Cutset Networks: A Simple, Tractable, and Scalable Approach for Improving the Accuracy of Chow-Liu Trees

Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate

Computer Science Department
The University of Texas at Dallas
{tahrima.rahman,prasanna.kothalkar,
vibhav.gogate}@utdallas.edu

Abstract. In this paper, we present cutset networks, a new tractable probabilistic model for representing multi-dimensional discrete distributions. Cutset networks are rooted OR search trees, in which each OR node represents conditioning of a variable in the model, with tree Bayesian networks (Chow-Liu trees) at the leaves. From an inference point of view, cutset networks model the mechanics of Pearl's cutset conditioning algorithm, a popular exact inference method for probabilistic graphical models. We present efficient algorithms, which leverage and adopt vast amount of research on decision tree induction for learning cutset networks from data. We also present an expectation-maximization (EM) algorithm for learning mixtures of cutset networks. Our experiments on a wide variety of benchmark datasets clearly demonstrate that compared to approaches for learning other tractable models such as thin-junction trees, latent tree models, arithmetic circuits and sum-product networks, our approach is significantly more scalable, and provides similar or better accuracy.

1 INTRODUCTION

Learning tractable probabilistic models from data has been the subject of much recent research. These models offer a clear advantage over Bayesian networks and Markov networks: exact inference over them can be performed in polynomial time, obviating the need for unreliable, inaccurate approximate inference, not only at learning time but also at query time. Interestingly, experimental results in numerous recent studies [5, 11, 16, 23] have shown that the performance of approaches that learn tractable models from data is similar or better than approaches that learn Bayesian and Markov networks from data. These results suggest that *controlling exact inference complexity* is the key to superior end-to-end performance.

In spite of these promising results, a key bottleneck remains: barring a few exceptions, algorithms that learn tractable models from data are computationally expensive, requiring several hours for even moderately sized problems (e.g., approaches presented in [11, 16, 23] need more than "10 hours" of CPU time for

datasets having 200 variables and 10^4 examples). There are several reasons for this, with the main reason being the high computational complexity of *conditional independence tests*. For example, the LearnSPN algorithm of Gens and Domingos [11] and the ID-SPN algorithm of Rooshenas and Lowd [23] for learning tractable sum-product networks, spend a substantial amount of their execution time on partitioning the given set of variables into conditionally independent components. Other algorithms with strong theoretical guarantees, such as learning efficient Markov networks [13], and learning thin junction trees [1, 4, 19] also suffer from the same problem.

In this paper, we present a tractable class of graphical models, called *cutset networks*, which are rooted OR search trees with tree Bayesian networks (Chow-Liu trees) at the leaves. Each OR node (or sum node) in the OR tree represents conditioning over a variable in the model. Cutset networks derive their name from Pearl's cutset conditioning method [20]. The key idea in cutset conditioning is to condition on a subset of variables in the graphical model, called the cutset, such that the remaining network is a tree. Since, exact probabilistic inference can be performed in time that is linear in the size of the tree (using Belief propagation [20] for instance), the complexity of cutset conditioning is exponential in the cardinality (size) of the cutset. If the cutset is bounded, then cutset conditioning is tractable. However, note that unlike classic cutset conditioning, cutset networks can take advantage of determinism [3, 12] and context-specific independence [2] by allowing different variables to be conditioned on at the same level in the OR search tree. As a result, they can yield a compact representation even if the size of the cutset is arbitrarily large.

The key advantage of cutset networks is that only the leaf nodes, which represent tree Bayesian networks, take advantage of conditional independence, while the OR search tree does not. As a result, to learn cutset networks from data, we do not have to run expensive conditional independence tests at any internal OR node. Moreover, the leaf distributions can be learned in polynomial time, using the classic Chow-Liu algorithm [6]. As a result, if we assume that the size of the cutset (or the height of the OR tree) is bounded by k, and given that the time complexity of the Chow-Liu algorithm is $O(n^2 d)$, where n is the number of variables and d is the number of training examples, the optimal cutset network can be learned in $O(n^{k+2} d)$ time.

Although, the algorithm described above is tractable, it is infeasible for any reasonable k (e.g., 5) that we would like to use in practice. Therefore, to make our algorithm practical, we use splitting heuristics and pruning techniques developed over the last few decades for inducing decision trees from data [21, 18]. The splitting heuristics help us quickly learn a reasonably good cutset network, without any backtracking, while the pruning techniques such as pre-pruning and reduced-error (post) pruning help us avoid over-fitting. To improve the accuracy further, we also consider mixtures of cutset networks, which generalize mixtures of Chow-Liu trees [17] and develop an expectation-maximization algorithm for learning them from data.

We conducted a detailed experimental evaluation comparing our three learning algorithms: learning cutset networks without pruning (CNet), learning cutset networks with pruning (CNetP), and learning mixtures of cutset networks (MCNet), with six existing algorithms for learning tractable models from data: sum-product networks with direct and indirect interactions (ID-SPN) [23], learning tractable Markov networks with Arithmetic Circuits (ACMN) [16], mixtures of trees (MT) [17], stand alone Chow-Liu trees [6], learning sum-product networks (LearnSPN) [11] and latent tree models (LTM) [5]. We found that MCNet is the best performing algorithm in terms of test-set log likelihood score on 11 out of the 20 benchmarks that we experimented with. ID-SPN has the best test-set log likelihood score on 8 benchmarks while ACMN and CNetP are closely tied for the third-best performing algorithm spot. We also measured the time taken to learn the model for ID-SPN and our algorithms, and found that ID-SPN was the slowest algorithm. CNet was the fastest algorithm, while CNetP and MCNet were second and third fastest, respectively. Interestingly, if we look at learning time and accuracy as a whole, CNetP is the best performing algorithm, providing reasonably accurate results in a fraction of the time as compared to MCNet, ACMN and ID-SPN.

The rest of the paper is organized as follows. In section 2, we present background and notation. Section 3 provides the formal definition of cutset networks. Section 4 describes algorithms for learning cutset networks. We present experimental results in section 5 and conclude in section 6.

2 Notation and Background

We borrow notation from [17]. Let V be a set of n discrete random variables where each variable $v \in V$ ranges over a finite discrete domain Δ_v and let $x_v \in \Delta_v$ denote a value that can be assigned to v. Let $A \subseteq V$, then x_A denotes an assignment to all the variables in A. For simplicity, we often denote x_A as x and Δ_v as Δ. The set of domains is denoted by $\Delta_V = \{\Delta_i | i \in V\}$.

A probabilistic graphical model $\mathcal{G}$ is denoted by a triple $\langle V, \Delta_V, F \rangle$ where V and Δ_V are the sets of variables and their domains respectively, and F is a set of real-valued functions. Each function $f \in F$ is defined over a subset $V(f) \subseteq V$ of variables, called its scope. For Bayesian networks (cf. [20, 7]) which are typically depicted using a directed acyclic graph (DAG), F is the set of conditional probability tables (CPTs), where each CPT is defined over a variable given its parents in the DAG. For Markov networks (cf. [14]), F is the set of potential functions. Markov networks are typically depicted using undirected graphs (also called the primal graph) in which we have a node in the graph for each variable in the model and edges connect two variables that appear together in the scope of a function.

A probabilistic graphical model represents the following joint probability distribution over V:

$$P(x) = \frac{1}{Z} \prod_{f \in F} f(x_{V(f)})$$

where x is an assignment to all variables in V, $x_{V(f)}$ denotes the projection of x on $V(f)$ and Z is a normalization constant. For Bayesian networks, $Z = 1$, while for Markov networks $Z > 0$. In Markov networks, Z is also called the *partition function*.

The two main inference problems in probabilistic graphical models are (1) posterior marginal inference: computing the marginal probability of a variable given evidence, namely computing $P(x_v|x_A)$ where x_A is the evidence and $v \in V \setminus A$; and (2) maximum-a-posteriori (MAP) inference: finding an assignment of values to all variables that has the maximum probability given evidence, namely computing $\arg\max_{x_B \in B} P(x_B|x_A)$ where x_A is the evidence and $B = V \setminus A$. Both problems are known to be NP-hard.

2.1 The Chow-Liu Algorithm for Learning Tree Distributions

A tree Bayesian network is a Bayesian network in which each variable has no more than one parent while a tree Markov network is a Markov network whose primal (or interaction) graph is a tree. It is known that both tree Bayesian and Markov networks have the same representation power and therefore can be used interchangeably.

The Chow-Liu algorithm [6] is a classic algorithm for learning tree networks from data. If $P(x)$ is a probability distribution over a set of variables V, then the Chow-Liu algorithm approximates $P(x)$ by a tree network $T(x)$. If $G_T = (V, E_T)$ is an undirected Markov network that induces the distribution $T(x)$, then

$$T(x) \;=\; \frac{\prod\limits_{(u,v)\in E_T} T(x_u, x_v)}{\prod\limits_{v\in V} T(x_v)^{deg(v)-1}}$$

where $deg(v)$ is the degree of vertex v or the number of incident edges to v. If G_T is a directed model such as a Bayesian network, then

$$T(x) \;=\; \prod_{v\in V} T(x_v|x_{pa(v)})$$

where $T(x_v|x_{pa(v)})$ is an arbitrary conditional probability distribution such that $|pa(v)| \leq 1$. The Kullback-Leibler divergence $KL(P,T)$ between $P(x)$ and $T(x)$ is defined as:

$$KL(P,T) \;=\; \sum_x P(x)\log\left(\frac{P(x)}{T(x)}\right)$$

In order to minimize $KL(P,T)$, Chow and Liu proved that each selected edge $(u,v) \in E_T$ has to maximize the total mutual information, $\sum_{(u,v)\in E_T} I(u,v)$. Mutual information, denoted by $I(u,v)$, is a measure of mutual dependence between two random variables u and v and is given by:

$$I(u,v) \;=\; \sum_{x_u\in\Delta_u}\sum_{x_v\in\Delta_v} P(x_u,x_v)\log\left(\frac{P(x_u,x_v)}{P(x_u)P(x_v)}\right) \tag{1}$$

To maximize (1), the Chow-Liu procedure computes the mutual information $I(u, v)$ for all possible pairs of variables in V and then finds the maximum weighted spanning tree $G_T = (V, E_T)$ such that each edge $(u, v) \in E_T$ is weighted by $I(u, v)$. The marginal distribution $T(u, v)$ of a pair of variables (u, v) connected by an edge is the same as $P(u, v)$.

Tree networks are attractive because: (1) learning both the structure and parameters of the distribution are tractable; (2) several probabilistic inference tasks can be solved in linear time; and (3) they have intuitive interpretations.

The time complexity of learning the structure and parameters of a tree network using the Chow-Liu algorithm is $O(n^2 d + n^2 \log(n))$ where n is the number of variables and d is the number of training examples. Since, in practice, $d > \log(n)$, for the rest of the paper, assume that the time complexity of the Chow-Liu algorithm is $O(n^2 d)$.

2.2 OR Trees

OR trees are rooted trees which are used to represent the search space explored during probabilistic inference by conditioning [20, 9]. Each node in an OR tree is labeled by a variable v in the model. Each edge emanating from a node represents the conditioning of the variable v at that node by a value $x_v \in \Delta_v$ and is labeled by the marginal probability of the variable-value assignment given the path from the root to the node. For simplicity, we will focus on binary valued variables. For binary variables, assume that left edges represent the assignment of variable v to 0 and right edges represent $v = 1$. A similar representation can be used for multi-valued variables.

Any distribution can be represented using a OR Tree. In the worst-case, the tree will require $O(2^{n+1})$ parameters to specify the distribution. Figure 1 shows a probability distribution and a possible OR tree.

The distribution represented by an OR tree O is given by:

$$P(x) = \prod_{(v_i, v_j) \in path_O(x)} w(v_i, v_j) \tag{2}$$

where $path_O(x)$ is the path from the root to the unique leaf node $l(x)$ corresponding to the assignment x and $w(v_i, v_j)$ is the probability value attached to the edge between the OR nodes v_i and v_j.

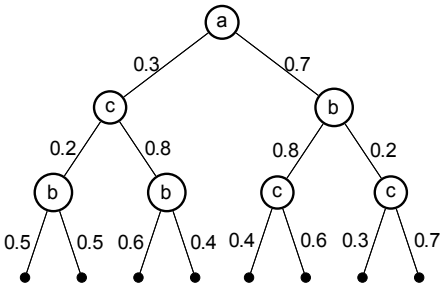
3 Cutset Networks

Cutset Networks (CNets) are a hybrid of rooted OR trees and tree Bayesian networks, with an OR tree at the top and a tree Bayesian network attached to each leaf node of the OR tree. Formally a cutset network is a pair $C = (O, \mathbf{T})$ where O is a rooted OR tree and $\mathbf{T} = \{T_1, \ldots, T_l\}$ is a collection of tree networks. The distribution represented by a cutset network is given by:

$$P(x) = \left(\prod_{(v_i, v_j) \in path_O(x)} w(v_i, v_j) \right) \left(T_{l(x)}(x_{V(T_{l(x)})}) \right) \tag{3}$$

a	b	c	P(a,b,c)
0	0	0	0.030
0	0	1	0.144
0	1	0	0.030
0	1	1	0.096
1	0	0	0.224
1	0	1	0.336
1	1	0	0.042
1	1	1	0.098

(a)



(b)

Fig. 1. (a) A probability distribution, (b) An OR tree representing the distribution given in (a). The left branch from a node represents conditioning by 0, whereas the right branch represents conditioning by 1.

where $path_O(x)$ is the path from the root to the unique leaf node $l(x)$ corresponding to the assignment x, $w(v_i, v_j)$ is the probability value attached to the edge between the OR nodes v_i and v_j and $T_{l(x)}$ is the tree Bayesian network associated with $l(x)$ and $V(T_{l(x)})$ is the set of variables over which $T_{l(x)}$ is defined. Fig. 2 shows an example cutset network.

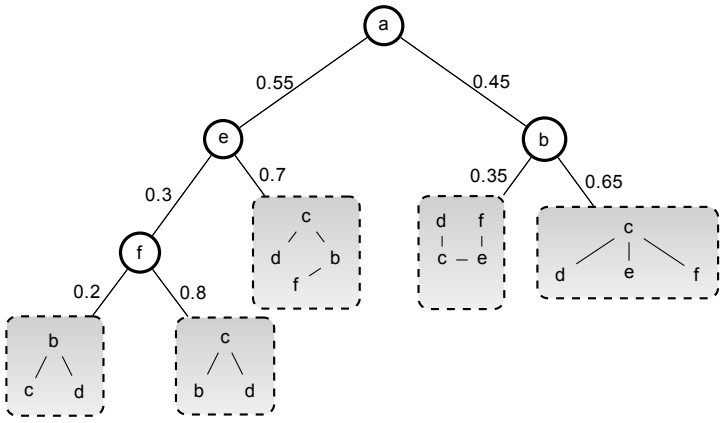


Fig. 2. A cutset network

Note that unlike classic cutset conditioning, cutset networks can take advantage of determinism [3] and context-specific independence [2] by branching on different variables at the same level (or depth) [12, 13]. As a result, they can

Algorithm 1 LearnCNet

Input: Training dataset $\mathcal{D} = \{x^1, ..., x^d\}$, Variables V.
Output: A cutset network C

if Termination condition is satisfied **then**
  return ChowLiuTree($\mathcal{D}$)
end if
Heuristically select a variable $v \in V$ for splitting
Create a new node O_v labeled by v.
/*
`Each node has a left child` $O_v.left$`, a right child` $O_v.right$`, a left probability` $O_v.lp$ `and a right probability` $O_v.rp$.
*/
Let $\mathcal{D}_{v=0} = \{x^i \in \mathcal{D} | x^i_v = 0\}$
Let $\mathcal{D}_{v=1} = \{x^i \in \mathcal{D} | x^i_v = 1\}$
$O_v.lp \leftarrow \frac{|\mathcal{D}_{v=0}|}{|\mathcal{D}|}$
$O_v.rp \leftarrow \frac{|\mathcal{D}_{v=1}|}{|\mathcal{D}|}$
$O_v.left \leftarrow$ LearnCNet($\mathcal{D}_{v=0}, V \setminus v$)
$O_v.right \leftarrow$ LearnCNet($\mathcal{D}_{v=1}, V \setminus v$)
return O_v

yield a compact representation, even if the size of the cutset[1] is arbitrarily large. For example, consider the cutset network given in Fig. 2. The left most leaf node represents a tree Bayesian network over $V \setminus \{a, e, f\}$ while the right most leaf node represents a tree Bayesian network over $V \setminus \{a, b\}$. Technically, the size of the cutset can be as large as the union of the variables mentioned at various levels in the OR tree. Thus, for the cutset network given in Fig. 2, the size of the cutset can be as large as $\{a, b, e, f\}$.

4 Learning Cutset Networks

As mentioned in the introduction, if the size of the cutset is bounded by k, we can easily come up with a polynomial time algorithm for learning cutset networks: systematically search over all subsets of size k. Unfortunately, this naive algorithm is impractical because of its high polynomial complexity; the time complexity is at least $\Omega(n^{k+2}d)$ where n is the number of variables and d is the number of training examples. Therefore, in this section we will present an algorithm that uses techniques adopted from the decision tree literature to learn cutset networks.

Simply put, given training data $\mathcal{D} = \{x^1, \ldots, x^d\}$ defined over a set V of variables, we can use the following recursive or divide-and-conquer approach to learn a cutset network from $\mathcal{D}$ (see Algorithm 1). Select a variable using the

[1] Given a graph $G = (V, E)$, $C \subseteq V$ is a cutset of G if the subgraph over $V \setminus C$ is a tree.

given *splitting heuristic*, place it at the root and make one branch for each of its possible values. Repeat the approach at each branch, using only those instances that reach the branch. If at any time, some pre-defined *termination condition* is satisfied, run the Chow-Liu algorithm on the remaining variables. It is easy to see that the optimal probability value, assuming that we are using the maximum likelihood estimation principle, attached to each branch in the OR tree is the fraction of the instances at the parent that actually reach the branch.

The two main choices in the above algorithm are which variable to split on and the termination condition. We discuss each of them in turn, next, followed by the algorithm to learn mixtures of cutset networks from data.

4.1 Splitting Heuristics

Intuitively, we should split on a variable that reduces the expected entropy (or the information content) of the two partitions of the data created by the split. The hope is that when the expected entropy is small, we will be able to represent it faithfully using a simple distribution such as a tree Bayesian network. Unfortunately, unlike traditional classification problems, in which we are interested in (the entropy of) a specific class variable, estimating the joint entropy of the data when the class variable is not known is a challenging task (we just don't have enough data to reliably measure the joint entropy). Therefore, we propose to approximate the joint entropy by the average entropy over individual variables. Formally, for our purpose, the entropy of data $\mathcal{D}$ defined over a set V of variables is given by:

$$\widehat{H}(\mathcal{D}) = \frac{1}{|V|} \sum_{v \in V} H_{\mathcal{D}}(v) \tag{4}$$

where $H_{\mathcal{D}}(v)$ is the entropy of variable v relative to $\mathcal{D}$. It is given by:

$$H_{\mathcal{D}}(v) = - \sum_{x_v \in \Delta_v} P(x_v) log(P(x_v))$$

Given a closed-form expression for the entropy of the data, we can calculate the information gain or the expected reduction in the entropy after conditioning on a variable v using the following expression:

$$Gain_{\mathcal{D}}(v) = \widehat{H}(\mathcal{D}) - \sum_{x_v \in \Delta_v} \frac{|\mathcal{D}_{x_v}|}{|\mathcal{D}|} \widehat{H}(\mathcal{D}_{x_v})$$

where $\mathcal{D}_{x_v} = \{x^i \in \mathcal{D} | x^i_v = x_v\}$.

From the discussion above, the splitting heuristic is obvious: select a variable that has the highest information gain.

4.2 Termination Condition and Post-Pruning

A simple termination condition that we can enforce is stopping when the number of examples at a node falls below a fixed threshold. Alternatively, we can

also declare a node as a leaf node if the entropy falls below a threshold. Unfortunately, both of these criteria are highly dependent on the threshold used. A large threshold will yield shallow OR trees that are likely to underfit the data (high bias) while a small threshold will yield deep trees that are likely to overfit the data (high variance). To combat this, inspired by the decision tree literature [21, 22], we propose to use reduced error pruning. In reduced error pruning, we grow the tree fully and post-prune in a bottom-up fashion. (Alternatively, we can also prune in a top-down fashion).

The benefits of pruning over using a fixed threshold are that it avoids the horizon effect (the thresholding method suffers from lack of sufficient look ahead). Pruning comes at a greater computational expense than threshold based stopped splitting and therefore for problems with large training sets, the expense can be prohibitive. For small problems, though, these computational costs are low and pruning should be preferred over stopped splitting. Moreover, pruning is an anytime method and as a result we can stop it at any time.

Formally, our proposed reduced error pruning for cutset networks operates as follows. We divide the data into two sets: training data and validation data. Then, we build a full OR tree over the training data, declaring a node as a leaf node using a weak termination condition (e.g., the number of examples at a node is less than or equal to 5). Then, we recursively visit the tree in a bottom up fashion, and replace a node and the sub-tree below it by a leaf node (namely, a Chow-Liu tree) if it increases the log-likelihood on the validation set.

We summarize the time and space complexity of learning (using Algorithm 1) and inference in cutset networks in the following theorem.

Theorem 1. *The time complexity of learning cutset networks is $O(n^2ds)$ where s is the number of nodes in the cutset network, d is the number of examples and n is the number of variables. The space complexity of the algorithm is $O(ns)$, which also bounds the space required by the cutset networks. The time complexity of performing marginal and maximum-a-posteriori inference in a cutset network is $O(ns)$.*

Proof. The time complexity of computing the gain at each internal OR node is $O(n^2d)$. Similarly, the time complexity of running the Chow-Liu algorithm at each leaf node is $O(n^2d)$. Since there are s nodes in the cutset network, the overall time complexity is $O(n^2ds)$. The space required to store an OR node is $O(1)$ while the space required to store a tree Bayesian network is $O(n)$. Thus, the overall space complexity is $O(max(n, 1)s) = O(ns)$. The time complexity of performing inference at each leaf Chow-Liu node is $O(n)$ while inference at each internal OR node can be done in constant time. Since the tree has s nodes, the overall inference complexity is $O(ns)$.

4.3 Mixtures of Cutset Networks

Similar to mixtures of trees [17], we define mixtures of cutset networks as distributions of the form:

$$P(x) = \sum_{i=1}^{k} \lambda_i C_i(x) \tag{5}$$

with $\lambda_i \geq 0$ for $i = 1, \ldots, k$, and $\sum_{i=1}^{k} \lambda_i = 1$. Each mixture component $C_i(x)$ is a cutset network and λ_i is its mixture co-efficient. At a high level, one can think of the mixture as containing a latent variable z which takes a value $i \in \{1, \ldots, k\}$ with probability λ_i.

Next, we present a version of the expectaton-maximization algorithm (EM) [10] for learning mixtures of cutset networks from data. The EM algorithm operates as follows. We begin with random parameters. At each iteration t, in the expectation-step (E-step) of the algorithm, we find the probability of completing each training example, using the current model. Namely, for each training example x^j and each component i, we compute

$$P^t(z = i|x^j) = \frac{\lambda_i^t C_i^t(x^j)}{\sum_{r=1}^{k} \lambda_r^t C_r^t(x^j)}$$

Then, in the maximization-step (M-step), we learn each mixture component i, using a weighted training set in which each example j has weight $P^t(z = i|x^j)$. This yields a new mixture component C_i^{t+1}. In the M-step, we also update the mixture co-efficients using the following expression:

$$\lambda_i^{t+1} = \frac{\sum_{j=1}^{d} P^t(z = i|x^j)}{d}$$

We can run EM until it converges or until a pre-defined bound on the number of iterations is exceeded. The quality of the local maxima reached by EM is highly dependent on the initialization used and therefore in practice, we typically run EM using several different initializations and choose parameter settings having the highest log-likelihood score. Notice that by varying the number of mixture components, we can explore interesting bias versus variance tradeoffs. Large k will yield high variance models and small k will yield high bias models.

We summarize the time and space complexity of learning and inference in mixtures of cutset networks in the following theorem.

Theorem 2. *The time complexity of learning mixtures of cutset networks is $O(n^2 dskt_{max})$ where s is the number of nodes in the cutset network, d is the number of examples, k is the number of mixture components, t_{max} is the maximum number of iterations for which EM is run and n is the number of variables. The space complexity of the algorithm is $O(nsk)$, which also bounds the space required by the mixtures of cutset networks. The time complexity of performing marginal and maximum-a-posteriori inference in a mixtures of cutset networks is $O(nks)$.*

Proof. Follows from Theorem 1

Table 1. Test-set Log-Likelihood. †indicates that the results of this algorithm are not available.

Dataset	CNet	CNetP	MCNet	ID-SPN	ACMN	MT	Chow-Liu	Learn SPN	LTM
NLTCS	-6.10	-6.05	-6.00	-6.02	-6.00	-6.01	-6.76	-6.11	-6.49
MSNBC	-6.06	-6.05	-6.04	-6.04	-6.04	-6.07	-6.54	-6.11	-6.52
KDDCup2000	-2.21	-2.19	**-2.12**	-2.13	-2.17	-2.13	-2.32	-2.18	-2.18
Plants	-13.37	-13.25	-12.78	**-12.54**	-12.80	-12.95	-16.51	-12.98	-16.39
Audio	-46.84	-41.97	**-39.73**	-39.79	-40.32	-40.08	-44.35	-40.50	-41.90
Jester	-64.50	-55.26	**-52.57**	-52.86	-53.31	-53.08	-58.21	-53.48	-55.17
Netflix	-69.74	-58.72	**-56.32**	-56.36	-57.22	-56.74	-60.25	-57.33	-58.53
Accidents	-31.59	-30.66	-29.96	**-26.98**	-27.11	-29.63	-33.17	-30.04	-33.05
Retail	-11.12	-10.98	**-10.82**	-10.85	-10.88	-10.83	-11.02	-11.04	-10.92
Pumsb-star	-25.06	-24.28	-24.18	**-22.40**	-23.55	-23.71	-30.80	-24.78	-31.32
DNA	-109.79	-87.50	-85.82	-81.21	**-80.03**	-85.14	-87.70	-82.52	-87.60
Kosarek	-11.53	-11.07	**-10.58**	-10.60	-10.84	-10.62	-11.52	-10.99	-10.87
MSWeb	-10.20	-10.12	-9.79	**-9.73**	-9.77	-9.85	-10.35	-10.25	-10.21
Book	-40.19	-37.51	**-33.96**	-34.14	-35.56	-34.63	-37.84	-35.89	-34.22
EachMovie	-60.22	-57.71	**-51.39**	-51.51	-55.80	-54.60	-64.79	-52.49	†
WebKB	-171.95	-161.58	-153.22	**-151.84**	-159.13	-156.86	-164.89	-158.20	-156.84
Reuters-52	-91.35	-87.64	-86.11	**-83.35**	-90.23	-85.90	-96.85	-85.07	-91.23
20Newsgroup	-176.56	-161.68	**-151.29**	-151.47	-161.13	-154.24	-164.99	-155.93	-156.77
BBC	-300.33	-260.55	-250.58	**-248.93**	-257.10	-261.84	-261.41	-250.69	-255.76
Ad	-16.31	-16.14	-16.68	-19.00	-16.53	**-16.02**	-16.67	-19.73	†

Table 2. Runtime Comparison(in seconds). Time-limit for each algorithm: 48 hours. †indicates that the algorithm did not terminate in 48 hours.

Dataset	Var#	Train	Valid	Test	CNet	CNetP	MCNet	ID-SPN	ACMN
NLTCS	16	16181	2157	3236	0.2	0.4	36.5	307.0	242.4
MSNBC	17	291326	38843	58265	13.0	29.2	2177.7	90354.0	579.9
KDDCup2000	64	180092	19907	34955	95.9	197.8	1988.0	38223.0	645.5
Plants	69	17412	2321	3482	6.5	10.5	135.0	10590.0	119.4
Audio	100	15000	2000	3000	17.2	19.6	187.0	14231.0	1663.9
Jester	100	9000	1000	4116	14.0	11.8	101.2	†	3665.8
Netflix	100	15000	2000	3000	25.2	22.6	224.4	†	1837.4
Accidents	111	12758	1700	2551	15.7	22.1	195.4	†	793.4
Retail	135	22041	2938	4408	18.9	27.6	104.7	2116.0	12.5
Pumsb-star	163	12262	1635	2452	30.1	41.8	233.8	18219.0	374.0
DNA	180	1600	400	1186	13.8	6.9	57.7	150850.0	39.9
Kosarek	190	33375	4450	6675	65.9	102.5	141.2	†	585.4
MSWeb	294	29441	32750	5000	208.6	365.8	642.8	†	286.3
Book	500	8700	1159	1739	129.1	204.2	154.4	125480.0	3035.0
EachMovie	500	4524	1002	591	90.7	133.4	204.8	78982.0	9881.1
WebKB	839	2803	558	838	169.7	228.7	160.4	†	7098.3
Reuters-52	889	6532	1028	1540	397.1	650.4	1177.2	†	2709.6
20Newsgroup	910	11293	3764	3764	695.2	935.8	1525.2	†	16255.3
BBC	1058	1670	225	330	206.7	223.9	70.2	4157.0	1862.2
Ad	1556	2461	327	491	365.8	594.3	155.4	285324.0	6496.4

5 Empirical Evaluation

The aim of our experimental evaluation is two fold: comparing the speed, measured in terms of CPU time, and accuracy, measured in terms of test-set log likelihood scores, of our methods with state-of-the-art methods for learning tractable models.

5.1 Methodology and Setup

We evaluated our algorithms as well as the competition on 20 benchmark datasets shown in Table 2. The number of variables in the datasets ranged from 16 to 1556, and the number of training examples varied from 1.6K to 291K examples. All variables in our datasets are binary-valued for a fair comparison with other methods, who operate primarily on binary-valued input. These datasets or a subset of them have also been used by [8, 16, 15, 11, 24].

We implemented three variations of our algorithms: (1) learning CNets without pruning (CNet), (2) learning CNets with pruning (CNetP) and (3) mixtures of CNets (MCNets). We compared their performance with the following learning algorithms from literature: learning sum-product networks with direct and indirect interactions (ID-SPN) [23], learning Markov networks using arithmetic

Table 3. Head-to-head comparison of the number of wins (in terms of test-set log-likelihood score) achieved by one algorithm (row) over another (column), for all pairs of the 6 best performing algorithms used in our experimental study.

	CNetP	MCNet	ID-SPN	ACMN	MT	LearnSPN
CNetP	-	1	1	2	2	6
MCNet	19	-	11	13	15	18
ID-SPN	19	8	-	16	16	20
ACMN	18	5	3	-	8	15
MT	18	5	3	12	-	16
LearnSPN	14	2	0	5	4	-

circuits (ACMN) [16], learning mixture of trees (MT) [17], Chow-Liu trees [6], learning Sum-Product Networks (LearnSPN) [11] and learning latent tree models (LTM) [5]. Most of the results on the datasets (except the results on learning Chow-Liu models) were made available to us by [23]. They are part of the Libra toolkit available on Daniel Lowd's web page.

We smoothed all parameters using 1-laplace smoothing. For learning CNets without pruning, we stopped building the OR tree when the number of examples at the leaf node were fewer than 10 or the total entropy was smaller than 0.01. To learn MCNets, we varied the number of components from 5 to 40, in increments of 5 and ran the EM algorithm for 100 iterations or convergence whichever was earlier. For each iteration of EM, we could update both the structure and the parameters of the cutset network associated with each component. However, to speed up the learning algorithm, we chose to update just the parameters, utilizing the structure learned at the first iteration.

5.2 Accuracy

Table 1 shows the test-set log likelihood scores for the various benchmark networks while Table 3 shows head-to-head comparison of the six best performing algorithms namely CNetP, MCNet, ID-SPN, ACMN, MT and LearnSPN. Excluding the first two datasets where there are multiple winners, we can see that MCNet has the best log-likelihood score on 9 out of the remaining 18 benchmarks, while ID-SPN is the second best performing algorithm, with the best log-likelihood score on 7 out of the 18 benchmarks. In the head-to-head comparison, MCNet is better than CNetP on 19 benchmarks, ID-SPN on 11 benchmarks, ACMN on 13 benchmarks, MT on 15 benchmarks and LearnSPN on 18 benchmarks. CNetP is better than ID-SPN only on 1 benchmark, ACMN and MT on 2 benchmarks while it is better than LearnSPN on 6 benchmarks. A careful look at the datasets reveal that when the number of training examples is large, MCNet and to some extent CNetP are typically better than the competition. However, for small training set sizes, ID-SPN is the best performing algorithm. As expected, Chow-Liu trees and CNet are the worst-performing algorithms, the former underfits and the latter overfits.

MCNet is consistently better than CNetP which suggests that whenever possible *it is a good idea to use latent mixtures of simple models*. This conclusion can also be drawn from the performance of MT, which greatly improves the accuracy of Chow-Liu trees.

5.3 Learning Time

Table 2 shows the time taken by CNet, CNetP, MCNet, ID-SPN and ACMN to learn a model from data. We gave a time limit of 48 hours to all algorithms and ran all our timing experiments on a quad-core Intel i7, 2.7 GHz machine with 8GB of RAM. The fastest cutset network learners, in order, are: CNet, CNetP, and MCNet. On an average, ACMN is slower than MCNet. ID-SPN is the slowest algorithm. In fact, ID-SPN did not finish on 8 out of the 20 datasets in 48 hours (note that for the datasets on which ID-SPN did not finish in 48 hours, we report the test set log-likelihood scores from [23]). The best performing cutset network algorithm, MCNet, was faster than ID-SPN on all 20 datasets and ACMN on 14 datasets. If we look at the learning time and accuracy as a whole, CNetP is the best performing algorithm, providing reasonably accurate results in quick time.

6 Summary and Future Work

In this paper we presented cutset networks - a novel, simple and tractable probabilistic graphical model. At a high level, cutset networks are operational representation of Pearl's cutset conditioning method, with an OR tree modeling conditioning (at the top) and a tree Bayesian network modeling inference over trees at the leaves. We developed an efficient algorithm for learning cutset networks from data. Our new algorithm uses a decision-tree inspired learning algorithm for inducing the structure and parameters of the OR tree and the classic Chow-Liu algorithm for learning the tree distributions at the leaf nodes. We also presented an EM-based algorithm for learning mixtures of cutset networks.

Our detailed experimental study on a variety on benchmark datasets clearly demonstrated the power of cutset networks. In particular, our new algorithm that learns mixtures of cutset networks from data, was the best performing algorithm in terms of log-likelihood score on 55% of the benchmarks when compared with 5 other state-of-the-art algorithms from literature. Moreover, our new *one-shot algorithm*, which builds a cutset network using the information gain heuristic and employs reduced-error pruning is not only fast (as expected) but also reasonably accurate on several benchmark datasets. This gives us a spectrum of algorithms for future investigations: fast, accurate one-shot algorithms and slow, highly accurate iterative algorithms based on EM.

Future work includes learning polytrees having at most w parents at the leaves yielding w-cutset networks; using AND/OR trees or sum-product networks instead of OR trees yielding AND/OR cutset networks [9]; introducing structured latent variables in the mixture model; and merging identical subtrees while learning to yield a compact graph-based representation.

Acknowledgements This research was partly funded by ARO MURI grant W911NF-08-1-0242, by the AFRL under contract number FA8750-14-C-0021 and by the DARPA Probabilistic Programming for Advanced Machine Learning Program under AFRL prime contract number FA8750-14-C-0005. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, AFRL, ARO or the US government.

References

1. Bach, F., Jordan, M.: Thin junction trees. Advances in Neural Information Processing Systems 14, 569–576 (2001)
2. Boutilier, C., Friedman, N., Goldszmidt, M., Koller, D.: Context-specific independence in Bayesian networks. In: Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence. pp. 115–123. Morgan Kaufmann, Portland, OR (1996)
3. Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. Artificial Intelligence 172(6-7), 772–799 (2008)
4. Chechetka, A., Guestrin, C.: Efficient principled learning of thin junction trees. In: Platt, J., Koller, D., Singer, Y., Roweis, S. (eds.) Advances in Neural Information Processing Systems 20. MIT Press, Cambridge, MA (2008)
5. Choi, M.J., Tan, V., Anandkumar, A., Willsky, A.: Learning latent tree graphical models. Journal of Machine Learning Research 12, 1771–1812 (May 2011)
6. Chow, C.K., Liu, C.N.: Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory 14, 462–467 (1968)
7. Darwiche, A.: Modeling and reasoning with Bayesian networks. Cambridge University Press (2009)
8. Davis, J., Domingos, P.: Bottom-up learning of Markov network structure. In: Proceedings of the Twenty-Seventh International Conference on Machine Learning. pp. 271–278. ACM Press, Haifa, Israel (2010)
9. Dechter, R., Mateescu, R.: AND/OR search spaces for graphical models. Artificial Intelligence 171(2), 73–106 (2007)
10. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society, Series B 39, 1–38 (1977)
11. Gens, R., Domingos, P.: Learning the structure of sum-product networks. In: Proceedings of the Thirtieth International Conference on Machine Learning. JMLR: W&CP 28 (2013)
12. Gogate, V., Domingos, P.: Formula-Based Probabilistic Inference. In: Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence. pp. 210–219 (2010)
13. Gogate, V., Webb, W., Domingos, P.: Learning efficient Markov networks. In: Proceedings of the 24th conference on Neural Information Processing Systems (NIPS'10) (2010)
14. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press, Cambridge, MA (2009)
15. Lowd, D., Davis, J.: Learning Markov network structure with decision trees. In: Proceedings of the 10th IEEE International Conference on Data Mining (ICDM). pp. 334–343. IEEE Computer Society Press, Sydney, Australia (2010)

16. Lowd, D., Rooshenas, A.: Learning Markov networks with arithmetic circuits. In: Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2013). Scottsdale, AZ (2013)
17. Meila, M., Jordan, M.: Learning with mixtures of trees. Journal of Machine Learning Research 1, 1–48 (2000)
18. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York, NY (1997)
19. Narasimhan, M., Bilmes, J.: Pac-learning bounded tree-width graphical models. In: Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (2004)
20. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco, CA (1988)
21. Quinlan, J.R.: Induction of decision trees. Machine Learning 1, 81–106 (1986)
22. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA (1993)
23. Rooshenas, A., Lowd, D.: Learning sum-product networks with direct and indirect interactions. In: Proceedings of the Thirty-First International Conference on Machine Learning. ACM Press, Beijing, China (2014)
24. Van Haaren, J., Davis, J.: Markov network structure learning: A randomized feature generation approach. In: Proceedings of the Twenty-Sixth National Conference on Artificial Intelligence. AAAI Press (2012)