# On Probabilistic Inference by Weighted Model Counting [⋆]

Mark Chavira, Adnan Darwiche

*Computer Science Department, UCLA, Los Angeles, CA 90095*

## Abstract

A recent and effective approach to probabilistic inference calls for reducing the problem to one of weighted model counting (WMC) on a propositional knowledge base. Specifically, the approach calls for encoding the probabilistic model, typically a Bayesian network, as a propositional knowledge base in conjunctive normal form (CNF) with weights associated to each model according to the network parameters. Given this CNF, computing the probability of some evidence becomes a matter of summing the weights of all CNF models consistent with the evidence. A number of variations on this approach have appeared in the literature recently, that vary across three orthogonal dimensions. The first dimension concerns the specific encoding used to convert a Bayesian network into a CNF. The second dimensions relates to whether weighted model counting is performed using a search algorithm on the CNF, or by compiling the CNF into a structure that renders WMC a polytime operation in the size of the compiled structure. The third dimension deals with the specific properties of network parameters (local structure) which are captured in the CNF encoding. In this paper, we discuss recent work in this area across the above three dimensions, and demonstrate empirically its practical importance in significantly expanding the reach of exact probabilistic inference. We restrict our discussion to exact inference and model counting, even though other proposals have been extended for approximate inference and approximate model counting.

*Key words:* Bayesian Networks, Exact Inference, Weighted Model Counting, Compilation

[⋆] Some of the work discussed in this paper is based on [1–3].

*Email addresses:* `chavira@cs.ucla.edu` (Mark Chavira), `darwiche@cs.ucla.edu` (Adnan Darwiche).

## 1 Introduction

Standard algorithms [4–8] for exact inference on Bayesian networks exploit only topological structure and are known to be $\Theta(n2^w)$, where $n$ is the number of network variables, and $w$ is the *treewidth* of the network. That is, they are both worst case and best case bounded exponentially by treewidth. In recent years, the types of problems considered in probabilistic reasoning have often yielded networks with large treewidths, calling into question the applicability of exact inference methods, and shifting interest more towards approximate inference algorithms. It has long been believed though that a network exhibiting high treewidth may not necessarily be difficult for exact inference, if it also exhibits a certain amount of local structure in the form of determinism [9] and context–specific independence (CSI) [10]. A typical algorithm that can exploit both topological and local structure is $O(n2^w)$. That is, only its worst case is exponential in treewidth (e.g., when no local structure exists), but where there is sufficient local structure, its performance can be significantly better.

A number of approaches have been proposed in the literature for exploiting local structure, e.g., [11,12,10,13–15,1,16,9,17]. Our aim in this paper is to survey and provide further results on a particular class of these approaches, which works by reducing the problem of probabilistic inference into one of weighted model counting (WMC) on a propositional knowledge base. In particular, the approach commences by encoding the Bayesian network into a knowledge base in conjunctive normal form (CNF), and then assigns weights to the CNF variables based on the network probabilities. These assignments of weights to variables induce a weight for each CNF model, allowing one to represent the probability of some evidence as the sum of weights for models consistent with the evidence. A number of variations on this model counting approach have appeared in the literature recently, and they vary across three orthogonal dimensions. The first dimension concerns the specific encoding used to convert a Bayesian network into a CNF. The second dimension relates to whether WMC is performed using a search algorithm on the CNF [18,19,16], or by compiling the CNF into a structure that renders WMC a polytime operation in the size of the compiled structure [15,20]. [1] The third dimension deals with the specific properties of network parameters (local structure) which are captured in the CNF encoding.

Probabilistic inference by WMC can be powerful for several reasons. First, the encoding of Bayesian networks into logical knowledge bases presents opportunities to effectively reveal local structure in the form of determinism and

[1] This does not escape the complexity of WMC since the compilation and size of resulting structure can in the worst case be exponential in the treewidth of the network.

context–specific independence. Second, it provides a very natural framework for exploiting the computational power of evidence [2]. Finally, it can leverage highly refined techniques for solving satisfiability, which form the basis for both search and knowledge compilation approaches to WMC. Such techniques are especially effective, as we shall see, on encodings of Bayesian networks that contain large amounts of determinism. In recent years, these advantages have lead to several breakthroughs in exact probabilistic inference that have challenged common perceptions of what exact inference can do. In this paper, we present a survey of these works by placing them along the three dimensions discussed earlier, and then demonstrate empirically their practical importance in significantly expanding the reach of exact inference.

The remainder of this paper is organized as follows. In Section 2, we provide background material. The paper then addresses each of the three orthogonal dimensions that differentiate approaches that utilize WMC: encodings in Section 3, search vs. knowledge compilation in Section 4, and local structure in Section 5. We briefly discuss what happens to WMC techniques in the absence of local structure in Section 6. The paper ends with conclusions in Section 7.

## 2 Probabilistic Inference by WMC

In this section, we review some fundamental concepts from probabilistic inference, show one way to reduce probabilistic inference to WMC, and provide some historical perspective.

### *2.1 Probabilistic inference*

Within the area of probabilistic reasoning, one typically models a situation as a joint distribution on a set of random variables and uses this distribution to answer probabilistic queries. In this paper, all variables have finite domains. Suppose that $\mathbf{X}$ is a set of variables.[2] An *instantiation* of $\mathbf{X}$ is an assignment to each $X \in \mathbf{X}$ of some value in $X$'s domain. *Evidence* on $\mathbf{X}$ is similar but need assign only some of the variables in $\mathbf{X}$. A *joint distribution* over $\mathbf{X}$ is a function $\mathsf{Pr}$ that maps each instantiation of $\mathbf{X}$ to a probability in $[0, 1]$ such that the sum of the probabilities is equal to 1. For example, suppose that we

[2] We are using the standard notation: variables are denoted by upper–case letters ($A$) and their values by lower–case letters ($a$). Sets of variables are denoted by bold–face upper–case letters ($\mathbf{A}$) and their instantiations are denoted by bold–face lower–case letters ($\mathbf{a}$). For a variable $A$ with values $\mathsf{false}$ and $\mathsf{true}$, we use $\bar{a}$ to denote $A$=$\mathsf{false}$ and $a$ to denote $A$=$\mathsf{true}$.

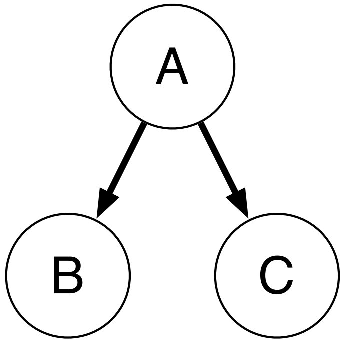
Table 1
A joint distribution over three variables.

| $A$ | $B$ | C | $Pr$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | 0.001 |
| $a_1$ | $b_1$ | $c_2$ | 0.002 |
| $a_1$ | $b_1$ | $c_3$ | 0.007 |
| $a_1$ | $b_2$ | $c_1$ | 0.009 |
| $a_1$ | $b_2$ | $c_2$ | 0.018 |
| $a_1$ | $b_2$ | $c_3$ | 0.063 |
| $a_2$ | $b_1$ | $c_1$ | 0.0018 |
| $a_2$ | $b_1$ | $c_2$ | 0.0162 |
| $a_2$ | $b_1$ | $c_3$ | 0.162 |
| $a_2$ | $b_2$ | $c_1$ | 0.0072 |
| $a_2$ | $b_2$ | $c_2$ | 0.0648 |
| $a_2$ | $b_2$ | $c_3$ | 0.648 |

are dealing with variables $\mathbf{X} = \{A, B, C\}$, where $A$ and $B$ have two values, and $C$ has three values. Then Table 1 depicts one possible joint distribution on $\mathbf{X}$.

An important query one might wish to answer with respect to the joint is the probability of evidence $\mathbf{e}$, denoted $\mathsf{Pr}(\mathbf{e})$. To answer this query, we first remove rows from the joint that contradict $\mathbf{e}$, and then sum the remaining probabilities. For example, using Table 1 as the model, one computes $\mathsf{Pr}(a_1, c_2)$ by summing the two consistent instantiations (second and fifth rows) for a result of 0.02. Although the joint distribution contains the information necessary to answer queries, its size is exponential in the number of variables. As a result, there is a need to specify a joint compactly. One common class of modeling language is *probabilistic graphical models*, which include Markov networks and Bayesian networks [21]. This paper will focus on the popular Bayesian network, but much of the description applies to other modeling languages, including Markov networks and ground instances of some first order probabilistic models [22].

A Bayesian network represents a specific joint distribution and is a pair $(G, P)$, where $G$ is a directed–acyclic graph and $P$ is a set of *factors*. The nodes in $G$ are the variables $\mathbf{X}$ in the joint, and the edges in $G$ imply certain probabilistic independence relationships among the variables. For each $X \in \mathbf{X}$ with parents $\mathbf{U}$, $P$ contains a factor $f$, which is a function over instantiations of $\mathbf{U} \cup \{X\}$ such that $f(\mathbf{u}, x) = \mathsf{Pr}(x \mid \mathbf{u})$. We refer to $X$ and its parents as a



| $A$ | $Pr$ |
|---|---|
| $a_1$ | 0.1 |
| $a_2$ | 0.9 |

| $A$ | $B$ | $Pr$ |
|---|---|---|
| $a_1$ | $b_1$ | 0.1 |
| $a_1$ | $b_2$ | 0.9 |
| $a_2$ | $b_1$ | 0.2 |
| $a_2$ | $b_2$ | 0.8 |

| $A$ | $C$ | $Pr$ |
|---|---|---|
| $a_1$ | $c_1$ | 0.1 |
| $a_1$ | $c_2$ | 0.2 |
| $a_1$ | $c_3$ | 0.7 |
| $a_2$ | $c_1$ | 0.01 |
| $a_2$ | $c_2$ | 0.09 |
| $a_2$ | $c_3$ | 0.9 |

Fig. 1. A small Bayesian network specifying the joint in Table 1.

*network family*. The semantics of Bayesian networks imply the following joint distribution:

$$\mathsf{Pr}(x_1, x_2, \ldots, x_k) = \prod_i \mathsf{Pr}(x_i \mid \mathbf{u}_i) \tag{1}$$

where $\mathbf{u}_i$ is the instantiation of $X_i$'s parents as they appear in $x_1, x_2, \ldots, x_k$. Figure 1 depicts a simple Bayesian network, which induces the joint distribution in Table 1.

A factor is simply a function and can be represented in various ways. Very often, a factor is represented as a conditional probability table (CPT), several of which are shown in Figure 1. In such cases, we refer to each table entry $\mathsf{Pr}(x \mid \mathbf{u})$ as a *network parameter*. In addition to tables, other representations of factors are also possible, including decision trees/graphs, noisy or/and/min/max, and logical rules. In this paper, we deal only with CPTs, although WMC techniques can also be very powerful in the context of other representations.

### *2.2 Probabilistic Inference as WMC*

In this section, we provide an example that illustrates how computing the probability of evidence for a particular Bayesian network can be reduced to WMC. To generate an instance of the WMC problem, one needs to define a logical theory $\Delta$ and to assign a weight $W(\ell)$ to each literal $\ell$. The weights for the literals induce a weight for each model $\omega$ of $\Delta$ as follows:

$$W(\omega) = \prod_{\omega \models \ell} W(\ell)$$

Finally, computing WMC for $\Delta$, which we shall denote $WMC(\Delta)$, is computing the sum of the weights of all models of $\Delta$:

$$WMC(\Delta) = \sum_{\omega \models \Delta} W(\omega)$$

The reduction scheme described in this section is based on the one proposed in [15]. Given a Bayesian network and evidence, the scheme constructs a weighted knowledge base $\Delta$, whose weighted model count corresponds to the probability of evidence with respect to the network. An intuition for the scheme can be attained by relating the models of $\Delta$ to the rows of the joint distribution of the network. In particular, the models will be in one–to–one correspondence with the rows, and the variable weights will be assigned in such a way that each model will have a weight equal to the probability of the corresponding row in the joint. Weighted counting of the models is then equivalent to summing all probabilities in the joint. The last step required to compute probability of evidence is a way to exclude appropriate models from being counted. We provide details of this reduction scheme next.

Consider again the Bayesian network $N$ in Figure 1 and suppose that we wish to compute the probability of evidence $\mathbf{e} = \{a_1, c_2\}$ with respect to this network. To perform the reduction, we require five tasks.

**Define logical variables:** The first task is to define the variables that will be used in the logic. For each value $x$ of each network variable $X$, we define an indicator variable $\lambda_x$ in the CNF. For network $N$, we obtain the following indicator variables:

Variable A: $\lambda_{a_1}$, $\lambda_{a_2}$

Variable B: $\lambda_{b_1}$, $\lambda_{b_2}$

Variable C: $\lambda_{c_1}$, $\lambda_{c_2}$, $\lambda_{c_3}$

Table 2
Variable instantiations for the network in Figure 1; definitions for the corresponding models; weights without evidence; and the weights after incorporating evidence $\mathbf{e} = a_1, c_2$.

| Network | Sets these CNF vars to | Weight | Weight |
|---|---|---|---|
| Instantiation | $\mathsf{true}$ and all others to $\mathsf{false}$ | without $\mathbf{e}$ | with $\mathbf{e}$ |
| $a_1b_1c_1$ | $\lambda_{a_1}\ \lambda_{b_1}\ \lambda_{c_1}\ \theta_{a_1}\ \theta_{b_1\|a_1}\ \theta_{c_1\|a_1}$ | $0.1 \cdot 0.1 \cdot 0.10 = 0.001$ | 0 |
| $a_1b_1c_2$ | $\lambda_{a_1}\ \lambda_{b_1}\ \lambda_{c_2}\ \theta_{a_1}\ \theta_{b_1\|a_1}\ \theta_{c_2\|a_1}$ | $0.1 \cdot 0.1 \cdot 0.20 = 0.002$ | 0.002 |
| $a_1b_1c_3$ | $\lambda_{a_1}\ \lambda_{b_1}\ \lambda_{c_3}\ \theta_{a_1}\ \theta_{b_1\|a_1}\ \theta_{c_3\|a_1}$ | $0.1 \cdot 0.1 \cdot 0.70 = 0.007$ | 0 |
| $a_1b_2c_1$ | $\lambda_{a_1}\ \lambda_{b_2}\ \lambda_{c_1}\ \theta_{a_1}\ \theta_{b_2\|a_1}\ \theta_{c_1\|a_1}$ | $0.1 \cdot 0.9 \cdot 0.10 = 0.009$ | 0 |
| $a_1b_2c_2$ | $\lambda_{a_1}\ \lambda_{b_2}\ \lambda_{c_2}\ \theta_{a_1}\ \theta_{b_2\|a_1}\ \theta_{c_2\|a_1}$ | $0.1 \cdot 0.9 \cdot 0.20 = 0.018$ | 0.018 |
| $a_1b_2c_3$ | $\lambda_{a_1}\ \lambda_{b_2}\ \lambda_{c_3}\ \theta_{a_1}\ \theta_{b_2\|a_1}\ \theta_{c_3\|a_1}$ | $0.1 \cdot 0.9 \cdot 0.70 = 0.063$ | 0 |
| $a_2b_1c_1$ | $\lambda_{a_2}\ \lambda_{b_1}\ \lambda_{c_1}\ \theta_{a_2}\ \theta_{b_1\|a_2}\ \theta_{c_1\|a_2}$ | $0.9 \cdot 0.2 \cdot 0.01 = 0.0018$ | 0 |
| $a_2b_1c_2$ | $\lambda_{a_2}\ \lambda_{b_1}\ \lambda_{c_2}\ \theta_{a_2}\ \theta_{b_1\|a_2}\ \theta_{c_2\|a_2}$ | $0.9 \cdot 0.2 \cdot 0.09 = 0.0162$ | 0 |
| $a_2b_1c_3$ | $\lambda_{a_2}\ \lambda_{b_1}\ \lambda_{c_3}\ \theta_{a_2}\ \theta_{b_1\|a_2}\ \theta_{c_3\|a_2}$ | $0.9 \cdot 0.2 \cdot 0.90 = 0.162$ | 0 |
| $a_2b_2c_1$ | $\lambda_{a_2}\ \lambda_{b_2}\ \lambda_{c_1}\ \theta_{a_2}\ \theta_{b_2\|a_2}\ \theta_{c_1\|a_2}$ | $0.9 \cdot 0.8 \cdot 0.01 = 0.0072$ | 0 |
| $a_2b_2c_2$ | $\lambda_{a_2}\ \lambda_{b_2}\ \lambda_{c_2}\ \theta_{a_2}\ \theta_{b_2\|a_2}\ \theta_{c_2\|a_2}$ | $0.9 \cdot 0.8 \cdot 0.09 = 0.0648$ | 0 |
| $a_2b_2c_3$ | $\lambda_{a_2}\ \lambda_{b_2}\ \lambda_{c_3}\ \theta_{a_2}\ \theta_{b_2\|a_2}\ \theta_{c_3\|a_2}$ | $0.9 \cdot 0.8 \cdot 0.90 = 0.648$ | 0 |

In addition, for each parameter $\mathsf{Pr}(x|\mathbf{u})$ in the Bayesian network, we define a corresponding parameter variable $\theta_{x|\mathbf{u}}$ in the CNF. For network $N$, this step yields the following parameter variables:

$$
\begin{aligned}
\text{CPT 1:} &\quad \theta_{a_1}, \theta_{a_2} \\
\text{CPT 2:} &\quad \theta_{b_1|a_1}, \theta_{b_2|a_1}, \theta_{b_1|a_2}, \theta_{b_2|a_2} \\
\text{CPT 3:} &\quad \theta_{c_1|a_1}, \theta_{c_2|a_1}, \theta_{c3|a_1}\theta_{c_1|a_2}, \theta_{c_2|a_2},\ \theta_{c_3|a_2}
\end{aligned}
$$

**Define the KB semantics:** The second task is to define the knowledge base $\Delta$ that will represent $N$. The models of $\Delta$ are in one–to–one correspondence with the instantiations of the network variables. If $\mathbf{w}$ is an instantiation of the network variables, then the corresponding model sets to $\mathsf{true}$ each CNF variable whose subscript is consistent with $\mathbf{w}$ and sets all other CNF variables to $\mathsf{false}$. For the network $N$, the first two columns of Table 2 show instantiations of network variables and definitions for the corresponding models.

**Define the CNF:** In general, it will not be possible to define $\Delta$ by listing models and instead we will generate a CNF that represents $\Delta$. As we shall see, obtaining a CNF that represents $\Delta$ can be done by processing each network variable and each parameter. We will be describing CNF encodings of a Bayesian network in more detail below. For now, we simply present one of these encodings for network $N$ without describing how it was generated:

$$\begin{array}{rll}
\text{Variable A:} & \lambda_{a_1} \vee \lambda_{a_2} & \neg\lambda_{a_1} \vee \neg\lambda_{a_2} \\
\text{Variable B:} & \lambda_{b_1} \vee \lambda_{b_2} & \neg\lambda_{b_1} \vee \neg\lambda_{b_2} \\
\text{Variable C:} & \lambda_{c_1} \vee \lambda_{c_2} \vee \lambda_{c_3} & \neg\lambda_{c_1} \vee \neg\lambda_{c_2} \\
 & & \neg\lambda_{c_1} \vee \neg\lambda_{c_3} \\
 & & \neg\lambda_{c_2} \vee \neg\lambda_{c_3} \\
\text{CPT 1:} & \lambda_{a_1} \Leftrightarrow \theta_{a_1} & \lambda_{a_2} \Leftrightarrow \theta_{a_2} \\
\text{CPT 2:} & \lambda_{a_1} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1|a_1} & \lambda_{a_2} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1|a_2} \\
 & \lambda_{a_1} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2|a_1} & \lambda_{a_2} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2|a_2} \\
\text{CPT 3:} & \lambda_{a_1} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1|a_1} & \lambda_{a_2} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1|a_2} \\
 & \lambda_{a_1} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2|a_1} & \lambda_{a_2} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2|a_2} \\
 & \lambda_{a_1} \wedge \lambda_{c_3} \Leftrightarrow \theta_{c_3|a_1} & \lambda_{a_2} \wedge \lambda_{c_3} \Leftrightarrow \theta_{c_3|a_2}
\end{array}$$

**Assign weights:** The fourth task requires that we associate a weight $W(\ell)$ with each CNF literal $\ell$. With each positive literal of a parameter variable, we associate a weight equal to the corresponding parameter, and with all other literals, we associate a weight of 1. That is, $W(\theta_{x|\mathbf{u}}) = \mathsf{Pr}(x|\mathbf{u})$, $W(\neg\theta_{x|\mathbf{u}}) = 1$, $W(\lambda_x) = 1$, and $W(\neg\lambda_x) = 1$. For the CNF variables defined for network $N$, all variable weights therefore become 1 except the following:

$$\begin{array}{lll}
W(\theta_{a_1}) = 0.1 & W(\theta_{a_2}) = 0.9 & \\
W(\theta_{b_1|a_1}) = 0.1 & W(\theta_{b_2|a_1}) = 0.9 & \\
W(\theta_{b_1|a_2}) = 0.2 & W(\theta_{b_2|a_2}) = 0.8 & \\
W(\theta_{c_1|a_1}) = 0.1 & W(\theta_{c_2|a_1}) = 0.2 & W(\theta_{c_3|a_1}) = 0.7 \\
W(\theta_{c_1|a_2}) = 0.01 & W(\theta_{c_2|a_2}) = 0.09 & W(\theta_{c_3|a_2}) = 0.9
\end{array}$$

Given this assignment of weights to variables, each model $\omega$ assumes a weight that is the product of the weights of the positive parameter literals in $\omega$ (other literals in $\omega$ have weight 1). The third column of Table 2 shows the weights of models for network $N$. An important observation is that each model is now guaranteed to have weight that is equal to the probability of the corresponding row in the joint distribution (compare weights in Table 2 to probabilities in Table 1).

**Incorporating evidence and computing WMC:** At this point, $\Delta$ does not incorporate evidence, and computing $WMC(\Delta)$ is equivalent to summing all probabilities in the joint distribution, which would yield an answer of 1.0. Recall that to compute the probability of evidence, one removes from the joint those rows that contradict the evidence before summing. There are two ways we can incorporate the evidence into our WMC computation. First, we can change the weight associated with each indicator $\lambda_x$ whose subscript contradicts the evidence from 1 to 0. The result of this change for network $N$ and evidence $\{a_1, c_2\}$ is shown in the fourth column of Table 2. Setting the indicators in this way has the effect of zeroing out those rows that are inconsistent with the evidence. Now $\mathsf{Pr}(\mathbf{e}) = WMC(\Delta) = 0.002 + 0.018 = 0.02$. A second way to eliminate the proper rows from the computation is to remove from the theory being counted those models that contradict the evidence. We do so by computing $\mathsf{Pr}(\mathbf{e}) = WMC(\Delta \wedge \Delta_e) = 0.02$, where $\Delta_e$ is a conjunction of indicators that encodes the evidence; in the case where $\mathbf{e} = \{a_1, c_2\}$, $\Delta_e = \lambda_{a_1} \wedge \lambda_{c_2}$. Each of the two methods has advantages, which will be discussed later in the paper.

### *2.3 Related Work*

The relationship between probabilistic inference and WMC was highlighted in [23], where it was shown that, like model counting, exact probabilistic inference is #P–complete (however [23] did not address weights directly). This connection was further discussed in [24], which shows, among other things, that a DPLL–style [25] SAT solver can be adapted to solve both Bayesian network inference and model counting by adding memoization.

An approach to probabilistic inference based on compiling a CNF encoding of a Bayesian network and then performing polytime WMC (in the size of the compiled representation) was introduced in [15], although this work did not use the term *weighted model counting*. By making use of determinism, this approach was shown in [15,22] to perform exact inference very efficiently on many networks having large amounts of determinism, in spite of very large treewidth. The encoding was later enhanced to utilize other types of local structure [1], allowing it to be effective on a larger class of networks (e.g., ones having lesser amounts of determinism). The ability of WMC to capitalize on available evidence while computing marginals was introduced in [2]. Here, WMC by compilation was shown to efficiently solve many problems with evidence, where they could not be compiled without evidence and where they could not be solved by algorithms that exploit only topological structure, with or without evidence, even after applying classical evidence–exploitation techniques. Methods for increasing the decomposability of the generated CNF by applying structured resolution were described in [3]. The increase in decomposability allowed many networks to be compiled for the first time and significantly decreased the time and space required to compile networks that were accessible previously. Many of the algorithms for this compilation approach are implemented in a publicly available tool called Ace,[3] which uses the c2d compiler [26].[4]

A system described in [16] leverages a state–of–the–art model counter called Cachet[5] to perform inference in a Bayesian network. This work uses a search algorithm rather than compilation and, like [15], performs WMC on a CNF constructed from a Bayesian network. Among the novel aspects of this work is a new encoding of the network into CNF, which is smaller than the encoding described in [15] in many cases. This search–based algorithm has also been able to significantly outperform algorithms that exploit only topological structure on many networks with large amounts of determinism.

Classical approaches to probabilistic inference, based on conditioning and variable elimination, have also been extended to exploit local structure. For example, a class of conditioning algorithms [8,27,12] proceed in a way that is very similar to that of DPLL–style SAT solvers and model counters that use component analysis. A clear connection between these types of algorithms was made in [12], which also describes how to incorporate additional advances from SAT solvers, including *nogood* learning and unit propagation, into an algorithm called value elimination. In this approach, search is performed on instantiations of network variables, and the logical techniques, many of which are also used by WMC algorithms, are utilized to capitalize on determinism and context–specific independence. The approach allows for dynamic ordering of variables, has the same guarantees as algorithms that exploit only topological structure, outperforms such algorithms in many circumstances, and is capable of trading time for space, similar to recursive conditioning [8]. In a sense, this class of conditioning approaches is quite similar in spirit to the WMC approaches we discuss in this paper, as they are both based on top-down, recursive conditioning and decomposition techniques. Yet, the WMC approaches we discuss have empirically appeared to be more scalable in general, as they operate directly on CNF representations, putting them at a greater advantage when exploiting satisfiability-based techniques.

Another class of approaches for exploiting local structure is based on variable elimination algorithms (and the closely related jointree algorithm). For example, the technique of zero–compression can be used to exploit determinism in the jointree framework [9], yet this technique requires one to initially perform inference on the jointree before it is zero–compressed. Hence, the technique initially takes time and space exponential in the network treewidth, potentially realizing savings in future runs, but making many of our data sets inaccessible

[3] Available for download at http://reasoning.cs.ucla.edu/ace.
[4] Available for download at http://reasoning.cs.ucla.edu/c2d.
[5] Available for download at http://www.cs.rochester.edu/u/kautz/Cachet.

to this method. More sophisticated extensions to variable elimination algorithms have also been proposed recently, including [11], [14], and [17], which propose non-tabular representations of factors, to avoid the exponential complexity in treewidth. Significantly, [17] is a compilation approach that is able to compile many networks in much less time than WMC compilation techniques described later, while matching WMC online performance. Yet, all of the networks on which these approaches (and zero–compression) have reported are solvable by methods that are exponential in network treewidth.

A third, more specialized, class of algorithms for exploiting local structure include Quickscore [13] and SUPERLINK.[6] These algorithms have demonstrated exponential improvements over classical algorithms by solving problems that have very large treewidths. The Quickscore algorithm applies only to a particular class of networks (two-level, noisy-or networks), and SUPERLINK includes domain-specific techniques for genetic linkage networks. WMC techniques were shown in [2] to be more efficient than Quickscore, and in many cases significantly more efficient than SUPERLINK. We repeat some of these empirical comparisons in Section 5.3. Another example of an algorithm for exploiting local structure, which is able to handle networks with very high treewidth, is value elimination [12]. As mentioned earlier, this algorithm is closest to the WMC algorithms we discuss in this paper. However, the WMC approach based on Cachet was shown to dominate value elimination on a wide class of networks [16].

Any method that is capable of approximating model counting can potentially be applied to approximate answers to probabilistic queries. One such a method, called ApproxCount, was proposed in [28] and is based on work from [29]. This method samples from the solution space of a CNF near–uniformly, by combining random walk with Metropolis moves. An advantage of the approach is that it can provide an approximation in cases where exact inference is not practical. Another advantage is that the time required can be adjusted by trading accuracy. Although this is an interesting area, we will only be covering exact probabilistic inference and exact model counting in this paper.

## 3 CNF Encodings

CNF encoding is the first dimension that has differentiated WMC approaches to probabilistic inference. It consists of defining the reduction to WMC, both semantically, by specifying variables, models, and weights, and syntactically, by specifying a CNF that captures the set of models. Encodings are important as the amount of local structure they capture can have a significant impact

[6] For more information, see http://bioinfo.cs.technion.ac.il/superlink.

on the performance of model counters, as we describe later.

### *3.1 Encoding* ENC1

The first encoding technique, which we will refer to as ENC1, was presented in [15]. Section 2.2 discussed most of what is involved in this encoding, including the semantic mapping and an example CNF. We now fill in the remaining pieces, which are how to produce the clauses of the CNF and how to capture determinism. First, we describe the clauses. For each network variable $X$ with domain $x_1, x_2, \ldots, x_k$, we have:

**Indicator Clauses:**

$$\lambda_{x_1} \vee \lambda_{x_2} \vee \ldots \vee \lambda_{x_k}$$
$$\neg\lambda_{x_i} \vee \neg\lambda_{x_j}, \text{ for } i < j$$

For example, variable $C$ from Figure 1 generates the following indicator clauses:

$$\lambda_{c_1} \vee \lambda_{c_2} \vee \lambda_{c_3}, \quad \neg\lambda_{c_1} \vee \neg\lambda_{c_2}, \quad \neg\lambda_{c_1} \vee \neg\lambda_{c_3}, \quad \neg\lambda_{c_2} \vee \neg\lambda_{c_3} \tag{2}$$

These clauses ensure that exactly one indicator variable for $C$ is set to true in each model. For each parameter $\mathsf{Pr}(x_i|u_1, u_2, \ldots, u_n)$, we produce the following clauses:

**ENC1 Parameter Clauses:**

$$\lambda_{u_1} \wedge \lambda_{u_2} \wedge \ldots \wedge \lambda_{u_n} \wedge \lambda_{x_i} \Leftrightarrow \theta_{x_i|u_1,u_2,\ldots,u_n}$$

For example, parameter $\theta_{c_1|a_1}$ in Figure 1 generates the following clauses:

$$\lambda_{a_1} \wedge \lambda_{c_1} \Rightarrow \theta_{c_1|a_1}, \quad \theta_{c_1|a_1} \Rightarrow \lambda_{a_1}, \quad \theta_{c_1|a_1} \Rightarrow \lambda_{c_1} \tag{3}$$

These clauses ensure that $\theta_{c_1|a_1}$ is set to true in a model iff $\lambda_{a_1}$ and $\lambda_{c_1}$ are set to true in that model.

ENC1 as discussed does not capture information about parameter values (local structure). We will be discussing encoding local structure in more detail in Section 5. However, from the very beginning, ENC1 effectively utilized determinism, which is quite easy to encode. Consider again Figure 1 and imagine that the parameter $\theta_{c_1|a_1}$ were 0. Given that this parameter is known to be 0, all models that set this parameter variable to true will have weight 0. Therefore, for this parameter variable, we can replace the clauses in Equation 3 by the single clause: $\neg\lambda_{a_1} \vee \neg\lambda_{c_1}$. This clause has the effect of eliminating all CNF models which contain the parameter $\theta_{c_1|a_1}$, which, like assigning them weight 0, also prevents them from influencing the result of the WMC computation. Furthermore, the parameter variable $\theta_{c_1|a_1}$ has been rendered superfluous and can be eliminated from the encoding. Although encoding determinism will not change the answer to the WMC computation, it can greatly improve the efficiency of WMC algorithms.

### *3.2 Encoding* ENC2

The second encoding technique, which we will refer to as ENC2, is based on the encoding presented in [16]. To describe ENC2, we assume that for each network variable, there is an ordering on its values. If value $x'$ comes earlier in the ordering than value $x$, we say $x' < x$. We describe ENC2 referring to the Bayesian network $N$ in Figure 1.

**Define logical variables:** ENC2 produces the same indicator variables as ENC1. For each network parameter $\mathsf{Pr}(x|\mathbf{u})$ such that $x$ is not $X$'s last value, ENC2 will produce a parameter variable $\rho_{x|\mathbf{u}}$. For example, for network $N$, ENC2 produces the following parameter variables.

| | |
|---|---|
| CPT 1: | $\rho_{a_1}$ |
| CPT 2: | $\rho_{b_1|a_1}, \rho_{b_1|a_2}$ |
| CPT 3: | $\rho_{c_1|a_1}, \rho_{c_2|a_1}, \rho_{c_1|a_2}, \rho_{c_2|a_2}$ |

**Define the KB semantics:** Each instantiation $\mathbf{w}$ of the network variables generates a set $\Omega$ of models. $\Omega$ is formed by requiring fixed values for certain CNF variables and leaving others as *don't cares.* In particular:

(1) Indicator Rule: $\mathbf{w}$ requires that indicator variable $\lambda_x$ be fixed to $\mathsf{true}$ if $x$ is compatible with $\mathbf{w}$ and fixed to $\mathsf{false}$ otherwise.
(2) Incompatible Parent Rule: $\mathbf{w}$ makes parameter variable $\rho_{x|\mathbf{u}}$ a don't care if $\mathbf{u}$ is incompatible with $\mathbf{w}$.
(3) Compatible Parent Rule: Let $x_w$ be the value that $\mathbf{w}$ assigns variable $X$; for each parameter variable $\rho_{x|\mathbf{u}}$ such that $\mathbf{u}$ is compatible with $\mathbf{w}$, (a) $\mathbf{w}$ requires that $\rho_{x|\mathbf{u}}$ be fixed to false if $x < x_w$, (b) $\mathbf{w}$ requires that $\rho_{x|\mathbf{u}}$ be fixed to $\mathsf{true}$ if $x = x_w$, and (c) $\mathbf{w}$ makes $\rho_{x|\mathbf{u}}$ a don't care if $x > x_w$.

For nework $N$, Table 3 lists network instantiations and corresponding sets of models. For network $N$, each network instantiation fixes either 10 or 11 CNF variables. Because there are a total of 14 CNF variables, each network instantiation therefore induces a set of $2^3$ or $2^4$ models.

Table 3
ENC2 models for the network in Figure 1.

| Network | Fixes these CNF | Fixes these CNF |
|---|---|---|
| Instantiation | vars to false | vars to true |
| $a_1b_1c_1$ | $\lambda_{a_2}, \lambda_{b_2}, \lambda_{c_2}, \lambda_{c_3}$ | $\lambda_{a_1}, \lambda_{b_1}, \lambda_{c_1}, \rho_{a_1}, \rho_{b_1\|a_1}, \rho_{c_1\|a_1}$ |
| $a_1b_1c_2$ | $\lambda_{a_2}, \lambda_{b_2}, \lambda_{c_1}, \lambda_{c_3}, \rho_{c_1\|a_1}$ | $\lambda_{a_1}, \lambda_{b_1}, \lambda_{c_2}, \rho_{a_1}, \rho_{b_1\|a_1}, \rho_{c_2\|a_1}$ |
| $a_1b_1c_3$ | $\lambda_{a_2}, \lambda_{b_2}, \lambda_{c_1}, \lambda_{c_2}, \rho_{c_1\|a_1}\rho_{c_2\|a_1}$ | $\lambda_{a_1}, \lambda_{b_1}, \lambda_{c_3}, \rho_{a_1}, \rho_{b_1\|a_1}$ |
| $a_1b_2c_1$ | $\lambda_{a_2}, \lambda_{b_1}, \lambda_{c_2}, \lambda_{c_3}, \rho_{b_1\|a_1}$ | $\lambda_{a_1}, \lambda_{b_2}, \lambda_{c_1}, \rho_{a_1}, \rho_{c_1\|a_1}$ |
| $a_1b_2c_2$ | $\lambda_{a_2}, \lambda_{b_1}, \lambda_{c_1}, \lambda_{c_3}, \rho_{b_1\|a_1}, \rho_{c_1\|a_1}$ | $\lambda_{a_1}, \lambda_{b_2}, \lambda_{c_2}, \rho_{a_1}, \rho_{c_2\|a_1}$ |
| $a_1b_2c_3$ | $\lambda_{a_2}, \lambda_{b_1}, \lambda_{c_1}, \lambda_{c_2}, \rho_{b_1\|a_1}, \rho_{c_1\|a_1}, \rho_{c_2\|a_1}$ | $\lambda_{a_1}, \lambda_{b_2}, \lambda_{c_3}, \rho_{a_1}$ |
| $a_2b_1c_1$ | $\lambda_{a_1}, \lambda_{b_2}, \lambda_{c_2}, \lambda_{c_3}, \rho_{a_1}$ | $\lambda_{a_2}, \lambda_{b_1}, \lambda_{c_1}, \rho_{b_1\|a_2}, \rho_{c_1\|a_2}$ |
| $a_2b_1c_2$ | $\lambda_{a_1}, \lambda_{b_2}, \lambda_{c_1}, \lambda_{c_3}, \rho_{a_1}, \rho_{c_1\|a_2}$ | $\lambda_{a_2}, \lambda_{b_1}, \lambda_{c_2}, \rho_{b_1\|a_2}, \rho_{c_2\|a_2}$ |
| $a_2b_1c_3$ | $\lambda_{a_1}, \lambda_{b_2}, \lambda_{c_1}, \lambda_{c_2}, \rho_{a_1}, \rho_{c_1\|a_2}, \rho_{c_2\|a_2}$ | $\lambda_{a_2}, \lambda_{b_1}, \lambda_{c_3}, \rho_{b_1\|a_2}$ |
| $a_2b_2c_1$ | $\lambda_{a_1}, \lambda_{b_1}, \lambda_{c_2}, \lambda_{c_3}, \rho_{a_1}, \rho_{b_1\|a_2}$ | $\lambda_{a_2}, \lambda_{b_2}, \lambda_{c_1}, \rho_{c_1\|a_2}$ |
| $a_2b_2c_2$ | $\lambda_{a_1}, \lambda_{b_1}, \lambda_{c_1}, \lambda_{c_3}, \rho_{a_1}, \rho_{b_1\|a_2}, \rho_{c_1\|a_2}$ | $\lambda_{a_2}, \lambda_{b_2}, \lambda_{c_2}, \rho_{c_2\|a_2}$ |
| $a_2b_2c_3$ | $\lambda_{a_1}, \lambda_{b_1}, \lambda_{c_1}, \lambda_{c_2}, \rho_{a_1}, \rho_{b_1\|a_2}, \rho_{c_1\|a_2}, \rho_{c_2\|a_2}$ | $\lambda_{a_2}, \lambda_{b_2}, \lambda_{c_3}$ |

**Define the CNF:** For each network variable, ENC2 defines the same indicator clauses as ENC1. ENC2 produces a single clause for each network parameter $\mathsf{Pr}(x_i|u_1, u_2, \ldots, u_n)$. Let the ordered domain of $X$ be $x_1, x_2, \ldots, x_k$. If $i \neq k$, then ENC2 produces the following parameter clause:

**ENC2 Normal Parameter Clause:**

$$\lambda_{u_1} \wedge \ldots \wedge \lambda_{u_n} \wedge \neg\rho_{x_1|\mathbf{u}} \wedge \ldots \wedge \neg\rho_{x_{i-1}|\mathbf{u}} \wedge \rho_{x_i|\mathbf{u}} \Rightarrow \lambda_{x_i}$$

Otherwise, ENC2 produces the following parameter clause:

**ENC2 Final Parameter Clause:**

$$\lambda_{u_1} \wedge \ldots \wedge \lambda_{u_n} \wedge \neg\rho_{x_1|\mathbf{u}} \wedge \ldots \wedge \neg\rho_{x_{k-1}|\mathbf{u}} \Rightarrow \lambda_{x_k}$$

For example, the parameter variable $\rho_{c_1|a_1}$ in network $N$ produces the clause:

$$\lambda_{a_1} \wedge \rho_{c_1|a_1} \Rightarrow \lambda_{c_1}$$

Following is the entire CNF produced by ENC2 for network $N$:

$$
\begin{array}{lll}
\text{Variable A:} & \lambda_{a_1} \vee \lambda_{a_2} & \neg\lambda_{a_1} \vee \neg\lambda_{a_2} \\
\text{Variable B:} & \lambda_{b_1} \vee \lambda_{b_2} & \neg\lambda_{b_1} \vee \neg\lambda_{b_2} \\
\text{Variable C:} & \lambda_{c_1} \vee \lambda_{c_2} \vee \lambda_{c_3} & \neg\lambda_{c_1} \vee \neg\lambda_{c_2} \\
 & & \neg\lambda_{c_1} \vee \neg\lambda_{c_3} \\
 & & \neg\lambda_{c_2} \vee \neg\lambda_{c_3} \\
\text{CPT 1:} & \rho_{a_1} \Rightarrow \lambda_{a_1} & \neg\rho_{a_1} \Rightarrow \lambda_{a_2} \\
\text{CPT 2:} & \lambda_{a_1} \wedge \rho_{b_1|a_1} \Rightarrow \lambda_{b_1} & \lambda_{a_2} \wedge \rho_{b_1|a_2} \Rightarrow \lambda_{b_1} \\
 & \lambda_{a_1} \wedge \neg\rho_{b_1|a_1} \Rightarrow \lambda_{b_2} & \lambda_{a_2} \wedge \neg\rho_{b_1|a_2} \Rightarrow \lambda_{b_2} \\
\text{CPT 3:} & \lambda_{a_1} \wedge \rho_{c_1|a_1} \Rightarrow \lambda_{c_1} & \lambda_{a_2} \wedge \rho_{c_1|a_2} \Rightarrow \lambda_{c_1} \\
 & \lambda_{a_1} \wedge \neg\rho_{c_1|a_1} \wedge \rho_{c_2|a_1} \Rightarrow \lambda_{c_2} & \lambda_{a_2} \wedge \neg\rho_{c_1|a_2} \wedge \rho_{c_2|a_2} \Rightarrow \lambda_{c_2} \\
 & \lambda_{a_1} \wedge \neg\rho_{c_1|a_1} \wedge \neg\rho_{c_2|a_1} \Rightarrow \lambda_{c_3} & \lambda_{a_2} \wedge \neg\rho_{c_1|a_2} \wedge \neg\rho_{c_2|a_2} \Rightarrow \lambda_{c_3}
\end{array}
$$

We also point out that determinism can be encoded in ENC2 as it was in ENC1. Specifically, we can replace a parameter clause corresponding to a 0 parameter with a shorter clause involving indicators only and remove parameter variables corresponding to 0 parameters.

**Assign weights:** ENC2 assigns weights to indicator variables in the same way as ENC1. For each positive literal of a parameter variable $\rho_{x_i|\mathbf{u}}$, we assign a weight that represents the probability that $X = x_i$ given $\mathbf{u}$ and given $X \neq x_j$ for any $x_j < x_i$. In particular, if the ordered domain of $X$ is $x_1, x_2, \ldots, x_k$, then $W(\rho_{x_1|\mathbf{u}}) = \mathsf{Pr}(x_1|\mathbf{u})$; $W(\rho_{x_2|\mathbf{u}}) = \mathsf{Pr}(x_2|\mathbf{u})/(1 - \mathsf{Pr}(x_1|\mathbf{u}))$; $W(\rho_{x_3|\mathbf{u}}) = \mathsf{Pr}(x_3|\mathbf{u})/(1 - \mathsf{Pr}(x_1|\mathbf{u}) - \mathsf{Pr}(x_2|\mathbf{u}))$, etc. Finally, $W(\neg\rho_{x_i|\mathbf{u}}) = 1 - W(\rho_{x_i|\mathbf{u}})$. For example, the weights of indicator variables for $N$ are all 1, and the weights of parameter variables are set as follows:

$$
\begin{array}{ll}
W(\rho_{a_1}) = 0.1 & W(\neg\rho_{a_1}) = 0.9 \\
W(\rho_{b_1|a_1}) = 0.1 & W(\neg\rho_{b_1|a_1}) = 0.9 \\
W(\rho_{b_1|a_2}) = 0.2 & W(\neg\rho_{b_1|a_2}) = 0.8 \\
W(\rho_{c_1|a_1}) = 0.1 & W(\neg\rho_{c_1|a_1}) = 0.9 \\
W(\rho_{c_2|a_1}) = 0.2/(1 - 0.1) = 0.222 & W(\neg\rho_{c_2|a_1}) = 0.778 \\
W(\rho_{c_1|a_2}) = 0.01 & W(\neg\rho_{c_1|a_2}) = 0.99 \\
W(\rho_{c_2|a_2}) = .09/(1 - 0.01) = 0.091 & W(\neg\rho_{c_2|a_2}) = 0.909
\end{array}
$$

**Incorporating evidence and computing WMC:** Given the above knowledge base and assignment of weights, we can now compute probability of evidence by incorporating evidence in the same manner as for ENC1. Both setting weights of incompatible indicators to zero and conjoining with a term encoding the evidence work as before.

### *3.3 On the Difference between* ENC1 *and* ENC2

ENC2 is like ENC1 in its handling of network variables, which correspond to indicator variables in the CNF. In particular, ENC2 defines the same set of indicator variables, produces the same set of indicator clauses, defines the same weights on indicator literals, and can utilize indicators to encode determinism and incorporate evidence in the same way. However, ENC2 differs significantly in its dealing with network parameters. One of the most obvious differences is that ENC2 will produce fewer parameter variables than ENC1. The major reason that ENC2 is able to produce fewer parameter variables is that, unlike ENC1, ENC2 encodes information in the weights of negative parameter literals. By packing more information into the two weights of each parameter variable, ENC2 is able to use fewer such variables. ENC2 also produceds fewer parameter clauses than ENC1. However, ENC2 produces larger clauses than ENC1 when variables have more than two values. Moreover, the encoding of information into negative literals causes ENC2 to be arguably less intuitive. Both encodings generate a CNF whose size is polynomial in the size of the Bayesian network.

The most striking difference between ENC1 and ENC2 is in the set of models included in the knowledge base. For each instantiation $\mathbf{w}$ of network variables, ENC1 defined a single model with weight equal to $\mathbf{w}$'s probability from the joint distribution. In contrast, ENC2 defines many models $\Omega$ for $\mathbf{w}$, in such a way that the sum of their weights will equal $\mathbf{w}$'s probability from the joint. When $\mathbf{w}$ is inconsistent with evidence, the weights of all models $\Omega$ must therefore be excluded from the sum of the WMC computation. Note that the sets of models corresponding to two different network instantiations are guaranteed to be disjoint, since each will fix a different instantiation of indicator variables.

We close this section with an empirical comparison of ENC1 and ENC2, which has not previously been performed in the literature. Table 4 shows two groups of networks which were used in [16], [22], and [1] to evaluate WMC techniques. The first group consists of networks that have only Boolean variables and large amounts of determinism. Many of the networks in the second group contain non–Boolean variables, large CPTs, and lesser amounts of determinism. For each of these networks, we encoded the network into CNF using both encodings, including encoding for determinism. We then compiled the resulting CNFs using a 1.6GHz Pentium M with 2GB of memory and the C2D compiler [26], upon which ACE is built. However, our experiments deviate from other published results in the following ways. First, C2D uses a data structure called a dtree to drive the compilation. To isolate differences in encoding only, we used a method of producing a dtree [1] that has proven more successful than other methods on many networks, and we ensure that the same dtree is used for both encodings. Second, we have added a few small enhancements to both encodings to make them more compact.[7] Let the size of a clause be the number of literals in the clause and the size of a CNF to be the sum of the sizes of its clauses. Table 4 shows results by listing, for each encoding, the CNF size, the compile time, and the compilation size. In general, ENC2 produces smaller CNFs, but this becomes less true when variables are non–binary, as in the case of mildew, or there is a massive amount of determinism, as in the case of the bm, mm, and st networks. On the first group of networks, ENC2 produces slightly faster compilations, and on the second, ENC2 is faster by a factor between two and five. However, the sizes of the compilations are roughly equivalent. Also note that both encodings failed on the same networks, as indicated by dashes. The main point is that ENC2 is somewhat more efficient than ENC1 in producing the compilation, but the compilation produced is about the same. Another dimension for comparing the two encodings relates to their flexibility in encoding network local structure, an issue that we discuss in a later section.

## 4 Search vs Knowledge Compilation

In the previous section, we discussed two different ways one can convert the probabilistic inference problem to WMC on a CNF. In this section, we discuss the second dimension on which WMC approaches differ. Specifically, we review how model counting can be performed using search or knowledge compilation.

### *4.1 WMC by Search*

Consider the problem of counting the models in the CNF shown at the top of Figure 2. It was observed in [30] that if one can decompose the CNF into components that do not share variables, then one can count the components independently. Because all of the clauses in this CNF contain variable $A$, we

[7] For example, let $X$ be a network variable. If $X$ has two values, then both encodings can use a single indicator variable for $X$ instead of two. If $X$ is a root, both encodings can omit the production of CNF parameter variables for $X$'s CPT by assigning appropriate weights to $X$'s indicator variables.

Table 4
Comparison of ENC1 and ENC2; compilation performed using the Ace compiler. Experiments ran on a 1.6GHz Pentium M with 2GB of memory. Dash indicates failure.

| | ENC1 | ENC2 | ENC1 | ENC2 | ENC1 | ENC2 |
|---|---:|---:|---:|---:|---:|---:|
| | CNF | CNF | Comp. | Comp. | Comp. | Comp. |
| Network | Size | Size | Time (s) | Time (s) | Edges | Edges |
| Grid-50-16-1 | 10,568 | 5,112 | 147.09 | 127.97 | 15,075,826 | 14,869,887 |
| Grid-50-16-2 | 11,252 | 5,316 | 223.46 | 207.22 | 20,855,797 | 21,872,882 |
| Grid-50-16-3 | 10,048 | 4,960 | 63.80 | 53.20 | 6,121,365 | 6,052,060 |
| Grid-50-16-4 | 10,748 | 5,164 | 192.72 | 162.44 | 17,313,216 | 16,703,733 |
| Grid-50-16-5 | 11,000 | 5,240 | 186.82 | 150.67 | 13,599,834 | 13,867,070 |
| Grid-50-16-6 | 10,768 | 5,168 | 74.72 | 57.60 | 5,394,504 | 5,294,194 |
| Grid-50-16-7 | 10,412 | 5,068 | 135.54 | 115.70 | 12,110,197 | 12,366,429 |
| Grid-50-16-8 | 10,592 | 5,120 | 64.25 | 52.50 | 5,754,715 | 5,817,835 |
| Grid-50-16-9 | 12,088 | 5,560 | 79.13 | 67.60 | 6,207,120 | 6,199,093 |
| Grid-50-16-10 | 11,416 | 5,368 | 118.66 | 98.18 | 10,648,704 | 10,670,540 |
| bm-5-3 | 10,132 | 10,012 | 0.21 | 0.20 | 19,082 | 18,982 |
| mm-3-8-3 | 12,096 | 11,880 | 1.19 | 1.26 | 275,754 | 284,901 |
| st-3-2 | 5,717 | 4,323 | 0.28 | 0.23 | 36,371 | 28,342 |
| alarm | 8,310 | 4,111 | 0.21 | 0.14 | 5,002 | 5,621 |
| diabetes | 1,823,267 | 1,498,051 | – | – | – | – |
| hailfinder | 41,009 | 20,052 | 1.37 | 0.53 | 23,118 | 25,676 |
| mildew | 1,723,852 | 1,619,395 | 5,014.87 | 2,713.99 | 3,555,313 | 2,752,047 |
| munin2 | 376,168 | 294,528 | 2,671.08 | 1,518.62 | 6,248,456 | 6,144,328 |
| munin3 | 385,809 | 303,670 | 2,279.14 | 671.36 | 5,337,629 | 5,344,186 |
| munin4 | 440,449 | 335,422 | – | – | – | – |
| pathfinder | 590,631 | 325,890 | 77.47 | 14.94 | 137,350 | 159,716 |
| pigs | 40,987 | 24,689 | 90.52 | 47.49 | 2,207,573 | 2,207,534 |
| tcc4f | 71,840 | 26,056 | 3.06 | 0.69 | 39,708 | 38,938 |
| water | 143,338 | 75,778 | 39.21 | 8.19 | 291,354 | 265,141 |

cannot decompose the CNF immediately. To force decomposition, we split on some variable; for example, $A$. Following the split, we perform unit resolution, and perhaps some other simplifications, and generate the two CNFs at the middle of Figure 2. At this point, we solve each of the two subproblems recursively. We see that in each of the two subproblems, we can now decompose

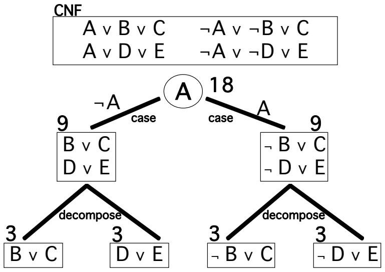


Fig. 2. An example of a search algorithm that performs model counting.

into two sets of clauses that share no variables. The four resulting sets are shown at the bottom of the figure. Computing the model count for each CNF at the bottom of the figure constitutes a base case in the recursion, and each is assigned a count of 3, as indicated in the figure. From the counts at the bottom, we can compute counts for the CNFs in the middle, both 9 in this case. And from these counts, there is a formula to compute the count for the CNF at the top of the figure, 18 in this case, which is the answer to the original problem. Performing weighted model counting is a simple generalization of the described procedure which accounts for weights in the base case of the recursion.

The steps we have described constitute solving WMC by search, as initially proposed by [30]. Although we have illustrated the search in a breadth first–fashion, it is normally performed depth–first [30]. In addition, advanced techniques such as clause learning, component caching, and non–chronological backtracking, are used to improve efficiency, but we do not detail them here; see [18,31,26,32]. Two major advantages of search are that it typically has lower space requirements than compilation and that it can exploit query–specific structure.

Such a search was used in [16] to demonstrate the power of WMC in performing probabilistic inference. Here, a state–of–the–art model counter called Cachet [18] was applied to CNFs generated according to ENC2. Table 5, adapted from [16], demonstrates some of these results, comparing Cachet times to times to perform probabilistic inference using two algorithms that exploit only topological structure, jointree [4,5] and recursive conditioning [8]. An important aspect of the networks used is the large amount of determinism. In this table, each number represents the median time to perform inference for ten separate experiments; a number $n$ in parenthesis means that only $n$ of the ten queries were successfully answered; and an $X$ means that no problems were solved successfully. The main observation is that Cachet times were superior when compared to jointree and recursive conditioning. In particular, jointree and recursive conditioning could not answer most of the listed queries, and took more than an order of magnitude longer than Cachet on many queries they could answer. This performance of algorithms that exploit only topological structure, however, is not too surprising as these methods take no advantage

Table 5
Results adapted from [16] that demonstrate the effectiveness of WMC using search and ENC2 on various highly deterministic networks. Each number represents the median time to perform inference for ten separate experiments; a number $n$ in parenthesis means that only $n$ of the ten queries were successfully answered; and an $X$ means that no problems were solved successfully.

| | Jointree | RC | Cachet |
|---|---|---|---|
| Network | Time (s) | Time (s) | Time (s) |
| DQMR–60+60–0.05 | 52 (5) | 5.7 (2) | 1.7 |
| DQMR–60+60–0.1 | 46 (3) | 33 (3) | 3.9 |
| DQMR–60+60–0.2 | 45 (5) | 60 (4) | 54 |
| DQMR–70+70–0.05 | X | X | 12 |
| DQMR–70+70–0.1 | X | X | 60 |
| DQMR–70+70–0.2 | X | X | 136 |
| Grid–0.75–10 | 0.02 | 0.87 | 0.3 |
| Grid–0.75–14 | 20 | 15 | 4.7 |
| Grid–0.75–18 | X | 1751 | 81 |
| Grid–0.75–22 | X | X | 1300 |
| 5–step | 56 | 36 | 0.03 |
| tire–2 | X | X | 0.09 |
| tire–4 | X | X | 1.1 |
| log–2 | X | X | 7.9 |
| log–4 | X | X | 65 |

of the massive determinism available in these networks. Moreover, the findings are consistent with those reported in [15], which presented similar experimental results on Bayesian networks with massive determinism. It was also shown in [16] that in addition to probability of evidence, marginals on variables can be computed by recording certain information during the search. However, in this case, much of the space advantage over compilation goes away.

### *4.2 WMC by Knowledge Compilation*

Knowledge compilation refers to a process whereby a logical theory is compiled from one logical form into a target logical form, which permits certain queries to be answered in polytime (in the size of the target form). A knowledge compilation map is given in [33], listing a number of target languages and classifying them according to their relative succinctness and the class of operations they support in polytime. A strong connection between search and knowledge compilation was presented in [34], showing that the trace of a search algorithm can be interpreted as a compiled sentence of some target language. Therefore, a search algorithm can be converted into a knowledge compiler by extending it so it saves its trace. For example, it was shown in [34] that the search algorithms for model counting, discussed in the previous section, generate traces that are members of the d–DNNF language which we will discuss later in this section; see Figure 3. Hence, the search and knowledge compilation approaches to WMC are indeed very closely related.

In fact, the knowledge compiler used in the experiments in this paper, called c2d, is based on the search algorithm described in the previous section. The main difference between c2d and Cachet is that c2d records a trace of its operations in the manner described in [34]. Other differences include different methods for implementing decompositions, variable splitting, and caching. It is interesting to note that Cachet could likely be easily converted into a compiler by modifying it to keep a trace of its operations.

The main advantage of compilation is that a significant amount of the work required for inference is performed once offline, which can then be amortized over many online queries. Repeated online inference is often much more efficient using knowledge compilation than search. One disadvantage of knowledge compilation is that the compiled theory could end up being too large for available memory. In these cases, a search, which requires less overhead and can trade off time for space, might be able to succeed where compilation fails. However, much of this space disadvantage disappears when computing multiple counts simultaneously [16]. Another disadvantage is that query–specific information can often be utilized to simplify a search, but since knowledge compilation is usually performed with a goal of being able to answer any query, it typically cannot take advantage of query–specific information (however, see Section 5.3).

The first application of knowledge compilation to probabilistic inference (and weighted model counting) was given in [15,20]. In this work, the CNF of a Bayesian network, encoded according to ENC1, was compiled into a target language called d–DNNF, which is known to support weighted model counting in polytime [33] in the size of the compiled form (this does not escape the complexity of WMC since the compilation and size of resulting structure are exponential in treewidth in the worst case). Among the results presented are compilation times and sizes for various networks generated from ISCAS 85 and ISCAS 89 circuits, which were later improved upon in [26], as shown in Table 6. For each network, the table first lists the maximum cluster size for a jointree generated for the network. This metric is important, because methods that exploit only topological structure for probabilistic inference run

Table 6
Results from [26] that demonstrate the effectiveness of WMC using compilation and ENC1 on circuits.

|  | Max | Compile | Compilation |
|---|---:|---:|---:|
| Network | Cluster | Time (s) | Size |
| c432 | 28 | 0 | 13,767 |
| c499 | 23 | 6 | 2,214,814 |
| c880 | 24 | 80 | 20,676,927 |
| c1355 | 23 | 15 | 2,748,340 |
| c1908 | 45 | 187 | 18,376,664 |
| s1423 | 25 | 3 | 467,935 |
| s3330 | 43 | 13 | 2,496,907 |

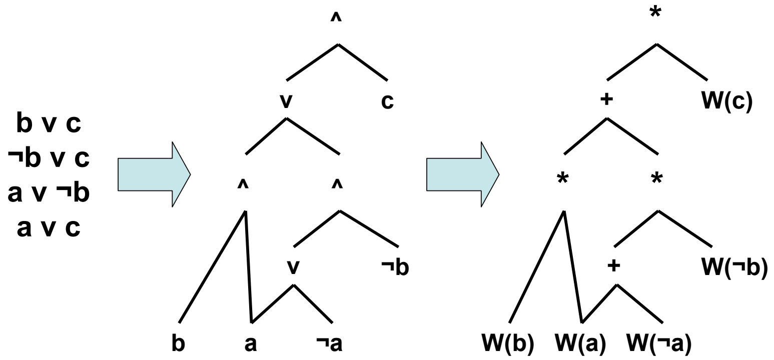


Fig. 3. Compiling CNF into a smooth d–DNNF and then extracting an AC by: replacing conjunctions with multiplications, disjunctions with additions, and literals with their weights.

in time that is exponential in maximum cluster size. From these sizes, it is clear that many of these networks are beyond the reach of algorithms that exploit only topological structure. However, compilation was successful in a reasonable amount of time in each case. Because d–DNNF sizes (measured in number of edges) are also quite reasonable, online inference, which is linear in d–DNNF size, will be very efficient.

More evidence for the effectiveness of compilation was presented in [22], which used Ace and reported on the successful compilation of networks generated from relational Bayesian networks [35] and encoded according to ENC1. A sampling of results for these highly deterministic networks are shown in Table 7. Again, the most striking observation is the success of compilation in spite of large maximum cluster size. We will discuss the last two columns of Table 7 below.

A d–DNNF is a rooted DAG with internal nodes labeled with disjunctions

Table 7
Results from [22] that demonstrate the effectiveness of WMC using compilation and ENC1 on ground instances of relational Bayesian networks. Dash indicates failure. Online time is averaged over thirty–one evidence sets, where for each evidence set, we compute probability of evidence and a posterior marginal for every network variable.

| | | | | Avg. Online | Avg. Jointree |
|---|---|---|---|---|---|
| | Max | Compile | Compile | Inference | Inference |
| Network | Cluster | Time (**min**) | Size | Time (s) | Time (s) |
| mastermind | | | | | |
| 04–08–03 | 26 | 1 | 541,356 | 0.0516 | 57.48 |
| 06–08–03 | 37 | 1 | 1,523,888 | 0.1518 | – |
| 10–08–03 | 54 | 3 | 4,315,566 | 0.6835 | – |
| 04–08–04 | 39 | 5 | 19,457,308 | 1.7341 | – |
| 03–08–05 | 40 | 10 | 55,417,639 | 4.3253 | – |
| students | | | | | |
| 03–12 | 59 | 1 | 113,876 | 0.0175 | – |
| 04–16 | 101 | 3 | 815,461 | 0.0930 | – |
| 05–20 | 148 | 7 | 5,236,257 | 1.8439 | – |
| 06–24 | 233 | 33 | 36,450,231 | 12.9663 | – |
| blockmap | | | | | |
| 05–03 | 23 | 1 | 20,636 | 0.0068 | 27.39 |
| 10–03 | 52 | 2 | 974,817 | 0.0582 | – |
| 15–03 | 68 | 6 | 7,643,307 | 0.3799 | – |
| 20–03 | 92 | 30 | 40,172,434 | 2.4529 | – |
| 22–03 | 104 | 61 | 76,649,302 | 4.6651 | – |

and conjunctions, and its leaf nodes labeled with propositional literals (or the constants true and false). A d–DNNF satisfies the decomposition property (conjuncts cannot share variables) and the determinism property (disjunctions must be logically disjoint). Another property is needed for model counting, called smoothness (disjuncts mention the same set of variables). Figure 3 depicts a CNF and a corresponding d–DNNF that is smooth. By compiling the CNF to a d–DNNF, one can perform WMC by simply traversing the d–DNNF and performing simple multiplication and addition operations at its nodes.

The approach in [15], however, further maps the d–DNNF into an arithmetic circuit (AC) that explicates these arithmetic operations, leading to a WMC circuit as given in Figure 3. In particular, the AC inputs can be used to capture the weights of literals and its output will provide the weighted model count for the given literal weights. For example, given the weights $W(a) = .2, W(\neg a) = 1, W(b) = .5, W(\neg b) = .5, W(c) = .9, W(\neg c) = 1$ for the CNF literals in Figure 3, one can simply evaluate the AC given in the same figure under these weights, leading to a weighted model count of 0.63. Mapping the CNF into an AC has a major advantage: by computing the partial derivatives of the AC, one can compute marginal model counts, in time linear only in the AC size [20,36]. This allows the AC to simultaneously compute the probability of evidence and posterior marginals over all network variables, in time linear in the AC size [36]. A main point is that this process may then be repeated for as many evidence sets as desired, without need to recompile. We note here that many of the results on probabilistic inference by WMC, using the compilation approach, are reported in terms of the compilation size, which refers to the number of edges in the AC for given CNF encoding. We also note that the AC can be given an alternative algebraic semantics: it is a factorization of an exponentially–sized polynomial that captures the probability distribution induced by the given Bayesian network [36].

We close this section with two additional comparisons, to put the compile/search times in some more perspective. Examine once more Table 7, which shows the key discrepancy between offline compile time (shown in minutes) and online inference time (shown in seconds). The compilation can be costly as shown in the table, but that cost can be amortized over many online queries, which become very efficient, and which may correspond to differing evidence. Each time reported for online inference is the average over many experiments. In each experiment, a different evidence set was generated and probability of evidence and a posterior marginal for each network variable were computed. The jointree algorithm cannot answer most queries (as indicated by the dashes), because of cluster size, and WMC is many orders of magnitude more efficient in cases where jointree is successful.

We now provide a second comparison not previously addressed in the literature, between WMC by search (performed by the Cachet model counter version 2.0 [16]) and WMC by compilation (performed by Ace v2.0 [3]). For each network in Table 8, we encoded the network using Cachet and then applied both Ace and Cachet to the generated CNF. The resulting search and compilation times can be seen in the second and third columns of Table 8, where a dash represents a timeout (2000s) or other failure, and where each row marked with a star (*) indicates an average over ten similar networks. The networks with which we experimented were drawn from the WMC literature. The networks in the first group, presented in [16] and [22], are highly deterministic and have small CPTs. These are the types of networks on which WMC performs particularly well. In general, Ace compilation times appear roughly equal to Cachet search times on the grid and OR networks and superior on

Table 8

Times to search using Cachet and to compile and perform online inference using Ace. A star (*) indicates an average over ten networks. Online time is averaged over ten evidence sets, where for each evidence set, we compute probability of evidence and a posterior marginal for every network variable. Experiments were performed on a 2.8GHz processor with 4GB of memory.

| | Cachet Encoding | | Ace Encoding | |
|---|---|---|---|---|
| | | | | Avg. |
| | Ace Comp. | Cachet Srch. | Ace Comp. | Ace Online |
| Network | Time (s) | Time (s) | Time (s) | Time (s) |
| grid-50-14-* | 22.9 | 426.6 | 7.0 | 0.076 |
| grid-50-16-* | 231.9 | 761.2 | 94.0 | 0.959 |
| grid-90-30-* | 92.8 | 111.1 | 37.0 | 0.064 |
| grid-90-34-* | 449.3 | 467.9 | 90.3 | 0.114 |
| or-60-10-*-10 | 2.8 | 2.8 | 7.7 | 0.001 |
| or-60-20-*-10 | 30.7 | 30.3 | 109.1 | 0.004 |
| blockmap-15-02 | 47.5 | 187.5 | 110.9 | 0.066 |
| blockmap-20-02 | 303.0 | 1332.8 | 709.2 | 0.245 |
| blockmap-22-02 | 473.1 | - | 1701.7 | 0.403 |
| mastermind-03-08-03 | 3.1 | - | 2.9 | 0.021 |
| mastermind-03-08-04 | 50.2 | - | 15.8 | 0.190 |
| mastermind-03-08-05 | 716.5 | - | 345.7 | 4.104 |
| mastermind-04-08-03 | 7.0 | - | 7.4 | 0.053 |
| mastermind-04-08-04 | 346.4 | - | 115.8 | 1.181 |
| alarm | 0.2 | 29.9 | 0.1 | 0.003 |
| hailfinder | 2.3 | 5.1 | 0.5 | 0.013 |
| munin2 | - | - | 1930.8 | 0.625 |
| munin3 | - | - | 784.1 | 0.365 |
| munin4 | - | - | 541.5 | 0.528 |
| pathfinder | - | 59.9 | 4.7 | 0.009 |
| pigs | 89.6 | - | 49.9 | 0.163 |
| tcc | 0.5 | - | 0.9 | 0.012 |
| water | 5.4 | 75.3 | 0.9 | 0.010 |

Table 9
Similar to Table 8 but using a faster computer applied to networks on which Ace sometimes had more difficulty than Cachet. Note that each row corresponds to a single network, unlike Table 8, in which some rows correspond to an average over ten networks. Online time is averaged over ten evidence sets, where for each evidence set, we compute probability of evidence and a posterior marginal for every network variable.

| | Cachet Encoding | | Ace Encoding | |
|---|---|---|---|---|
| | | | | Avg. |
| | Ace Comp. | Cachet Srch. | Ace Comp. | Ace Online |
| Network | Time (s) | Time (s) | Time (s) | Time (s) |
| grid-75-24-02 | 2.4 | 0.5 | 2.0 | 0.004 |
| grid-75-24-04 | 333.9 | 882.9 | 39.0 | 0.127 |
| grid-75-24-06 | 555.0 | 1,467.8 | 134.3 | 0.623 |
| grid-75-24-08 | 69.3 | 258.9 | 33.5 | 0.129 |
| grid-75-24-10 | - | - | 97.9 | 0.426 |
| grid-90-42-02 | - | 126.4 | 22.9 | 0.022 |
| grid-90-42-04 | - | - | - | - |
| grid-90-42-06 | - | 1,918.2 | 865.2 | 1.490 |
| grid-90-42-08 | - | - | - | - |
| grid-90-42-10 | - | - | 1,455.9 | 2.718 |
| or-070-20-02-10 | 6.1 | 12.8 | 19.5 | 0.154 |
| or-070-20-04-10 | 129.1 | 181.6 | - | - |
| or-070-20-06-10 | 20.8 | 37.6 | 208.1 | 1.252 |
| or-070-20-08-10 | 288.5 | 172.3 | - | - |
| or-070-20-10-10 | 27.8 | 33.0 | 120.2 | 0.820 |
| or-100-20-02-20 | 108.8 | 106.5 | - | - |
| or-100-20-04-20 | 26.8 | 126.8 | 70.1 | 0.523 |
| or-100-20-06-20 | 243.5 | 197.7 | - | - |
| or-100-20-08-20 | 127.9 | 94.9 | 1,110.9 | 4.761 |
| or-100-20-10-20 | 55.3 | 225.4 | 365.6 | 2.350 |

the blockmap and mastermind networks. The networks in the second group, from [1], contain multi–valued variables, lesser amounts of determinism, and in some cases, large CPTs. On these networks, both Cachet and Ace struggle.

It is important to realize that the goal in these initial columns was to isolate differences as much as possible to those resulting from search vs. compilation. Consequently, we used the same CNF encoding for both tools, even though each tool most naturally prefers its own encoding.[8] Moreover, we disabled in ACE many features for exploiting other types of local structure beyond determinism, which often improve efficiency (e.g., encoding equal parameters, structured resolution, eclauses, etc.), because Cachet does not implement them. Finally, we used precisely those options recommended by the authors of the programs in all experiments.[9] Even with all of these measures, it turns out to be difficult to compare search to compilation using current tools, because Cachet and ACE differ in many additional ways that could not be excluded from the experiments. For example, each tool implements different methods for decomposition, variable splitting, and caching.

We also ran ACE using its own native encoding and turning on all of its additional features for exploiting local structure, which we will describe more in Section 5. Compilation times in this case are shown in the fourth column of Table 8. Finally, once compilation is complete, we can use the compiled form to answer many queries. The fifth column shows the time to compute, for given evidence, the probability of evidence and a posterior marginal on each network variable, averaged over ten different evidence sets. Once compilation is complete, this online inference may be repeated for as many evidence sets as desired, and the variance in the time required is very small. For networks in the first group, using ACE's own encoding sometimes reduces compilation times significantly, as in the grid networks, but it also sometimes increases compilation time, as in the OR and blockmap networks. One reason for these losses is that ACE applies many techniques to exploit local structure other than determinism. When such other structure does not exist, these techniques serve to increase overhead. For the networks in the second group, utilizing other forms of local structure than determinism benefits compilation time significantly. In both groups, the advantages of compilation become apparent, when one examines the online times, which are orders of magnitude faster than the search times.

For each of the grid, OR, blockmap, and mastermind networks, there were a large number of networks from which to choose, providing a range of difficulties. In general, the results presented in Table 8 are representative across all difficulties. However, some of the grid and OR networks were more problematic for ACE. Results for some of these more problematic networks are shown

[8] We used Cachet's encoding, because Cachet does not work currently with ACE's default encoding.

[9] Thanks to Tian Sang for suggesting Cachet's settings and for other assistance.

in Table 9,[10] where each row represents a single experiment, in contrast to Table 8, in which some rows correspond to averages over similar networks.

In general, one would expect search to be somewhat more efficient than compilation, since it does not need to keep a trace of its actions as compilation does. In these experiments, this was not always the case, because we were not able to do a pure comparison of search vs. compilation. Nevertheless, these numbers should give some insight into the current stat–of–the–art tools for performing inference by WMC.

## 5 Local Structure and Evidence

The WMC approaches discussed thus far perform well on networks where variables are binary, CPTs are small, and where there is massive determinism. However, on other types of networks, the generated CNF encodings can be quite large, challenging state of the art WMC systems. In this section, we review work that enhances ENC1 and scales the WMC approach to perform well on a much wider variety of networks and problems. In previous sections, we discussed the encoding of determinism, which can be very effective. In this section, we discuss the encoding of other forms of structure, including equal parameters, decomposability, and evidence.

### *5.1 Equal Parameters*

Table 10, repeated from [1], lists a set of benchmark networks, some having variables with large cardinalities, others having very large CPTs, and where the amount of determinism is not necessarily excessive. Table 11, also repeated from [1], provides statistics on the CNFs generated for some of these networks, according to ENC1, which includes encoding of determinism as discussed in Section 3.1. These CNFs are quite large, but the striking property is the large percentage (up to 99% in some cases) of Boolean variables that represent parameters as opposed to those representing indicators. Some of these CNFs proved initially challenging to compile, some taking too long and others running out of memory.

Consider the CPT depicted in Table 12, which has 12 parameters, yet only 5 of these are distinct. One would want to exploit parameter equality, at least to reduce the number of Boolean variables one must generate. Table 10

[10] We used a faster computer for these experiments, because both Cachet and Ace often timed out using the slower computer.

Table 10
Information from [1] on the networks utilized in experiments.

| | | | | | | Max | Ave | | |
|---|---|---|---|---|---|---|---|---|---|
| | Max | | | Ave | Total | CPT | CPT | | |
| Network | Clust | Vars | Card | Card | Parms | Parms | Parms | %Det | %DP |
| alarm | 7.2 | 37 | 2-4 | 2.8 | 752 | 108 | 20 | 0.9 | 24.6 |
| bm | 20.0 | 1005 | 2-2 | 2.0 | 6972 | 8 | 7 | 99.6 | 100.0 |
| diabetes | 17.2 | 413 | 3-21 | 11.3 | 461069 | 7056 | 1116 | 78.2 | 17.6 |
| hailfinder | 11.7 | 56 | 2-11 | 4.0 | 3741 | 1188 | 67 | 15.7 | 26.9 |
| mildew | 21.4 | 35 | 3-100 | 17.6 | 547158 | 280000 | 15633 | 93.2 | 25.1 |
| mm | 23.0 | 1220 | 2-2 | 2.0 | 8326 | 8 | 7 | 98.7 | 75.0 |
| munin1 | 26.8 | 189 | 1-21 | 5.3 | 19466 | 600 | 103 | 66.5 | 61.2 |
| munin2 | 18.6 | 1003 | 2-21 | 5.4 | 83920 | 600 | 84 | 63.3 | 69.5 |
| munin3 | 17.8 | 1044 | 1-21 | 5.4 | 85855 | 600 | 82 | 63.1 | 71.3 |
| munin4 | 21.4 | 1041 | 1-21 | 5.4 | 98183 | 600 | 94 | 64.5 | 65.3 |
| pathfinder | 15.0 | 109 | 2-63 | 4.1 | 97851 | 8064 | 898 | 56.1 | 5.1 |
| pigs | 17.4 | 441 | 3-3 | 3.0 | 8427 | 27 | 19 | 56.2 | 23.9 |
| students | 22.0 | 376 | 2-2 | 2.0 | 2616 | 8 | 7 | 90.7 | 79.3 |
| tcc4f | 10.0 | 105 | 2-2 | 2.0 | 3236 | 512 | 31 | 0.4 | 35.6 |
| water | 19.9 | 32 | 3-4 | 3.6 | 13484 | 3072 | 421 | 54.0 | 57.0 |

Table 11
Information from [1] on CNFs generated by ENC1 (determinism encoded).

| Network | CNF Vars | Parm Vars | Clauses | Literals |
|---|---|---|---|---|
| pathfinder | 55,229 | 54,781 | 300,576 | 821,814 |
| water | 6,630 | 6,514 | 49,367 | 152,686 |
| mildew | 38,540 | 37,924 | 683,552 | 1,958,952 |
| munin1 | 9,551 | 8,556 | 49,363 | 129,358 |
| munin4 | 48,864 | 43,216 | 247,582 | 641,839 |
| diabetes | 113,527 | 108,845 | 814,412 | 2,196,008 |

shows the extent to which parameter equality can help. In particular, the table reports as %Det the percentage of parameters that are extreme (0 or 1) and as %DP the percentage of non–extreme parameters that would remain if, for each CPT, one collapsed equal parameters into a single parameter. The dramatic example is *pathfinder*, where roughly have of its parameters are extreme, and therefore benefit by encoding determinism, and where only 5% of the non–

Table 12
A CPT showing local structure in the form of determinism and CSI.

| row | $A$ | $B$ | $C$ | $\mathsf{Pr}(c \mid a, b)$ |
|---|---|---|---|---|
| 1 | $a_1$ | $b_1$ | $c_1$ | $\theta_{c_1\|a_1b_1} = 0$ |
| 2 | $a_1$ | $b_1$ | $c_2$ | $\theta_{c_2\|a_1b_1} = 0.5$ |
| 3 | $a_1$ | $b_1$ | $c_3$ | $\theta_{c_3\|a_1b_2} = 0.5$ |
| 4 | $a_1$ | $b_2$ | $c_1$ | $\theta_{c_1\|a_1b_2} = 0.2$ |
| 5 | $a_1$ | $b_2$ | $c_2$ | $\theta_{c_2\|a_1b_1} = 0.3$ |
| 6 | $a_1$ | $b_2$ | $c_3$ | $\theta_{c_3\|a_1b_1} = 0.5$ |
| 7 | $a_2$ | $b_1$ | $c_1$ | $\theta_{c_1\|a_2b_2} = 0$ |
| 8 | $a_2$ | $b_1$ | $c_2$ | $\theta_{c_2\|a_2b_2} = 0$ |
| 9 | $a_2$ | $b_1$ | $c_3$ | $\theta_{c_3\|a_2b_1} = 1$ |
| 10 | $a_2$ | $b_2$ | $c_1$ | $\theta_{c_1\|a_2b_1} = 0.2$ |
| 11 | $a_2$ | $b_2$ | $c_2$ | $\theta_{c_2\|a_2b_2} = 0.3$ |
| 12 | $a_2$ | $b_2$ | $c_3$ | $\theta_{c_3\|a_2b_2} = 0.5$ |

extreme parameters would remain after collapsing.

A key observation is that for ENC1, no two parameter variables generated for the same CPT can both be set to $\mathsf{true}$ in the same model, since they correspond to inconsistent network instantiations. This observation suggests that we can use the same Boolean variable to represent multiple parameters, provided that such parameters have equal values and appear in the same CPT. However, the idea will not work when applied directly to ENC1. Consider again the CPT in Table 12. If we use the same CNF variable $\theta$ to represent parameters $\mathsf{Pr}(c_2|a_1, b_1)$ and $\mathsf{Pr}(c_3|a_1, b_2)$, which are both equal to 0.5, we would get the following parameter clauses in the CNF:

$$\lambda_{a_1} \wedge \lambda_{b_1} \wedge \lambda_{c_2} \Leftrightarrow \theta \qquad \lambda_{a_1} \wedge \lambda_{b_2} \wedge \lambda_{c_3} \Leftrightarrow \theta \tag{4}$$

$\theta$ then implies incompatible configurations of the variables $A$, $B$, and $C$, as enforced by the indicator clauses for those variables. The undesired result is the removal from the theory of all models that set variable $\theta$ to $\mathsf{true}$. The solution adopted in [1] is to convert the equivalence in the parameter clauses of ENC1 to an implication. The clauses in Equation 4 are therefore changed to the following:

$$\lambda_{a_1} \wedge \lambda_{b_1} \wedge \lambda_{c_2} \Rightarrow \theta \qquad \lambda_{a_1} \wedge \lambda_{b_2} \wedge \lambda_{c_3} \Rightarrow \theta \tag{5}$$

For the moment, assume that we have not merged parameter variables. For each parameter variable $\theta_{x_i|u_1,u_2,\ldots,u_n}$, rather than asserting ENC1 parameter clauses, we assert the following clause:

**ENC3 Parameter Clause:**

$$\lambda_{u_1} \wedge \lambda_{u_2} \wedge \ldots \wedge \lambda_{u_n} \wedge \lambda_{x_i} \Rightarrow \theta_{x_i|u_1,u_2,\ldots,u_n}$$

Changing the equivalence to an implication is equivalent to introducing additional (unintended) models into the logical theory. However, these unintended models can be filtered since their cardinalities (the number of variables they set to true) are larger than the cardinality of original models (which all have the same cardinality) [1]. An operation called minimization performs exactly the desired filtering operation. We now provide an example of how minimization restores the theory by removing unintended models. Consider a Bayesian network having just a single binary variable $X$ with values $x_1$ and $x_2$, where $\mathsf{Pr}(x_1) = 0.1$. For ENC3, the CNF variables are $I_{x_1}$, $I_{x_2}$, $P_{x_1}$, and $P_{x_2}$. The weights are as follows:

$$\begin{array}{ll} W(\neg I_{x_1}) = 1.0 & W(I_{x_1}) = 1.0 \\ W(\neg I_{x_2}) = 1.0 & W(I_{x_2}) = 1.0 \\ W(\neg P_{x_1}) = 1.0 & W(P_{x_1}) = 0.1 \\ W(\neg P_{x_2}) = 1.0 & W(P_{x_2}) = 0.9 \end{array}$$

The ENC3 CNF consists of four clauses as follows:

$$\begin{array}{ll} I_{x_1} \vee I_{x_2} & \neg I_{x_1} \vee \neg I_{x_2} \\ I_{x_1} \Rightarrow P_{x_1} & I_{x_2} \Rightarrow P_{x_2} \end{array}$$

The logic therefore has four models having the weights shown below:

| Model | Weight |
|---|---|
| (1) $I_{x_1}$, $\neg I_{x_2}$, $P_{x_1}$, $\neg P_{x_2}$ | $1 * 1 * 0.1 * 1 = 0.1$ |
| (2) $I_{x_1}$, $\neg I_{x_2}$, $P_{x_1}$, $P_{x_2}$ | $1 * 1 * 0.1 * 0.9 = 0.09$ |
| (3) $\neg I_{x_1}$, $I_{x_2}$, $P_{x_2}$, $\neg P_{x_1}$ | $1 * 1 * 0.9 * 1 = 0.9$ |
| (4) $\neg I_{x_1}$, $I_{x_2}$, $P_{x_2}$, $P_{x_1}$ | $1 * 1 * 0.9 * 0.1 = 0.09$ |

Observe that the models are not in one–to–one correspondence with instantiations of network variables. Since no evidence has been incorporated, we would want a weighted model count to be 1.0. However, the sum of the counts for these four models is 1.18. Now consider what happens when we minimize the theory described by the CNF. In this case, we remove models that set more than two variables to true, leaving us with models (1) and (3). Now the models are indeed in one–to–once correspondence with the instantiations of network varibles. In fact, the set of models is the same as would be produced by ENC1. Moreover, the weighted model count is now 1.0.

By using ENC3 parameter clauses, we can now safely represent all equal parameters within the same CPT by a single Boolean variable in the CNF encoding, provided that we minimize also. For the *pathfinder* network for example, this drops the number of Boolean variables needed to represent non–extreme parameters from $42,946$ to $2,186$, a 95% reduction! Similar reductions are obtained for many other networks; see Table 10. In general, minimizing a logical theory is expensive. However, minimizing a d–DNNF is linear in the size of the d–DNNF [33]. Consequently, minimizing increases compilation time by only a constant factor. The hope is that gains from encoding equal parameters will outweigh this modest loss by reducing compilation time and the size of the compiled representation (for more efficient online inference).

In addition to equal parameters, several other techniques are presented in [1] for improving performance, which we summarize next. The CNF compilation algorithm employs two key techniques. The first is variable splitting, which can be thought of as doing case analysis. The second is caching, so that one can avoid factoring the same CNF subset multiple times. Which variables the algorithm ends up splitting on can very much affect its running time, and the size of factorizations it generates. Moreover, the complexity of the caching scheme is proportional to the number of variables appearing in the cached CNF subset, as the state of such variables are used to generate keys that uniquely define CNFs. The following observations state interesting properties of our CNF encodings, which if passed to the factoring algorithm can significantly improve both the splitting and caching processes.

First, if two clauses share a parameter variable, then their indicators must be over the same network variables. This property allows the CNF factoring algorithm to restrict its splitting to indicator variables, which would be sufficient to decompose the problem into independent components (hence, no splitting/case analysis is needed on parameter variables). Second, given the structure of indicator and parameter clauses, the state of indicator variables are sufficient to characterize the state of parameter variables. This property allows us to only involve indicator variables when generating CNF keys during the caching process. Both of the above optimizations can be exploited by simply identifying parameter variables to the factoring algorithm.

Another technique involves the construction of a decomposition tree (dtree) for the given Bayesian network, and then converting it into a dtree for its CNF encoding, which is used to drive the compilation algorithm. A dtree for a Bayesian network is simply a binary tree whose leaves correspond to the network CPTs [8]. A dtree for a CNF is also a binary tree, but its leaves correspond to the CNF clauses. Since each clause in the CNF encoding is generated by a CPT, we can convert a network dtree into a CNF dtree by simply unfolding each dtree node corresponding to a CPT into a subtree whose leaves correspond to the clauses generated by that CPT. The main point of this technique is to more efficiently generate dtrees for very large CNF encodings that are generated by Bayesian networks with a small number of CPTs (this happens when the network contains very large CPTs).

Our CNF encodings utilize some additional enhancements, two of which are described next. First, we define a new type of clause, called an *eclause*, which has the same syntax as a regular clause but stronger semantics: it asserts that *exactly* one of its literals is true. We use eclauses for representing indicator clauses, therefore reducing the size of CNFs considerably in networks having multi–valued variables. Moreover, we outfit the DPLL procedure used in factoring the CNF to work directly with eclauses, without having to unfold them into regular clauses. For another optimization example, the indicators and parameters corresponding to the same state of a root variable are logically equivalent, making it possible to delete the parameter variables and the corresponding parameter clauses, which establish the equivalence.

Experiments reported in [1] show that by incorporating equal parameters, splitting on indicator variables only, caching only indicators, and generating a dtree from the network, large improvements can be obtained over ENC1 both in offline compilation time and in online inference time. Note that the purpose here is not simply to compare two encodings but to show what can be gained by communicating additional structure to the WMC algorithm. Table 13 shows some of the results from that paper. The most important point is the improvement to compile times (vs. ENC1) and improvement in online inference times (vs. jointree). Improvements are order–of–magnitude or more in many cases. Moreover, in some cases, the additional use of local structure allowed compilation to succeed, where it failed previously.

In the context of compilation, we have reviewed how ENC1 can be changed into ENC3 to capture local structure in addition to determinism, most notably by encoding equal parameters. A search algorithm could utilize ENC3 and other advances discussed in this section in the same way that compilation does. Although the same techniques cannot be applied *as–is* to ENC2, it is likely that there exist other methods to capture equal parameters within ENC2.

We close this section with another empirical comparison that has not previously been addressed in the literature, between compilation using ENC2 and compilation using ENC3 and the other enhancements described in this section, in the same manner as we compared ENC1 to ENC3. To make the

Table 13
Results from [1] comparing ENC3 and other advances to ENC1 compile times and ENC3 and other advances to jointree online inference times. Dash indicates failure. Online time is averaged over sixteen evidence sets, where for each evidence set, we compute probability of evidence and a posterior marginal for every network variable.

| | Offline Compile Time (s) | | | Avg. Online Inf. Time (s) | | |
|---|---|---|---|---|---|---|
| Network | ENC1 | ENC3 | Improv. | Jointree | ENC3 | Improv. |
| alarm | 0.93 | 0.52 | 1.8 | 0.004 | 0.001 | 6.4 |
| bm-5-3 | 2.51 | 1.11 | 2.3 | 10.322 | 0.006 | 1,814.8 |
| diabetes | — | 2269.05 | — | 2.318 | 1.017 | 2.3 |
| hailfinder | 2.26 | 0.86 | 2.6 | 0.008 | 0.004 | 2.2 |
| mildew | — | 7483.8 | — | 0.837 | 0.209 | 4.0 |
| mm-3-8-3 | 7.44 | 1.87 | 4 | 15.509 | 0.013 | 1,181.6 |
| munin1 | — | 1534.97 | — | 82.605 | 2.807 | 29.4 |
| munin2 | 3248.42 | 225.46 | 14.4 | 1.770 | 0.412 | 4.3 |
| munin3 | 1553.43 | 151.72 | 10.2 | 1.168 | 0.228 | 5.1 |
| munin4 | 2440.3 | 677.92 | 3.6 | 8.621 | 0.481 | 17.9 |
| pathfinder | 226.37 | 20.36 | 11.1 | 0.105 | 0.004 | 23.7 |
| pigs | 110.1 | 17.84 | 6.2 | 0.216 | 0.100 | 2.2 |
| students-3-2 | 1.53 | 0.82 | 1.9 | 3.499 | 0.004 | 799.7 |
| tcc4f.obfuscated | 4.11 | 1.15 | 3.6 | 0.009 | 0.003 | 2.8 |
| water | 34.82 | 4.83 | 7.2 | 1.418 | 0.013 | 107.5 |

comparison as fair as possible, for each network evaluated, we use the same dtree for both encodings and compile using the C2D compiler. Table 14 shows, for two sets of networks and both encodings, the CNF size, the compile time, and the compilation size. Recall that in general the CNF produced by ENC2 could be smaller than that produced by ENC1, and that there is a similar decrease in compile time. Comparing ENC2 to ENC3 shows that in general the CNF sizes are much more similar. Despite the similar size, ENC3 consistently outperforms ENC2 in compile time, especially on the networks in the second group, which have non–binary variables, larger CPTs, and lesser amounts of determinism. Recall also that ENC2 produced compilations of roughly the same size as ENC1 and failed on exactly the same networks. In contrast, we see that when there is local structure other than determinism, as in the case of the second set of networks, ENC3 is capable of producing significantly smaller compilations and succeeds more often.

Table 14
Comparison of ENC3 and other advances with ENC2; compilation performed using the C2D compiler on a 1.6GHz Pentium M with 2GB of memory. Dash indicates failure.

| | ENC2 | ENC3 | ENC2 | ENC3 | ENC2 | ENC3 |
|---|---:|---:|---:|---:|---:|---:|
| | CNF | CNF | Comp. | Comp. | Comp. | Comp. |
| Network | Size | Size | Time (s) | Time (s) | Edges | Edges |
| Grid-50-16-1 | 5,112 | 5,624 | 127.97 | 95.69 | 14,869,887 | 15,000,534 |
| Grid-50-16-2 | 5,316 | 5,828 | 207.22 | 145.98 | 21,872,882 | 21,871,736 |
| Grid-50-16-3 | 4,960 | 5,472 | 53.20 | 40.79 | 6,052,060 | 6,042,590 |
| Grid-50-16-4 | 5,164 | 5,676 | 162.44 | 117.56 | 16,703,733 | 16,405,776 |
| Grid-50-16-5 | 5,240 | 5,752 | 150.67 | 115.39 | 13,867,070 | 13,905,110 |
| Grid-50-16-6 | 5,168 | 5,680 | 57.60 | 42.34 | 5,294,194 | 4,878,566 |
| Grid-50-16-7 | 5,068 | 5,580 | 115.70 | 91.36 | 12,366,429 | 12,400,554 |
| Grid-50-16-8 | 5,120 | 5,632 | 52.50 | 41.46 | 5,817,835 | 5,821,550 |
| Grid-50-16-9 | 5,560 | 6,072 | 67.60 | 52.89 | 6,199,093 | 6,164,233 |
| Grid-50-16-10 | 5,368 | 5,880 | 98.18 | 71.52 | 10,670,540 | 10,416,538 |
| bm-5-3 | 10,012 | 12,011 | 0.20 | 0.26 | 18,982 | 18,770 |
| mm-3-8-2 | 11,880 | 14,401 | 1.26 | 1.56 | 284,901 | 276,198 |
| st-3-2 | 4,323 | 5,663 | 0.23 | 0.26 | 28,342 | 28,277 |
| alarm | 4,111 | 3,342 | 0.14 | 0.25 | 5,621 | 2,711 |
| diabetes | 1,498,051 | 1,261,325 | – | 2,308.67 | – | 15,388,737 |
| hailfinder | 20,052 | 16,546 | 0.53 | 0.27 | 25,676 | 15,780 |
| mildew | 1,619,395 | 1,494,824 | 2,713.99 | 1,583.02 | 2,752,047 | 2,601,210 |
| munin2 | 294,528 | 217,470 | 1,518.62 | 293.68 | 6,144,328 | 3,757,857 |
| munin3 | 303,670 | 222,687 | 671.36 | 185.86 | 5,344,186 | 2,542,759 |
| munin4 | 335,422 | 261,017 | – | 409.58 | – | 8,217,229 |
| pathfinder | 325,890 | 337,901 | 14.94 | 7.76 | 159,716 | 35,852 |
| pigs | 24,689 | 26,187 | 47.49 | 22.63 | 2,207,534 | 1,606,319 |
| tcc4f | 26,056 | 26,287 | 0.69 | 0.41 | 38,938 | 23,052 |
| water | 75,778 | 77,524 | 8.19 | 3.69 | 265,141 | 101,061 |

### *5.2 Decomposability*

We review work from [3] in this section, which introduced an encoding method that retains the advantages of ENC3 while making it easier for algorithms to benefit computationally from the encoded local structure. The work is based

| $A$ | $B$ | $C$ | $\mathsf{Pr}(c\|a,b)$ | |
|---|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | 0.7 | $(\theta_1)$ |
| $a_1$ | $b_1$ | $c_2$ | 0.0 | (false) |
| $a_1$ | $b_1$ | $c_3$ | 0.3 | $(\theta_2)$ |
| $a_1$ | $b_2$ | $c_1$ | 0.4 | $(\theta_3)$ |
| $a_1$ | $b_2$ | $c_2$ | 0.3 | $(\theta_2)$ |
| $a_1$ | $b_2$ | $c_3$ | 0.3 | $(\theta_2)$ |
| $a_2$ | $b_1$ | $c_1$ | 0.333 | $(\theta_4)$ |
| $a_2$ | $b_1$ | $c_2$ | 0.333 | $(\theta_4)$ |
| $a_2$ | $b_1$ | $c_3$ | 0.333 | $(\theta_4)$ |
| $a_2$ | $b_2$ | $c_1$ | 0.2 | $(\theta_5)$ |
| $a_2$ | $b_2$ | $c_2$ | 0.3 | $(\theta_2)$ |
| $a_2$ | $b_2$ | $c_3$ | 0.5 | $(\theta_6)$ |

(a)

$$\begin{array}{l}
\lambda_{a_1} \wedge \lambda_{b_1} \wedge \lambda_{c_1} \Rightarrow \theta_1 \\
\neg\lambda_{a_1} \vee \neg\lambda_{b_1} \vee \neg\lambda_{c_2} \\
\lambda_{a_1} \wedge \lambda_{b_1} \wedge \lambda_{c_3} \Rightarrow \theta_2 \\
\lambda_{a_1} \wedge \lambda_{b_2} \wedge \lambda_{c_1} \Rightarrow \theta_3 \\
\lambda_{a_1} \wedge \lambda_{b_2} \wedge \lambda_{c_2} \Rightarrow \theta_2 \\
\lambda_{a_1} \wedge \lambda_{b_2} \wedge \lambda_{c_3} \Rightarrow \theta_2 \\
\lambda_{a_2} \wedge \lambda_{b_1} \wedge \lambda_{c_1} \Rightarrow \theta_4 \\
\lambda_{a_2} \wedge \lambda_{b_1} \wedge \lambda_{c_2} \Rightarrow \theta_4 \\
\lambda_{a_2} \wedge \lambda_{b_1} \wedge \lambda_{c_3} \Rightarrow \theta_4 \\
\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_1} \Rightarrow \theta_5 \\
\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_2} \Rightarrow \theta_2 \\
\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_3} \Rightarrow \theta_6
\end{array}$$

(b)

$$\begin{array}{l}
\lambda_{a_1} \wedge \lambda_{b_1} \wedge \lambda_{c_1} \Rightarrow \theta_1 \\
\neg\lambda_{a_1} \vee \neg\lambda_{b_1} \vee \neg\lambda_{c_2} \\
\lambda_{a_1} \wedge \lambda_{c_3} \Rightarrow \theta_2 \\
\lambda_{b_2} \wedge \lambda_{c_2} \Rightarrow \theta_2 \\
\lambda_{a_1} \wedge \lambda_{b_2} \wedge \lambda_{c_1} \Rightarrow \theta_3 \\
\lambda_{a_2} \wedge \lambda_{b_1} \Rightarrow \theta_4 \\
\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_1} \Rightarrow \theta_5 \\
\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_3} \Rightarrow \theta_6
\end{array}$$

(c)

Fig. 4. (a) A CPT over three variables, (b) Clauses generated by the encoding from [1] for the CPT, and (c) an equivalent encoding.

on the observation that CNF syntax can sometimes stand in the way of recognizing independent components when running a model counting algorithm, since these algorithms typically depend on syntactic checks for identifying independent components. Hence, the approach tries to empower the syntactic identification of components by preprocessing the encoding in order to simplify it and make it more semantically revealing.

For an example, consider Figure 4(a) and observe that given values for certain variables, other variables sometimes become irrelevant. For example, given $A = a_2$ and $B = b_1$, the probability no longer depends upon $C$ ($C$ has a uninform probability). Moreover, given values $A = a_1$ and $C = c_3$, variable $B$ becomes irrelevant to the probability. This phenomenon is similar to, but more general than, context–specific–independence (CSI) [10] and can be very powerful. Yet, search algorithms can fail to exploit this structure, even if it is encoded correctly. For example, even though this structure is encoded by the clauses in Figure 4(b) and the equivalent ones in Figure 4(c), it was shown empirically and analytically in [3] that search algorithms will better exploit this structure when applied to the simplified clauses in Figure 4(c). The main reason is that the simplified clauses will tend to have fewer occurrences of irrelevant variables as we set variables in search process, allowing one to more easily recognize independent components based on syntactic checks.

Before defining a general procedure for simplifying the clauses of a given CPT as discussed above, we observe that because we are working with multi–valued variables, it makes sense to use a multi–valued form of resolution. We therefore define a logic over multi–valued variables $\mathbf{X}$. The syntax of the logic is identical to that of standard propositional logic, except that an *atom* is an assignment to a variable in $\mathbf{X}$ of a value in its domain. For example, $C = c_2$ is an atom. The semantics is also like that of standard propositional logic, except that a *world*, which consists of an atom for each variable, satisfies an atom iff it assigns the common variable the same value. Within this logic, a *term* over $\mathbf{X}' \subseteq \mathbf{X}$ is a conjunction of atoms, one for each variable in $\mathbf{X}'$. Let $\Gamma$ be a disjunction of terms over $\mathbf{X}$. An *implicant* $\gamma$ of $\Gamma$ is a term over $\mathbf{X}' \subseteq \mathbf{X}$ that implies $\Gamma$. A *prime implicant* $\gamma$ of $\Gamma$ is an implicant that is minimal in the sense that the removal of any atom would result in a term that is no longer an implicant of $\Gamma$.

**Algorithm 1** EncodeCPT($\phi$: CPT) Generates a set of clauses for $\phi$.

```
Partition the rows of $\phi$ into encoding groups
for each encoding group $\Gamma$ do
  $M \leftarrow$ terms of $\Gamma$
  $\theta \leftarrow$ consequent of $\Gamma$
  $P \leftarrow$ the prime implicants of $M$
  for $p$ in $P$ do
    $I \leftarrow$ encoding of $p$
    if $\theta = 0$ then
      assert clause $\neg I$
    else
      assert clause $I \Rightarrow \theta$
    end if
  end for
end for
```

Given these definitions, we can encode the network by generating the same CNF variables and indicator clauses as in ENC3 and by generating clauses for each CPT according to Algorithm 1. We are also able to use the other improvements (e.g., branching on indicators only) from the previous section. This algorithm encodes a CPT $\phi$ over variables $\mathbf{X}$ by first partitioning the CPT into *encoding groups,* which are sets of rows that share the same parameter value. Note that each row in the CPT induces a term over variables $\mathbf{X}$ and so each encoding group induces a set of terms. Moreover, the terms within an encoding group will share a common parameter variable or all correspond to falsehood. We refer to this variable (or falsehood) as the consequent of the encoding group. To process encoding group $\Gamma$, we find the prime implicants of $\Gamma$'s terms, and for each prime implicant $p$, we assert a clause $I \Rightarrow \theta$, where $I$ is the conjunction of indicators corresponding to $p$, and $\theta$ is the consequent of the encoding group. If the parameter $\theta$ equals 0, we simply generate the

| Encoding | Param | | Conse- | Prime | |
|---|---|---|---|---|---|
| Group | Value | Terms | quent | Implicants | Encoding |
| $\Gamma_1$ | .7 | $a_1b_1c_1$ | $\theta_1$ | $a_1b_1c_1$ | $\lambda_{a_1} \wedge \lambda_{b_1} \wedge \lambda_{c_1} \Rightarrow \theta_1$ |
| $\Gamma_2$ | 0 | $a_1b_1c_2$ | false | $a_1b_1c_2$ | $\neg\lambda_{a_1} \vee \neg\lambda_{b_1} \vee \neg\lambda_{c_2}$ |
| $\Gamma_3$ | .3 | $a_1b_1c_3, a_1b_2c_2,$ $a_1b_2c_3, a_2b_2c_2$ | $\theta_2$ | $a_1c_3, b_2c_2$ | $\lambda_{a_1} \wedge \lambda_{c_3} \Rightarrow \theta_2,$<br>$\lambda_{b_2} \wedge \lambda_{c_2} \Rightarrow \theta_2$ |
| $\Gamma_4$ | .4 | $a_1b_2c_1$ | $\theta_3$ | $a_1b_2c_1$ | $\lambda_{a_1} \wedge \lambda_{b_2} \wedge \lambda_{c_1} \Rightarrow \theta_3$ |
| $\Gamma_5$ | .333 | $a_2b_1c_1, a_2b_1c_2,$ $a_2b_1c_3$ | $\theta_4$ | $a_2b_1$ | $\lambda_{a_2} \wedge \lambda_{b_1} \wedge \Rightarrow \theta_4$ |
| $\Gamma_6$ | .2 | $a_2b_2c_1$ | $\theta_5$ | $a_2b_2c_1$ | $\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_1} \Rightarrow \theta_5$ |
| $\Gamma_7$ | .5 | $a_2b_2c_3$ | $\theta_6$ | $a_2b_2c_3$ | $\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_3} \Rightarrow \theta_6$ |

Fig. 5. Encoding a CPT using prime implicants.

clause $\neg I$. Figure 5 demonstrates this algorithm for the CPT in Figure 4(a).

The algorithm we use to find prime implicants is an extension of the venerable Quine-McCluskey (QM) algorithm (e.g., [37]). QM works only for binary variables, so we extend it to multi–valued variables in a straightforward manner. Extensions of the QM algorithm for multi–valued variables are common, some of them defining a prime implicant differently (e.g., [38]). The definition given here was found effective for the purpose at hand.

The new encoding method, which we call ENC4, defines a structured resolution strategy. The strategy is structured in the sense that rather than working on a set of clauses, the strategy works on a partition of clauses, and restricts resolution to clauses within the same element of the partition. Each element in the partition corresponds to clauses belonging to the same CPT and having the same consequent.

We now make two important observations. First, even though computing prime implicants can be expensive in general, [3] provides extensive experiments showing that the increase in encoding time is actually negligible (we omit these results for brevity). This efficiency stems from the small number of variables involved in the computation (those appearing in a CPT). Second, the primary advantage of ENC4 is the removal of spurious occurrences of literals, transforming a set of terms into a more minimal set. This removal of literals allows us to decompose more often without resorting to splitting.

Experimental results in [3] show that ENC4 can significantly outperform ENC3 in both compile time and the size of the resulting compilation. We repeat a few of those results in Table 15. Compile times improve anywhere from 1.45 times to over 17 times. Moreover, many of the grid networks and also barley caused the compiler to run out of memory (as indicated by dashes) when applied to ENC3 but compiled successfully using ENC4. The last three columns show the improvement to the size (number of edges) of the resulting compilations. Here, we see that on networks where ENC3 was successful, ENC4 sizes were sometimes comparable and otherwise significantly reduced.

### *5.3 Evidence*

It is well–known that exploiting evidence can make inference in a Bayesian network more tractable. Two of the most common techniques are removing leaf nodes that are not part of the evidence or query [39] and removing edges outgoing from observed nodes [40]. These preprocessing steps, which we call *classical pruning*, can significantly reduce the connectivity of the network, making accessible many queries that would be inaccessible otherwise. Although classical pruning can be very effective, one can identify situations where it does not exploit the full power of evidence, especially when the network contains local structure. The investigation in [2] demonstrates how WMC provides a natural and effective method of exploiting evidence that can provide much more benefit than classical pruning. The paper works within the context of ENC1 and compilation. This section reviews that work and then says a few words on how these same concepts also apply to search algorithms and other encodings.

In Section 2.2, we described two ways of incorporating evidence into the WMC framework. Within the context of compilation, the first method compiles the CNF into an AC without evidence, and for each query, adjusts the weights of indicators to account for the evidence. This approach has the advantage that the compiled AC can answer multiple queries with respect to any evidence. However, sometimes the compilation task is too difficult, causing the compiler to take too long or run out of memory. Even when compilation succeeds, the resulting AC may be too large for some tasks. The second method is described in this section and works by conjoining the CNF with unit clauses that encode the evidence $\mathbf{e}$ prior to compiling, thereby eliminating from the theory models inconsistent with the evidence. The advantage is that compilation can become more tractable, and the resulting AC may be much smaller. The disadvantage is that the resulting AC is then only good for answering queries with respect to evidence that is a superset of $\mathbf{e}$.

The benefit of compiling with evidence may seem illusory at first, since it restricts the set of queries. However, one of the contributions of [2] is to identify

Table 15
Results for compiling a number of networks using ENC3 and the ENC4 encoding.

| | Max. | ENC3 | ENC4 | Imp- | ENC3 | ENC4 | Imp- |
|---|---|---|---|---|---|---|---|
| | Clst. | Comp. | Comp. | rove- | Comp. | Comp. | rove- |
| Network | Size | Time(s) | Time(s) | ment | Size | Size | ment |
| s1238 | 61.0 | 11.32 | 1.83 | 6.19 | 853,987 | 263,786 | 3.24 |
| s526n | 18.0 | 0.23 | 0.12 | 1.92 | 10,088 | 10,355 | 0.97 |
| s526 | 18.0 | 0.22 | 0.14 | 1.57 | 13,352 | 14,143 | 0.94 |
| s641 | 19.0 | 2.54 | 0.38 | 6.68 | 78,071 | 36,555 | 2.14 |
| s1494 | 48.0 | 1.82 | 0.44 | 4.14 | 419,274 | 85,469 | 4.91 |
| s838.1 | 13.0 | 0.43 | 0.20 | 2.15 | 49,856 | 30,899 | 1.61 |
| bm-05-03 | 19.0 | 0.29 | 0.20 | 1.45 | 19,190 | 10,957 | 1.75 |
| bm-15-03 | 62.0 | 254.96 | 44.07 | 5.79 | 7,351,823 | 1,460,842 | 5.03 |
| bm-22-03 | 107.0 | 4,869.64 | 748.13 | 6.51 | 72,169,022 | 14,405,730 | 5.01 |
| or-60-20-1 | 24.0 | 338.48 | 54.47 | 6.21 | 6,968,339 | 7,777,867 | 0.90 |
| or-60-20-3 | 25.0 | 1.40 | 0.77 | 1.82 | 104,275 | 119,779 | 0.87 |
| or-60-20-5 | 27.0 | 728.36 | 118.17 | 6.16 | 17,358,747 | 14,986,497 | 1.16 |
| or-60-20-7 | 26.0 | 250.72 | 97.13 | 2.58 | 11,296,613 | 12,510,488 | 0.90 |
| or-60-20-9 | 25.0 | 19.58 | 7.17 | 2.73 | 1,011,193 | 1,060,217 | 0.95 |
| gr-50-16-1 | 24.0 | 137.25 | 43.95 | 3.12 | 14,692,963 | 5,739,854 | 2.56 |
| gr-50-16-3 | 24.0 | 65.03 | 40.45 | 1.61 | 7,755,318 | 5,280,027 | 1.47 |
| gr-50-16-5 | 25.0 | - | 26.70 | - | - | 3,431,139 | - |
| gr-50-16-7 | 24.0 | 51.68 | 2.99 | 17.28 | 6,413,897 | 421,060 | 15.23 |
| gr-50-16-9 | 24.0 | - | 60.55 | - | - | 7,360,872 | - |
| gr-50-18-1 | 27.0 | 411.45 | 48.36 | 8.51 | 39,272,847 | 6,451,916 | 6.09 |
| gr-50-18-3 | 27.0 | 362.90 | 29.18 | 12.44 | 32,120,267 | 2,507,215 | 12.81 |
| gr-50-18-5 | 27.0 | - | 158.13 | - | - | 18,291,116 | - |
| gr-50-18-7 | 27.0 | - | 79.97 | - | - | 9,439,318 | - |
| gr-50-18-9 | 27.0 | - | 68.51 | - | - | 7,890,645 | - |

many practical situations where restricting evidence in this way can be very practical. First, the evidence may be fixed on only a subset of the variables, leaving room for posing a large number of queries with respect to other variables (this happens in MAP algorithms, e.g., [41–43]). Second, one may be interested in estimating the value of network parameters which will maximize the probability of given evidence (this happens, for example, in genetic linkage analysis [44,45]). In this case, one may want to use iterative algorithms such as EM or gradient ascent [46], which pose many network queries with respect to the given evidence but different network parameter values. A similar application appears in sensitivity analysis [47], where the goal is to search for some network parameters that satisfy a given constraint . The change to the encoding of the network into CNF is simple: for each network variable $X$ set to value $x$ by the evidence, assert the unit clause $\lambda_x$ into the CNF. However, the effect of this seemingly innocent action can belie its true power. Several detailed examples are provided in [2], which demonstrate how the introduction of such unit clauses can reduce the work required of the compilation algorithm. We repeat one of these examples next.

The example is from genetic linkage analysis, and is a common occurrence in that domain. It involves four variables: child $C$ with parents $A, B$, and $S$. The variable $C$ is the genotype in a child which is inherited from one of the parent's genes, $A/B$, based on the value of selector $S$. We assume that all four variables are binary and that the portion of the CPT with $S = s_1$ is as follows, [11]

| $S$ | $A$ | $B$ | $C$ | $Pr(C\|A, B)$ |
|---|---|---|---|---|
| $s_1$ | $a_1$ | $b_1$ | $c_1$ | 1.0 |
| $s_1$ | $a_1$ | $b_2$ | $c_1$ | 1.0 |
| $s_1$ | $a_2$ | $b_1$ | $c_2$ | 1.0 |
| $s_1$ | $a_2$ | $b_2$ | $c_2$ | 1.0 |

As described in Section 4.1, the algorithm on which the compilation is based works by repeatedly conditioning to decompose the CNF. Let us consider the case where we are given evidence $\{c_1\}$, and during compilation, we condition on $S = s_1$. Assuming a proper encoding of the network into CNF, combining the evidence with the value for $S$ allows us to infer $a_1$, which unit resolution can use to achieve further gains. Conditioning on $S = s_2$ yields a similar conclusion for $b_1$. In this case, the full power of conditioning on $S$ is realized only when combined with evidence on $C$. This example reveals how evidence can combine with the operations of the compilation algorithm to simplify the task.

Recall that classical pruning severs edges leaving evidence nodes and deletes certain leaf nodes. Injecting unit clauses is analogous to this severing of edges but is strictly more powerful for several reasons. First, this technique not only

[11] In general, the variables are multi–valued, and this discussion also applies in this case.

exploits the fact that a variable has been instantiated, but also exploits the specific value to which it has been instantiated. Second, rather than simply affecting the CPTs of children of evidence nodes, injecting unit clauses can affect many more parts of the network since unit clauses will often allow the WMC algorithm to infer additional unit clauses, and the effects can propagate to many ancestors and many descendants of evidence nodes. Third, rather than only realizing a limited number of gains during initialization, injecting unit clauses can continue to realize gains throughout the WMC algorithm. [12]

Several results are given in [2] demonstrating large gains when compiling with evidence. Algorithms that exploit only topological structure could not perform inference on many of the data sets, even after performing classical pruning, because of high treewidth. Furthermore, in the majority of cases, applying classical pruning and compiling the CNF without the introduction of the unit clauses based on the evidence also failed. However, with the introduction of the unit clauses, compilation became possible in many cases. Moreover, the paper showed that the performance of this *general* technique subsumed the performance of the *specialized* quickscore algorithm [13], which capitalizes on evidence in certain types of diagnostic networks. Finally, the paper showed that when combined with some aggressive preprocessing and applied to several difficult problems from genetic linkage analysis, the technique outperformed SUPERLINK 1.4, a state–of–the–art system for the task, on a number of challenging problems.

Table 16 lists a few of the results from the paper from the field of genetic linkage analysis and compares the performance to that of SUPERLINK. There are several observations. First, general–purpose algorithms that exploit only topological structure such as jointree could solve only one of the listed networks, because of high treewidth, even after applying classical pruning techniques. Second, only one of these networks could be compiled without the introduction of unit clauses to capture evidence. However, once the unit clauses were injected, all of the networks yielded to compilation in minutes. Finally, WMC compilation times are in most cases more efficeint than SUPERLINK online times, and WMC online times are much more efficient still. Given that compilation must occur once, and online inference must be repeated many times, this effect of this improvement multiplies.

We conclude by noting that ENC2, ENC3, and ENC4 can effectively take advantage of evidence in the way described in this section, since they utilize indicator variables in the same way as ENC1. Furthermore, performing WMC by search can utilize evidence by examining the weights of variables and as-

[12] If we are not interested in computing posterior marginals on some variables $\mathbf{M}$, we can achieve even larger gains by deleting leaf nodes that are not part of the evidence and not part of $\mathbf{M}$, as with classical pruning.

Table 16
EE results from [2] with full preprocessing.

| Net | Max<br>Clust. | Comp.<br>Time (s) | Comp.<br>Size | Oneline<br>Time (s) | SUPERLINK<br>Time (s) |
|---|---|---|---|---|---|
| ee33 | 20.2 | 25.33 | 2,070,707 | 0.59 | 1046.72 |
| ee37 | 29.6 | 61.29 | 1,855,410 | 0.39 | 1381.61 |
| ee30 | 35.9 | 376.78 | 27,997,686 | 8.37 | 815.33 |
| ee23 | 38.0 | 89.47 | 3,986,816 | 1.08 | 502.02 |
| ee18 | 41.5 | 283.96 | 23,632,200 | 6.63 | 248.11 |

serting a negative unit clause any time a weight is equal to 0. In the case of compilation, the disadvantage of incorporating evidence was that compilation would need to be performed again for some queries, which removes one of the chief advantages of compiling (although we have seen that in many practical cases, this is not necessary). However, in the case of a search, the algorithm is re–run for each new evidence anyway, so there is really no disadvantage to incorporating evidence in this case. Encoding evidence in the context of search algorithms was indeed applied effectively in [16].

## 6 When WMC Offers No Advantage

In previous sections, we have demonstrated that WMC can outperform algorithms that exploit only topological structure, such as jointree and variable elimination. However, because WMC incurs overhead looking for ways to exploit local structure, if there is insufficient local structure, then WMC may not be the best choice. For example, in the absence of local structure, variable elimination will outperform WMC by search. The difference will be a constant factor, but possibly a very large one.

We discuss the effect of limited local structure on compilation in the context of one last experiment, which has not previously been addressed in the literature. We first identified a set of networks having treewidth small enough for the jointree algorithm to work. For each network, we removed all local structure by setting each parameter to a random number and then normalizing appropriately. We then compiled the network using ACE (which uses ENC4) and answered a set of queries using both ACE and jointree. To make the comparison as fair as possible, we used the same elimination order to drive ACE compilation and to construct the jointree. Experiments ran on a 2.13GHz Intel Core Duo processor with 4GB of RAM.

Table 17 demonstrates that in the absence of local structure, WMC compi-

Table 17
Ace vs. Jointree when there is no local structure. Online time is averaged over sixteen evidence sets, where for each evidence set, we compute probability of evidence and a posterior marginal for every network variable.

| | Ace Offline | Jointree Avg. | Ace Avg. | |
|---:|---:|---:|---:|---:|
| Network | Time (s) | Online Time (s) | Online Time (s) | Improv. |
| alarm | 1 | 0.007 | 0.005 | 1.41 |
| bm-5-3 | 721 | 3.328 | 3.965 | 0.84 |
| diabetes | 1,345 | 1.202 | 1.268 | 0.95 |
| hailfinder | 3 | 0.018 | 0.007 | 2.66 |
| mm-3-8-3 | 195 | 1.117 | 1.336 | 0.84 |
| munin2 | 284 | 0.764 | 0.596 | 1.28 |
| munin3 | 254 | 0.495 | 0.534 | 0.93 |
| munin4 | 1,248 | 1.770 | 1.872 | 0.95 |
| pathfinder | 37 | 0.062 | 0.036 | 1.72 |
| pigs | 41 | 0.115 | 0.123 | 0.93 |
| students-3-2 | 241 | 0.961 | 1.806 | 0.53 |
| tcc4f.obfuscated | 3 | 0.022 | 0.007 | 3.17 |
| water | 340 | 0.659 | 0.591 | 1.12 |

lation will incur significant overhead during the offline phase (more than if local structure were present), and no online gains will be realized from the extra work. Online times will be roughly equivalent to jointree times. Online space will be somewhat larger than jointree space, since the WMC compilation stores both nodes and edges, whereas a jointree can be seen to describe a similarly sized compilation, yet only nodes need be stored explicitly (see [48] for details).

## 7 Conclusion

We provided in this paper a survey and some new results on a class of approaches for exact probabilistic inference, which reduces the problem to one of weighted model counting (WMC) on a CNF encoding of a Bayesian network. If the network exhibits sufficient local structure, and if this structure is captured during the encoding process, then WMC techniques can efficiently handle networks that have very high treewidths (i.e., ones that are outside the scope of algorithms that exploit only topological structure). The advantages of

WMC approaches include a declarative method for encoding local structure, an ability to leverage highly refined techniques from satisfiability solving, and a powerful way for exploiting available evidence. Recent literature on WMC approaches provides different ways of encoding a Bayesian network, different ways of performing WMC (e.g., search vs compilation), and different degrees of utilizing local structure. Our survey of recent WMC approaches has been made systematic by placing these approaches across these three dimensions.

## References

[1] M. Chavira, A. Darwiche, Compiling Bayesian networks with local structure, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI), 2005, pp. 1306–1312.

[2] M. Chavira, D. Allen, A. Darwiche, Exploiting evidence in probabilistic inference, in: Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI), 2005, pp. 112–119.

[3] M. Chavira, A. Darwiche, Encoding cnfs to empower component analysis, in: Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT), Springer Berlin / Heidelberg, Lecture Notes in Computer Science, Volume 4121, 2006, pp. 61–74.

[4] F. V. Jensen, S. Lauritzen, K. Olesen, Bayesian updating in recursive graphical models by local computation, Computational Statistics Quarterly 4 (1990) 269–282.

[5] S. L. Lauritzen, D. J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems, Journal of Royal Statistics Society, Series B 50 (2) (1988) 157–224.

[6] N. L. Zhang, D. Poole, Exploiting causal independence in Bayesian network inference, Journal of Artificial Intelligence Research 5 (1996) 301–328.

[7] R. Dechter, Bucket elimination: A unifying framework for probabilistic inference, in: Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI), 1996, pp. 211–219.

[8] A. Darwiche, Recursive conditioning, Artificial Intelligence 126 (1-2) (2001) 5–41.

[9] F. Jensen, S. K. Andersen, Approximations in Bayesian belief universes for knowledge based systems, in: Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence (UAI), Cambridge, MA, 1990, pp. 162–169.

[10] C. Boutilier, N. Friedman, M. Goldszmidt, D. Koller, Context–specific independence in Bayesian networks, in: Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI), 1996, pp. 115–123.

[11] D. Larkin, R. Dechter, Bayesian inference in the presence of determinism., in: C. M. Bishop, B. J. F. (eds) (Eds.), Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics (AI-STAT), Key West, FL., 2003.

[12] F. Bacchus, S. Dalmao, T. Pitassi, Value elimination: Bayesian inference via backtracking search, in: Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03), Morgan Kaufmann Publishers, San Francisco, CA, 2003, pp. 20–28.

[13] D. Heckerman, A tractable inference algorithm for diagnosing multiple diseases, in: Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence, 1989, pp. 174–181.

[14] D. Poole, N. Zhang, Exploiting contextual independence in probabilistic inference, Journal of Artificial Intelligence 18 (2003) 263–313.

[15] A. Darwiche, A logical approach to factoring belief networks, in: Proceedings of KR, 2002, pp. 409–420.

[16] T. Sang, P. Beame, H. Kautz, Solving Bayesian networks by weighted model counting, in: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05), Vol. 1, AAAI Press, 2005, pp. 475–482.

[17] M. Chavira, A. Darwiche, Compiling Bayesian networks using variable elimination, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), 2007, pp. 2443–2449.

[18] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, T. Pitassi, Combining component caching and clause learning for effective model counting., in: SAT, 2004.

[19] T. Sang, P. Beame, H. A. Kautz, Heuristics for fast exact model counting., in: SAT, 2005, pp. 226–240.

[20] A. Darwiche, On the tractability of counting theory models and its application to belief revision and truth maintenance, Journal of Applied Non-Classical Logics 11 (1-2) (2001) 11–34.

[21] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.

[22] M. Chavira, A. Darwiche, M. Jaeger, Compiling relational Bayesian networks for exact inference, International Journal of Approximate Reasoning 42 (1–2) (May 2006) 4–20.

[23] D. Roth, On the hardness of approximate reasoning, Artificial Intelligence 82 (1-2) (1996) 273–302.

[24] F. Bacchus, S. Dalmao, T. Pitassi, Algorithms and complexity results for #sat and bayesian inference., in: FOCS, 2003, pp. 340–351.

[25] M. Davis, G. Logemann, D. Loveland, A machine program for theorem proving, CACM 5 (1962) 394–397.

[26] A. Darwiche, New advances in compiling CNF to decomposable negational normal form, in: Proceedings of European Conference on Artificial Intelligence, 2004, pp. 328–332.

[27] R. Dechter, R. Mateescu, AND/OR search spaces for graphical models, Artificial Intelligence 171 (2-3) (2007) 73–106.

[28] W. Wei, B. Selman, A new approach to model counting., in: SAT, 2005, pp. 324–339.

[29] W. Wei, J. Erenrich, B. Selman, Towards efficient sampling: Exploiting random walk strategies., in: AAAI, 2004, pp. 670–676.

[30] R. Bayardo, J. Pehoushek, Counting models using connected components, in: AAAI, 2000, pp. 157–162.

[31] A. Darwiche, A compiler for deterministic, decomposable negation normal form, in: Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI), AAAI Press, Menlo Park, California, 2002, pp. 627–634.

[32] F. Bacchus, S. Dalmao, T. Pitassi, Dpll with caching: A new algorithm for #SAT and Bayesian inference, Electronic Colloquium on Computational Complexity (ECCC) 10 (003).

[33] A. Darwiche, P. Marquis, A knowlege compilation map, Journal of Artificial Intelligence Research 17 (2002) 229–264.

[34] J. Huang, A. Darwiche, Dpll with a trace: From sat to knowledge compilation, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI), 2005, pp. 156–162.

[35] M. Jaeger, Relational Bayesian networks, in: D. Geiger, P. P. Shenoy (Eds.), Proceedings of the 13th Conference of Uncertainty in Artificial Intelligence (UAI-13), Morgan Kaufmann, Providence, USA, 1997, pp. 266–273.

[36] A. Darwiche, A differential approach to inference in Bayesian networks, Journal of the ACM 50 (3) (2003) 280–305.

[37] J. P. Hayes, Introduction to Digital Logic Design, Addison Wesley, 1993.

[38] M. M. Mirsalehi, T. K. Gaylord, Logical minimization of multilevel coded functions, Applied Optics 25 (1986) 3078–3088.

[39] R. D. Shachter, Evaluating influence diagrams, Operations Research 34 (6) (1986) 871–882.

[40] S. Ross, Evidence absorption and propagation through evidence reversals, in: Proceedings of the 5th Annual Conference on Uncertainty in Artificial Intelligence (UAI-90), Elsevier Science Publishing Company, Inc., New York, NY, 1990.

[41] J. Park, A. Darwiche, Approximating MAP using stochastic local search, in: Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI), Morgan Kaufmann Publishers, Inc., San Francisco, California, 2001, pp. 403–410.

[42] J. Park, A. Darwiche, Solving map exactly using systematic search, in: Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI), 2003, pp. 459–468.

[43] C. Yuan, T.-C. Lu, M. Druzdzel, Annealed MAP, in: Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI), 2004, pp. 628–635.

[44] M. Fishelson, D. Geiger, Exact genetic linkage computations for general pedigrees., Bioinformatics 18 (1) (2002) 189–198.

[45] M. Fishelson, N. Dovgolevsky, D. Geiger, Maximum likelihood haplotyping for general pedigrees, Tech. Rep. CS-2004-13, Technion, Haifa, Israel (2004).

[46] A. Dempster, N. Laird, D. Rubin, Maximum likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society 39 (1) (1977) 1–38.

[47] H. Chan, A. Darwiche, Sensitivity analysis in Bayesian networks: From single to multiple parameters, in: Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI), AUAI Press, Arlington, Virginia, 2004, pp. 67–75.

[48] J. Park, A. Darwiche, A differential semantics for jointree algorithms, Artificial Intelligence 156 (2004) 197–216.