CONSTRUCTOR is to find qualitative structure from data. CONSTRUCTOR finds structure by first, modeling each feature in a data set as a node in a Markov network and secondly, by finding the neighbors of each node in the network. In Markov networks, the neighbors of a node have the property of being the smallest set of nodes which "shield" the node from being affected by other nodes in the graph. This property is used in a heuristic search to identify each node's neighbors. The traditional χ^2 test for independence is used to test if a set of nodes "shield" another node. Cross-validation is used to estimate the quality of alternative structures.

Introduction

The probabilistic networks technology is a new knowledge-based approach for reasoning under uncertainty. Because of its well-defined semantics and solid theoretical foundations, it is finding increasing application in fields such as medical diagnosis, machine vision, military situation assessment, petroleum exploration, and information retrieval. However, like other knowledge-intensive approaches, acquiring the qualitative and quantitative information needed to build these networks is a highly labor-intensive task which requires trained personnel (*i.e.,* knowledge engineers). In an effort to address this problem, techniques for network-induction [4, 14] have been explored. However these techniques are limited to the recovery of tree structures and these structures are often not expressive enough to represent real-world situations.

In this paper, we describe CONSTRUCTOR—a system designed to more fully address this "knowledge acquisition bottleneck". CONSTRUCTOR induces discrete, Markov networks of *arbitrary topology,* from data. These networks contain a quantitative (*i.e.,* probabilistic) characterization of the data but, perhaps more importantly, also contain a qualitative structural description of the data. By qualitative structure we mean, loosely, the positive and negative *causal* relationships between factors as well as the positive and negative *correlative* relationships between factors in the processes under analysis. CONSTRUCTOR has as a primary focus the recovery of qualitative structures since these structures not only determine which quantitative relationships are recovered, but also because such structures are readily interpretable and thus are valuable in explaining the real world processes under analysis.

The CONSTRUCTOR algorithm is based on the concept of "constructing" a network from a data set by instantiating a node for each attribute in the data set and identifying the neighbors of each node in the network. Identifying the neighbors of a particular node is operationalized by heuristically searching for the smallest set of nodes which makes the node independent of all other nodes in the network. Independence is tested through the use of the χ^2 test of independence. The resulting network is a Markov network.

Throughout this paper we illustrate concepts with variations on a single problem, which was originally described in [1]. This problem involves an LED display connected to a numeric keypad. The display is *faulty,* however, since the output of the display may not always match the key that was depressed. Figure 1 shows the numerical digit display unit, and illustrates the components of the display that must be illuminated

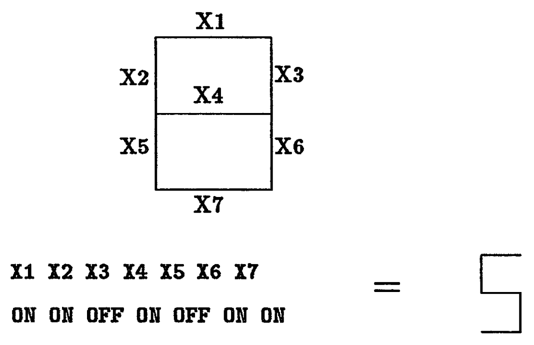

Figure 1: *The LED Display*

to generate the number 5. Since each component of the display has a 0.9 probability of illuminating in error, a training set can be generated in which each example consists of the key that was depressed and the actual state of the LED. The example task is to use CONSTRUCTOR to recover a probabilistic model which can be used both *qualitatively* to understand the workings of the LED display and *quantitatively* to assess the intended digit given new LED displays.

Related Work

This section describes a number of alternative approaches to the model construction task addressed by CONSTRUCTOR.

Chow & Liu, 1968

This research [4] was concerned with reducing the significant amounts of memory needed to represent large discrete probability distributions. The research resulted in the Maximum Weight Spanning Tree (MWST) algorithm. The algorithm takes as input a training set and produces as output a probabilistic network with a *tree* topology.

The algorithm has several desirable properties. First, if the underlying distribution from which the training set is sampled has a tree structure, then the MWST algorithm will find the "optimal" tree. Secondly, the algorithm is computationally tractable—it runs in $O(n^2)$ where n is the number of attributes in the training set. Thirdly, since the algorithm only uses pairwise statistics, the size of the training set can be modest and still achieve good results.

The major drawback of this algorithm is that situations with more complex structures (*i.e.*, non-tree topologies) cannot be accurately represented. This is a serious drawback since most real problems do not have "tree" structures.

Rebane & Pearl, 1987

This research [14] extended the MWST algorithm of Chow and Liu to include the partial recovery of singly-connected graphs.[1] The basic idea is to first run the Chow and Liu algorithm to determine node neighbors. A test is then made for each node to determine which of the node's neighbors are pairwise independent. If a set of mutually pairwise independent nodes exists, then they are labeled as *predecessors* of the node in question and the remainder are labeled as *successors*. However, if no pairwise independent nodes are found, no labeling can take place.

Although this algorithm keeps all the positive features of the MWST algorithm, it does little to address its major drawback—the accurate representation of situations which do not have a singly-connected network representation.

Concept Formation Research

The CONSTRUCTOR approach to structure learning shares some similarities with the more widely known machine learning work on concept formation [5, 11, 12]. The clearest similarity relates to the goals of the algorithms: the CONSTRUCTOR and concept formation algorithms are both designed "to help one better understand the world and to make predictions about its future behavior."[7]. In addition, both approaches learn in an unsupervised manner—no advice or intervention is required from a "teacher" and like the AutoClass system [3], *Constructor* is inherently Bayesian in spirit, making use of probability distributions over the attributes of the observations in the training set. Furthermore, the networks generated by *Constructor* are not strictly hierarchical, as are the knowledge representations produced by many of the concept formation techniques.

Component Techniques

CONSTRUCTOR makes use of traditional techniques of artificial intelligence as well as recent advances in both probabilistic representation and inference and statistical analysis. The probabilistic network technologies provide an intuitive representation for a joint probability distribution as well as efficient techniques of probabilistic inference. From AI, we use techniques of heuristic search in order to efficiently find the structure which best represents the data. From statistics, we use the χ^2 test to decide when attributes are probabilistically independent and cross-validation for selecting the "best connected" network.

Probabilistic Networks: Probabilistic networks [8, 13] are used for representing and reasoning with uncertain beliefs, and are based on the well-established

[1]A singly connected graph is one in which there are no (undirected) cycles.

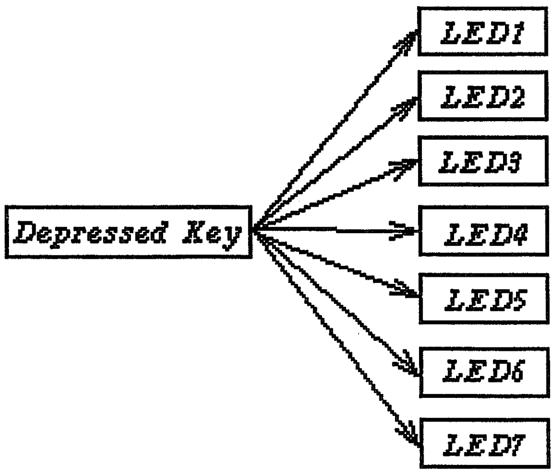

Figure 2: *Probabilistic Network for the LED Problem*

theory of Bayesian probability. The primary innovation of probabilistic networks is the explicit representation of conditional independence relations. The representation of conditional independence relations in a probabilistic network is encoded in the topology of the network and can be illustrated with an example. In the network topology shown in Figure 2, the *Depressed Key* node separates all of the LED segments from each other. This topology implies that the state of any of the LED segments is independent of the state of every other segment in the display given that it is known what key has been depressed. This independence is *conditional* however, since the LED states are dependent if it is not known which key has been depressed. The three-place relation symbol I will be used to denote conditional independence. For example, $I(n_i, n_k, n_j)$ denotes that n_i is conditionally independent of n_j given n_k.

There are two types of probabilistic networks: Bayesian and Markov. A Bayesian network contains directed arcs whereas a Markov network contains undirected arcs. Networks of both types represent a joint state space and a probability distribution on that space. Each node of a probabilistic network represents one component of the joint state space (*i.e.,* a mutually exclusive and exhaustive set of states). For example, the *Depressed Key* node in Figure 2 represents the set of states {0 1 2 3 4 5 6 7 8 9}. Each arc of a probabilistic network represents a relationship between the nodes it connects. For example, the arc between the *Depressed Key* node and the *LED1* node indicates that the top element of the LED is probabilistically related to the key that is depressed (*i.e.,* if there is no failure, it will turn on if the depressed key is 0, 2, 3, 5, 7, 8, or 9).

The form of the probability distribution for a network depends on its type. In a Bayesian network, a conditional probability is stored at each node where the conditioning variables are the node's *predecessors.* In a Markov network, the nodes are grouped into cliques.[2], and probabilistic quantities are associated with the cliques of a network instead of with the nodes themselves.

Useful inferences can be made given a probabilistic network that represents a situation and evidence about the situation. For example, given the evidence that *LED2 LED4 LED6* and *LED7* are illuminated, and the network shown in Figure 2, one could infer updated beliefs about the "depressed key". As one would expect for this example, the result would be relatively strong beliefs for "5" and "6" and relatively weak beliefs for the other possible values for "depressed key". Several algorithms for inference have been reported, [2, 10, 13]. While each of these algorithms has significantly different methods for inference, they are equivalent in that given a probabilistic network and a particular query, they will infer exactly the same result.

Every node in a probabilistic network has a *Markov boundary* that "shields" it from being affected by every other node outside the boundary. In other words, given the Markov boundary of a node, that node is conditionally independent of every other node in the network. Formally, the Markov boundary of a node n_i in a network $\mathcal{U}$ is a subset of nodes S such that:

$$I(n_i, S, \mathcal{U} \backslash S \backslash n_i) \qquad (1)$$

and no proper subsets of S satisfy Equation 1. (The operator "\" denotes set difference.) The Markov boundary of a Bayesian network node is simply the union of the node's predecessors, the node's successors, and the predecessors of the node's successors. The Markov boundary for a Markov network node is simply its neighbors. Figure 3 shows the Markov boundary for a Bayesian network node. The Markov boundary concept is crucial to the development of the CONSTRUCTOR algorithm.

Statistical Methods: When attributes take on categorical values, the degree to which two attributes are statistically independent can be ascertained via the well known χ^2 test of independence. For this test, the data are arranged in a two-way *contingency* table such that the possible values of one attribute make up the rows of the table, the possible values of the other attribute make up the columns, and the data proportions of the various attribute/value combinations fill the cells of the table. The conditional independence of attributes a_i and a_j *given* attribute a_k can be tested by preparing a three-way contingency table so that the values of a_i and a_j make up the rows and columns of the table, and the values of a_k make up the *layers.*

[2]A clique is a set of nodes in which every node in the set is a neighbor of every other node in the set.

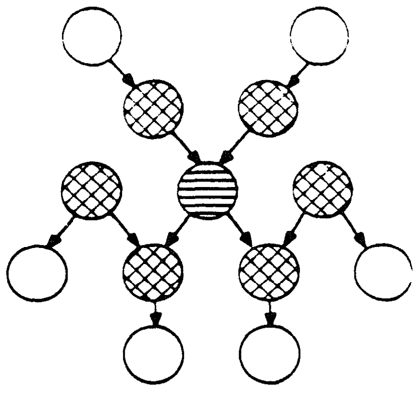

Figure 3: *A Markov Boundary*

Conditional independence is tested by computing separate χ^2 statistics for each layer and summing. The conditional independence of attributes a_i and a_j given a *set* of attributes $\{a_k, a_l, \ldots\}$ can be tested by simply forming a "macro" attribute, a_m, whose values consist of all combinations of values of the attributes in the conditioning set.

In most instances of model fitting algorithms (*e.g.*, CONSTRUCTOR), the eventual goal is to uncover a model that will be useful for *prediction*. It is an unfortunate fact that the predictive performance of such data-derived models often falls short of expectations. The model will often be highly predictive when test cases are drawn from the data used to formulate the model, but less predictive when new data are presented. This phenomenon is called "statistical overfitting" and indicates that, on average, the fit of the model to the data used to build the model is much closer than the fit to new data. A simple mechanism for addressing the problem of overfitting is *cross-validation* [15], often characterized as a "leave some out" technique in which a model formed from a portion of the data is subsequently tested against the data left out of the model formation process. The use of cross-validation is not restricted to model *assessment*, however, since the approach can also be used to assist with model *selection*. Breiman *et al.* [1] describe a powerful use of cross-validation for model selection in the context of finding the "right sized" classification tree. Cross-validation is used in a similar manner in CONSTRUCTOR for finding the "best-connected" network.

Constructor Algorithm

The input to the CONSTRUCTOR algorithm is a training set $\mathcal{X}$ consisting of N examples, $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$. Each example, $\vec{x}_i$, consists of P discrete values $\{x_{i1}, \ldots, x_{ij}, \ldots, x_{iP}\}$, one for each of the P attributes a_j in a set of attributes $\mathcal{A}$. The value set v_j is the possible set of values for attribute a_j. The value set can either be input by the user or derived from the training set, $\bigcup_i x_{ij}$.

In the faulty LED example, the first attribute, a_1, is an integer-valued attribute indicating the numeric key that was depressed. The rest of the attributes $\{a_2 \ldots a_8\}$ are boolean-valued attributes to indicate whether or not a particular LED component is illuminated. Therefore, v_1 is $\{0, 1, \ldots, 9\}$, and $v_{2\ldots8}$ is $\{on\ off\}$.

The output of the CONSTRUCTOR algorithm is a Markov network $\mathcal{U}$. For each attribute a_i in $\mathcal{X}$, the network contains a node n_i whose state space is the value set v_i of a_i. Qualitatively, the network represents the relevance relationships between attributes in the training set. Quantitatively, the network represents an estimate of the joint probability distribution from which the samples are derived.

The CONSTRUCTOR algorithm takes as its starting point the simple theoretical notion that the structure of the probabilistic network $\mathcal{U}$ can be identified if the neighbors (*i.e.*, Markov boundary) of each node of the network are found. The Markov boundary $\mathcal{B}_i$ of a node n_i in the network $\mathcal{U}$ is found by searching for the set of nodes which correspond to the smallest set of attributes $\mathcal{S}$ such that:

$$I(a_i, \mathcal{S}, \mathcal{A}\backslash\mathcal{S}\backslash a_i) \tag{2}$$

and no proper subsets of $\mathcal{S}$ satisfy Equation 2. In CONSTRUCTOR the determination of conditional indpendence is made with the χ^2 test. However, finding such relations in large data sets is computationally complex since every such test will include the consideration of every feature of the data set.

CONSTRUCTOR therefore limits its attentions to distributions which are "composable". Such distributions have the property that

$$I(B, C, D) \Longleftrightarrow (\forall b \in B)(\forall d \in D)\ I(b, C, d). \tag{3}$$

where b, d represent individual discrete random variables and B, C, D represent sets of such variables. This property states that finding whether two sets of elements are conditionally independent of a third set can be determined by the much simpler computational task of checking whether every pairwise combination of elements in the two sets is conditionally independent of the third set and therefore simplifies the finding of Markov boundaries to finding the smallest set of attributes $\mathcal{S}$ such that:

$$(\forall a_j \in \mathcal{U}\backslash\mathcal{S}\backslash a_i)\ I(a_i, \mathcal{S}, a_j) \tag{4}$$

"Composable" models are a much broader class of models topologically and contain the previously explored "tree" models of [4, 14].

```
begin Constructor: (Training Set)
   compute list of $\alpha$-levels;
   initialize;
   for each $\alpha$-level do:
        select cross-validation sample;
        initialize frequency tables;
        find network;
        estimate network error;
        compute network complexity;
   end for
   report networks, errors, complexities;
end Constructor.
```

Figure 4: *Overall* CONSTRUCTOR *Algorithm.*

Besides its use to find Markov boundaries, CONSTRUCTOR makes use of "composability" in a preprocessing step to separate attributes into mutually-independent sets.

$$I(S_1, \emptyset, S_2) \Longleftrightarrow (\forall a_i \in S_1)(\forall a_j \in S_2)\ I(a_i, \emptyset, a_j) \quad (5)$$

Figure 4 shows the overall CONSTRUCTOR algorithm. In this and subsequent listings, a step is underlined and shown in more detail in a later figure.

Network identification is the heart of the CONSTRUCTOR algorithm and is designed to find the probabilistic network that best represents the training set given an input parameter, α. This parameter is the standard statistical measure of type I error and controls how much confirmation is needed before independence relations can be concluded in the process. Network identification works by searching for the Markov boundary of each attribute in the training set. Wrapped around network identification is cross-validation—used to select a level of α that will deliver the "best connected" network. The function of cross-validation in CONSTRUCTOR is to assess the performance of a *set* of induced networks, each of which has a different degree of "connectivity". Each network in the set is obtained via the use of a different setting of the α parameter. The network with the best cross-validated performance is then selected as the "best connected" network.

Network Identification: Network identification involves successively finding the *neighbors* of each attribute in the training set. Unfortunately, the problem of finding the neighbors of an attribute involves searching through the power set of a set of possible neighbors and the computational complexity for exhaustive search thus grows exponentially with problem size.

Managing the exponential process of finding neighbors is the *primary challenge* for the network identification task. This is accomplished by a number of diverse mechanisms which include *heuristic search.*

```
begin Find-Network: ($\alpha$-level)
   find mutually independent attribute subsets;
   for each subset do:
        for each attribute do:
             find neighbors;
        end for
   end for
   instantiate Bayesian network;
   find cliques;
   estimate clique potentials;
   return network;
end Find-network.
```

Figure 5: *The Network Identification Algorithm.*

In addition, the CONSTRUCTOR algorithm uses other mechanisms to minimize the size of the set of possible neighbors of an attribute. For example, the network identification process uses findings of previous search results in order to reduce the possible neighbors set. This is possible because neighbor relationships are symmetric: *Neighbor-of*$(n_i, n_j) \Leftrightarrow$ *Neighbor-of*(n_j, n_i). In this way, finding the neighbors and non-neighbors for one attribute adds to the knowledge about the neighborhood relationships of every other attribute in the training set.

In spite of these mechanisms, there will be cases in which the search process will be lengthy. There are, however, two immediate observations which can be made about such cases. First, we do not expect to face such situations very often. It has been claimed that training sets derived from real world situations will usually yield sparse graphs (*i.e.*, only a few neighbors per attribute) since real world situations are inherently structured. In these cases, attribute neighbors can be found quickly. In cases which do not have this property, the resulting network will be densely connected and will probably be of little use since little qualitative information will be extractable from the network.

Secondly, a feasible solution (*i.e.*, set of neighbors) always exists—the whole set of possible neighbors. Therefore, in hard subproblems the process can always be terminated successfully by returning as a solution the full set of possible neighbors.

The network identification algorithm is outlined in Figure 5 and described in more detail below.

Subset Selection: The first step in network identification requires using the χ^2 test to assess the pairwise independence of all attributes in the training set. With these results, the attributes can be partitioned into independent sets of attributes. That is, for any pair of independent sets, each attribute in one set will be independent of every attribute in the other set. Locating these subsets helps to reduce the complexity of the neighbor search process.

```
begin Find-Neighbors: (attribute $a_j$)
   root ← $p$;
   known-neighbors ← $\mathcal{N}^k$;
   shortest-path ← $\mathcal{N}^c$;
   shortest-path ← $a_j$;
   spawn first level;
   repeat:
      select node to expand;
      current-path ← current-path + node;
      when shortest-path > current-path:
         spawn and coalesce children;
         for each child do:
            if independent($a_j$,intervening nodes,child)
            then prune child;
            if all children pruned
            then shortest-path ← current-path;
         end for
      end when
   until: no nodes can be expanded;
   return shortest-path;
end Find-neighbors.
```

Figure 6: *The Neighbor Identification Algorithm.*

Neighbor Identification: Finding the neighbors for every attribute in a training set is an iterative search process based on finding the Markov boundary for each attribute. As discussed earlier, the Markov boundary is the smallest set of attributes that makes the target attribute conditionally independent of every other attribute in the training set.

Finding the neighbors of a particular attribute a_j, outlined in Figure 6, begins with the instantiation of an *attribute tree*, consisting of *attribute nodes*, each of which represents an attribute in the training set. The root node of an attribute tree always represents the target attribute a_j (*i.e.*, the attribute for which neighbors are being sought). Each branch of a partial attribute tree (*i.e.*, the path from the root node to a leaf attribute node) represents a *hypothesis* that the attributes associated with the attribute nodes in that branch make up the Markov boundary for the target attribute.

The search for the Markov boundary of attribute a_j begins by instantiating the root node of the attribute tree as a_j. An attribute node for each member of the set of *known* neighbors, $\mathcal{N}^k$, is then sequentially added to the attribute tree in a single branch rooted at a_j. The set of *candidate* new neighbors, $\mathcal{N}^c$, is initialized to $\mathcal{A}\backslash\{a_j\}\backslash\mathcal{N}^k\backslash\mathcal{N}^n$, where $\mathcal{N}^n$ is the set of nodes known *not* to be neighbors of a_j.[3] In addition, the current *shortest path* is initialized to be $\mathcal{N}^c$.

[3]For the very first attribute to be explored, $\mathcal{N}^k = \mathcal{N}^n = \emptyset$ and so the initial set of candidate neighbors consists of $\mathcal{N}^c = \mathcal{A}\backslash\{a_j\}$.

The first step of the search process is to choose a leaf node for expansion. Initially there will only be one leaf node—the end of the $\mathcal{N}^k$ branch. Expansion of a node simply means that a child node is instantiated for every member of $\mathcal{N}^c$ and takes place only if expansion will *not* create a branch whose length exceeds that of the current shortest path. Note that to avoid duplication, each new branch is merged with any existing branch that contains exactly the same attributes. After expansion, a χ^2 test is then carried out to determine if a_j is conditionally independent of *each* of the new attribute leaf nodes *given* the intervening branch. If so, a *new* shortest path has been found. If not, the newly instantiated nodes are pruned away and another leaf node is chosen for expansion. The search continues until there are no leaf attributes to expand and the shortest path is then returned as the set of neighbors for the attribute.

This search is performed for each attribute in the training set. Once completed, a probabilistic network is instantiated in which each attribute is represented by a node whose state space is the value set of its associated attribute. The structure (*i.e.*, connectivity) of the network is determined by placing an *undirected* arc between each pair of neighbors.

Given the network structure, arcs are added which "fill-in" [16] the network and then the nodes in the network are grouped into *cliques*. A joint probability distribution for each node in the clique is then estimated using the empirical distribution found in the training set.

Search Heuristics: The computational efficiency and therefore the viability of CONSTRUCTOR depends on the overall strength of the search heuristics that can eventually be identified. We discuss two of the more powerful heuristics which have been identified.

First, since computational cost grows exponentially with the number of attributes that must be examined in the neighbor search, it makes sense to first explore those attributes which are likely to have only a few neighbors. Because it is unlikely that attributes that are pairwise independent of the target node are neighbors of the target node, this heuristic simply states that neighbors should first be located for those attributes with the largest number of pairwise independencies. Note that, because of the neighbor symmetry relationship described earlier in this section, locating any neighbor relationships will reduce the search space for subsequent neighbor relationships and so it is efficient to do the shorter searches first.

Second, given that the distance between nodes in a network is a function of their probabilistic dependence, it makes sense to focus the search for the neighbors of a node on those attributes exhibiting the strongest probabilistic dependence with that node. Thus, the second heuristic simply states that, when selecting a node for expansion, first select that node with the largest de-

Table 1: *LED1 Dependency On LED2 and LED3*

LED2	LED3	Prob(LED1 is ON)
ON	ON	0.9
OFF	OFF	0.1
ON	OFF	0.7
OFF	ON	0.3

pendency on the target node given the current state of the tree.

Examples

To illustate the algorithm's operation, we describe the results obtained when CONSTRUCTOR is applied to the digit recognition training set. In particular, we focus on the search to discover the neighbors of attribute *LED1*. The pairwise independence relationship between *LED1* and all other attributes is first examined and it is discovered that *LED1* is independent (at $\alpha = 0.001$) of all attributes save for *LED7*, *LED5*, and *Depressed Key*. All other attributes are therefore pruned from the search tree. The remaining attributes (children of the *LED1* node) are examined and an expansion order is computed. The "dependence" heuristic forms an order based upon the degree of dependence between children and parent. Since *LED7* shows the highest degree of dependence with *LED1*, it is expanded first. It is found that *LED1* and *LED2* through *LED6* are conditionally independent given *LED7*, but since *LED1* and *LED7* are found to be conditionally independent given *Depressed Key*, *LED7* cannot be a neighbor of *LED1* and so it, and its children are pruned. The *Depressed Key* node is expanded next and it is found that all other attributes are conditionally independent given *Depressed Key* and are therefore pruned from consideration. The *Depressed Key* attribute makes up a feasible set of Markov neighbors for the *LED1* attribute and since no other path can possibly be shorter than this one (a path of length 1), the search is terminated.

The neighbors for all attributes are located in the same manner and the final result of this process is a network with exactly the same topology as that illustrated in Figure 2.

Digit Recognition with Loop: For this training set, we introduce an additional source of error into the model for our already faulty display. In this model, the state of *LED1* is determined entirely by the states of *LED2* and *LED3*, both of which depend upon the state of *Depressed Key*. The dependencies are summarized in Table 1.

The introduction of this additional source of error in the digit recognition model amounts to the introduction of a *loop* in the Bayesian network representation of the model, as shown in Figure 7.

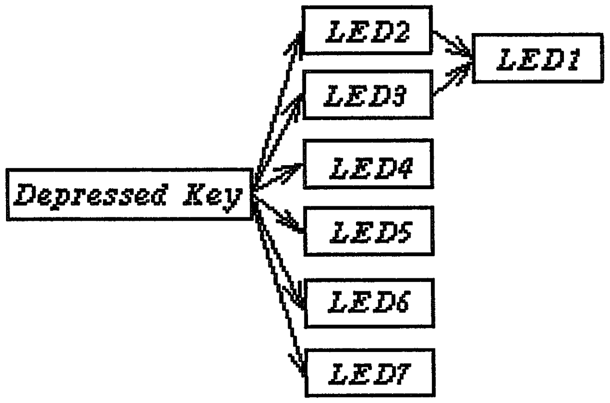

Figure 7: *Digit Recognition Network with Loop*

CONSTRUCTOR can uncover this structure when provided with a set of training examples generated from the probability model. The final result is a Markov network with the same topology as the network in Figure 7 with the exception of an additional arc between *LED2* and *LED3*. This demonstrates, in the simplest possible way, the feasibility of CONSTRUCTOR recovering models with loops.

Discussion

In this paper, we have described the CONSTRUCTOR system for inducing probabilistic models from data. The motivation for CONSTRUCTOR is that previous structure-finding algorithms have been limited to trees and that such structures are often too restrictive for addressing real problems. The basic idea of CONSTRUCTOR is theoretically sound—it is to construct a network by finding the neighbors of each node through a heuristic search process. CONSTRUCTOR operationalizes this theoretical idea through the application of modern statistical techniques and addresses the computational complexities of the task to make significant progress towards a practical machine-learning system.

CONSTRUCTOR has been tested not only on training sets generated from probability models like those in Section 4, but has been tested recently on real data in an information retrieval application [6]. For those training sets generated from probability models, CONSTRUCTOR was able to reconstruct the models and for the information retrieval application, CONSTRUCTOR yielded a network which was intuitive to an expert and performed well in practice.

Although CONSTRUCTOR can be run without user intervention, we strongly believe that users often have important insights that may be easy to acquire and may significantly improve the process of structure learning. In order to benefit from these insights, CONSTRUCTOR allows the user to interact with the system in every phase of the algorithm. For example, the user can choose to have the structure for the entire set of attributes be discovered using the search heuristics, or may wish to choose to find the neighbors for a single attribute himself. In the search for the neighbors of a particular attribute, the user can observe the growth of the search tree and can control the order in which attribute nodes will be expanded. The user can also control pruning and cross-validation and can view intermediate results such as pairwise-independencies, neighbor relationships and the networks which have been found.

Future Research: The primary focus of subsequent research will focus on addressing problems created by the exponential search space in neighbor identification and problems imposed by the "curse" of dimensionality — the observation that high-dimensional space is inherently sparsely populated. For CONSTRUCTOR, the curse of dimensionality is manifested in low cell counts in the contingency tables used to assess independence via the χ^2 test. One approach to this problem is to reduce the dimensionality of the training set and, to this end, some relatively new dimension reducing techniques such as *projection pursuit* [9] will be investigated.

Two other important areas for research are: enlarging the class of distributions which CONSTRUCTOR can recover and exploring parallel implementations. While CONSTRUCTOR recovers *general* graphical structures, there are some classes of distributions that are currently not adequately recovered with CONSTRUCTOR. To address this issue, it is possible to use the conditional independence test of Equation 2 as a heuristic and do an exact test when this heuristic is successful. Much of the CONSTRUCTOR algorithm is inherently parallel. It therefore appears likely that substantial performance gains could result from implementation of CONSTRUCTOR on a parallel architecture.

References

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, 1984.

[2] K. C. Chang and R. M. Fung. Node aggregation for distributed inference in bayesian networks. In *Proceedings of the 11th IJCAI*, Detroit, Michigan, August 1989.

[3] Peter Cheeseman, James Kelly, Matthew Self, John Stutz, Will Taylor, and Don Freeman. Autoclass: a bayesian classification system. In *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, Michigan, June 1988.

[4] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. on Info Theory*, 1968.

[5] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, September 1987.

[6] R. M. Fung, S. L. Crawford, L. Appelbaum, and R. Tong. An architecture for probabilistic concept-based information retrieval. In *Proceedings of the 13th International Conference on Research and Development in Information Retrieval*, September 1990.

[7] John Gennari, Pat Langley, and Doug Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40(1-3):11–61, 1990.

[8] R.A. Howard and J.E. Matheson. Influence diagrams. In R.A. Howard and J.E. Matheson, editors, *The Principles and Applications of Decision Analysis, vol. II*, Menlo Park: Strategic Decisions Group, 1981.

[9] P.J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 1985.

[10] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application in expert systems. *Journal Royal Statistical Society B*, 50, 1988.

[11] M. Lebowitz. Experiments with incremental concept formation: UNIMEM. *Machine Learning*, 2(2):103–138, September 1987.

[12] R.S. Michalski and R.E. Stepp. Learning from observation: conceptual clustering. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufman, 1983.

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

[14] G. Rebane and J. Pearl. The recovery of causal poly-trees from statistical data. *Proc., 3rd Workshop on Uncertainty*, 1987.

[15] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147, 1974. Series B.

[16] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce hypergraphs. *SIAM J. Comput.*, 13:566–579, 1984.