UCLA
UCLA Previously Published Works

Title
Encoding CNFs to empower component analysis

Permalink
https://escholarship.org/uc/item/3f9783cp

Journal
THEORY AND APPLICATIONS OF SATISFIABILITY TESTING - SAT 2006, PROCEEDINGS, 4121

ISSN
0302-9743

Authors
Chavira, M
Darwiche, A

Publication Date
2006

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Encoding CNFs to Empower Component Analysis*

Mark Chavira and Adnan Darwiche

Computer Science Department
University of California, Los Angeles
chavira,darwiche@cs.ucla.edu

Abstract. Recent algorithms for model counting and compilation work by decomposing a CNF into syntactically independent components through variable splitting, and then solving the components recursively and independently. In this paper, we observe that syntactic component analysis can miss decomposition opportunities because the syntax may hide existing semantic independence, leading to unnecessary variable splitting. Moreover, we show that by applying a limited resolution strategy to the CNF prior to inference, one can transform the CNF to syntactically reveal such semantic independence. We describe a general resolution strategy for this purpose, and a more specific one that utilizes problem–specific structure. We apply our proposed techniques to CNF encodings of Bayesian networks, which can be used to answer probabilistic queries through weighted model counting and/or knowledge compilation. Experimental results demonstrate that our proposed techniques can have a large effect on the efficiency of inference, reducing time and space requirements significantly, and allowing inference to be performed on many CNFs that exhausted resources previously.

1 Introduction

Recent algorithms for model counting [17, 6] and compilation [13] work by decomposing a CNF into syntactically independent components through variable splitting, and then solving the components recursively and independently. Critical to the efficiency of these *search with decomposition* algorithms is the early identification of independent components, which would minimize the amount of variable splitting required (a typical source of exponential behavior).

Search–with–decomposition algorithms consider two CNFs independent when they do not have variables in common, a condition which we call *syntactic independence.* Note, however, that even though two CNFs α and β may share variables (and are hence syntactically dependent), they may still be capable of being solved separately in two circumstances. First, there may exist CNFs α' and β' that encode the same semantics as α and β, respectively, and which do

* This work has been partially supported by Air Force grant #FA9550-05-1-0075-P00002 and JPL/NASA grant #1272258.

not have variables in common. This happens when one of the the CNFs α or β mentions irrelevant variables. Second, it may be that values of shared variables are implied by α and β, but removing subsumed clauses and performing unit resolution is insufficient to recognize this situation. If the information were known, then the variables could be set accordingly and the CNFs would thus become syntactically independent. Both of these situations cause decomposition algorithms to perform unnecessary splitting on the variables common to α and β. The phenomenon is not only present at the first level of decomposition, but can be exhibited at any level in the search tree, leading to compounded inefficiencies. As we demonstrate in this paper, the gap between syntactic independence and what we will call semantic independence can be bridged considerably by applying limited forms of resolution to the CNF, leading to major improvements to search–with–decomposition algorithms. In fact, the effect of such pre–processing can be much more dramatic if one pays attention to where the CNF originated.

				Resolution	
	Syntax 1	Syntax 2	Improvement	Strategy 1	Improvement
Circuit	Time (s)	Time (s)		Time (s)	
s510	0.09	0.06	1.55	0.06	1.50
s444	0.12	0.07	1.69	0.07	1.74
s382	0.12	0.07	1.78	0.07	1.73
s400	0.12	0.07	1.80	0.07	1.78
s420	0.21	0.07	3.19	0.07	3.19
s344	0.25	0.07	3.44	0.07	3.31
s349	0.25	0.07	3.47	0.08	3.29
s386	0.29	0.07	4.26	0.07	4.26
s838	0.86	0.19	4.55	0.14	6.22
s1238	7.48	0.99	7.59	1.03	7.29
s713	4.66	0.50	9.37	0.40	11.61
s526n	2.28	0.18	12.73	0.18	12.32
s1196	11.89	0.93	12.81	0.97	12.29
s526	2.27	0.18	12.82	0.18	12.61
s953	5.34	0.34	15.48	0.33	16.13
s641	5.20	0.32	16.30	0.33	15.67
s1488	2.78	0.13	21.24	0.13	21.24
s1494	2.89	0.13	22.43	0.13	22.25
s832	3.15	0.10	31.21	0.11	29.19
s838.1	2.95	0.08	38.80	0.08	38.80
s1423	timeout	63.78	n/a	49.94	n/a
s13207.1	timeout	186.61	n/a	199.49	n/a
s35932	timeout	2.83	n/a	3.09	n/a

Table 1. Cachet model count times for ISCAS89 circuits using three different CNF encodings. Timeout was four hours on a 2.40 GHz Intel Xeon CPU with 4GB of memory.

To demonstrate the effect of initial CNF syntax on the performance of search–with–decomposition algorithms, consider Table 1 which depicts the result of run-

ning a state–of–the–art model counter Cachet [17, 1] on two CNF encodings of ISCAS89 benchmark circuits (e.g., [2]).[1] We will have more to say about the two encodings later, but for now, suffice it to say that SYNTAX 1 is chosen carefully to worsen the gap between syntactic and semantic independence, and that SYNTAX 2 is chosen to bridge this gap. The first four columns (other columns will be discussed in Section 3) of Table 1 illustrate the dramatic performance difference between these two encodings, where SYNTAX 2 model count times range from 1.5 times faster on easy problems to over 38 times faster on harder ones, and where three networks could not be processed in four hours using SYNTAX 1 but required only minutes or less using SYNTAX 2.

The primary goal of this paper is to demonstrate that CNFs can be pre–processed, or carefully encoded, to better bridge the gap between syntactic and semantic independence. The approach we propose is to apply a limited resolution strategy to the CNF prior to execution of search. We first identify a general resolution strategy that can be applied to any CNF. As we shall see, for example, this strategy matches the performance of SYNTAX 2 when applied to CNFs encoded according to SYNTAX 1 from Table 1. We also show that by paying attention to where a CNF originates, and by bringing to bear structure that exists in the system being modeled, it is possible to define a more effective *structured* resolution strategy. We demonstrate this on CNF encodings of Bayesian networks, which have been used as inputs to both model counters [18] and knowledge compilers [12]. Using this structured strategy, we achieve significant improvements in the time and space efficiency of inference compared to unprocessed CNFs. Moreover, we are able to perform inference on some models that proved too difficult without applying the resolution strategy.

This paper is organized as follows. In Section 2, we review search with decomposition and demonstrate the importance of syntax. In Section 3, we define semantic independence, and describe a resolution strategy that is meant to bridge the gap between syntactic and semantic independence. Section 4 then reviews CNF encodings of Bayesian networks. Section 5 presents a technique that utilizes structure in a Bayesian network to guide the encoding of the corresponding CNF. In Section 6, we provide experimental results that show the benefits of this strategy. Finally, we conclude with a few remarks in Section 7.

2 The Effect of Syntax on the Syntactic Identification of Components

In this section, we review how search with decomposition works and then demonstrate the effect that syntax can have on the ability of the algorithm to identify components. Consider the problem of counting the models in the CNF at the top of Figure 1. Because all of the clauses in this CNF contain variable A, we

[1] Sequential circuits have been converted into combinational circuits in a standard way, by cutting feedback loops into flip-flops, treating a flip-flop's input as a circuit output and its output as a circuit input.

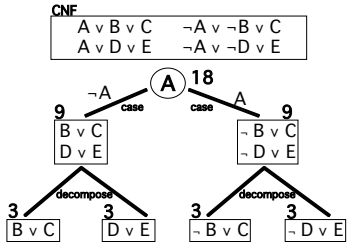

Fig. 1. An example of a search algorithm (with decomposition) that performs model counting.

cannot syntactically decompose the CNF, and so we must split on some variable. Splitting on variable A and performing unit resolution generates the two CNFs at the middle of Figure 1. At this point, we solve each of the two subproblems recursively and independently. We see that each subproblem decomposes into two sets of syntactically independent clauses. The four resulting sets are shown at the bottom of the figure. The CNFs at the bottom represent base cases in the recursion, each having a count of 3, as indicated. From these counts, we can compute counts for the CNFs in the middle, both 9 in this case. And from the middle counts, we compute the count for the CNF at the top of the figure, 18, which is the answer to the original problem. Although we have illustrated the search in a breadth first–fashion, it is normally performed depth–first [6]. 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 [17, 11, 13, 5].

We next present an example which reveals the effect that syntax can have on the identification of components. Consider Figure 2(a) which depicts two fragments of a CNF: fragment α which includes, among other things, an encoding of an AND gate g with output D and inputs A, B, and C, and fragment β which includes clauses that mention variables A, B and C. Suppose further that the clauses for gate g are the only ones that mention variables A, B and C within α. These two fragments are then syntactically dependent as they share common variables, and cannot be solved in isolation. Suppose now that we decide to split on variable A by setting it to false. Under this setting, the output D of the gate must become false, and the inputs B and C are no longer relevant to fragment α. Semantically, fragments α and β are now independent and can be solved in isolation. However, depending on how we encode the gate g, this semantic independence may or may not be revealed syntactically! In particular, Figures 2(b) and 2(c) depict two different encodings of g, which we shall call Syntax 1 and Syntax 2, respectively. Either of these encodings could form the part of fragment α pertaining to gate g. The figures also show the result of simplifying (by performing unit resolution and removing subsumed clauses) these encodings when setting variable A to false. As is clear from this example, variables B and C continue to appear in the clauses of Syntax 1 even though

they are irrelevant. These variables, however, cease to appear in the clauses of SYNTAX 2. Therefore, SYNTAX 2 enables decomposition, but SYNTAX 1 will probably require splitting on variables B and C.

A different situation occurs when we set the output D to true. In this case, all gate inputs must be true. Setting them accordingly is sufficient to sever the dependency between the two fragments. In the case of SYNTAX 1, simplifying is insufficient to discover that the inputs can be set, but in the case of SYNTAX 2, simplifying does indeed tell us the values of the inputs. As a result, SYNTAX 2 once again enables decomposition, but SYNTAX 1 requires more splitting (or a more powerful inference than unit resolution). In fact, the two different encodings of Table 1 are based on the encodings of gates shown in Figure 2, which encode each gate in isolation, in the two ways described. We have seen the significant discrepancy in performance that search with decomposition can have on these two different encodings.

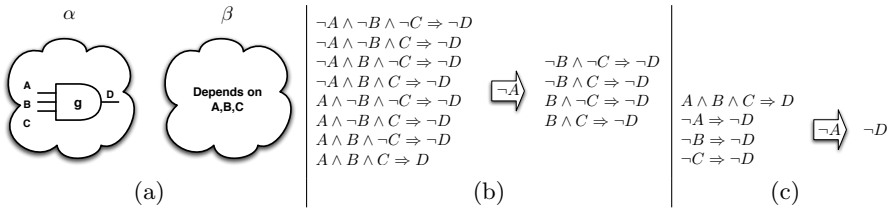

Fig. 2. (a) A depiction of two sets of clauses; (b) the AND gate encoded according to SYNTAX 1; and (c) the AND gate encoded according to SYNTAX 2.

3 Semantic Independence

In this section, we define the notions of syntactic and semantic independence and discuss the encoding of CNFs to reduce the gap between them. Two CNFs are *syntactically independent* if they do not have variables in common. Two CNFs are *semantically independent* if each variable is irrelevant to either CNF (or both). More formally, two CNFs α and β are semantically independent iff for every variable V, $\alpha|V \equiv \alpha|\neg V$ or $\beta|V \equiv \beta|\neg V$, where $\alpha|V$ is the result of setting variable V to true in α, and $\alpha|\neg V$ is the result of setting V to false.

Given a logical theory Δ on which we must perform inference, there are many CNFs that specify Δ, any one of which may be supplied to the search with decomposition to obtain a correct answer. However, to make inference efficient, the goal will be to supply a CNF that makes semantic independence visible syntactically throughout the search. That is, whenever two subsets of the clauses are semantically independent, one should strive to also make them syntactically independent.

Given a CNF for Δ, we now describe a general method that produces another CNF for Δ that may better reveal semantic independence. The idea is to

perform a limited type of resolution on the CNF prior to invoking the search. In particular, the strategy, which we will call Resolution Strategy 1, specifies that whenever there are two clauses of the form $\alpha \vee \beta \vee X$ and $\alpha \vee \neg X$, where α and β are clauses and X is a variable, replace the former clause with $\alpha \vee \beta$. Resolution Strategy 1 makes semantic independence more visible within the CNF, as demonstrated by the following example. Consider again the AND gate with inputs A, B, and C and output D. As we have seen, encoding this gate into CNF according to Syntax 1 results in the following clauses:

$$\begin{array}{cc} \neg A \wedge \neg B \wedge \neg C \Rightarrow \neg D & A \wedge \neg B \wedge \neg C \Rightarrow \neg D \\ \neg A \wedge \neg B \wedge C \Rightarrow \neg D & A \wedge \neg B \wedge C \Rightarrow \neg D \\ \neg A \wedge B \wedge \neg C \Rightarrow \neg D & A \wedge B \wedge \neg C \Rightarrow \neg D \\ \neg A \wedge B \wedge C \Rightarrow \neg D & A \wedge B \wedge C \Rightarrow D \end{array}$$

Applying Resolution Strategy 1 transforms the clauses as follows:

$$A \wedge B \wedge C \Rightarrow D \qquad \neg A \Rightarrow \neg D \qquad \neg B \Rightarrow \neg D \qquad \neg C \Rightarrow \neg D$$

These reduced clauses correspond to Syntax 2's encoding of the AND gate.

For ISCAS89 circuits, applying Resolution Strategy 1 to Syntax 1 is very efficient. The last two columns of Table 1 demonstrate what happens to model count times using Cachet [1]. The most important point is that Resolution Strategy 1 matches Syntax 2's performance, even though Syntax 2 had the advantage of utilizing structure from the source domain (gate types), which was unavailable to Resolution Strategy 1.

It will help at this point to describe two types of structure that can exist in a circuit: local and global. One approach to encoding a circuit is to construct a truth table over all variables in the circuit, and for each term that corresponds to falsehood, generate a clause that outlaws the term. This approach utilizes no structure and is clearly impractical in most cases. Syntax 1 described earlier represents an improvement that makes use of structure that can be inferred from the topology of the circuit. In particular, the topology implies a *factorization* of the global truth table into many smaller truth tables, one for each gate, that allows us to encode each smaller truth table in isolation. We refer to this type of structure as *global structure*. Utilizing global structure makes many problems practical that would not be otherwise. Syntax 2 goes even further, paying attention to gate type during the encoding of a specific gate. We refer to this type of structure as *local structure*. As we have seen, harnessing local structure can uncover additional semantic independence, making a large difference in how efficiently search with decomposition runs. Benefits that arise from exploiting global and local structure have long been realized in the domain of logical circuits, as Syntax 2 is the standard way of encoding such circuits. However, these benefits may also exist in other domains, where they are not always fully exploited. To demonstrate further how both global and local structure can be utilized to reveal semantic independence, we now turn to a specific application where CNFs correspond to encodings of Bayesian networks. Although Resolution Strategy 1 is very efficient when applied to logical circuits, when dealing

with Bayesian networks, more can be gained by paying attention to where the CNF originated.

4 CNF Encodings of Bayesian Networks

The encoding of Bayesian networks into CNFs was proposed in [12], which called for compiling these CNFs into a tractable form, d-DNNF, allowing probabilistic inference to be performed in time linear in the size of resulting compilation (through weighted model counting on the compiled form [10]). More recently, [18] proposed a similar approach, but using a different CNF encoding and applying a model counter directly on the CNF, instead of compiling the CNF first. Both approaches, however, use search with decomposition as the core algorithm, yet the compilation approach keeps a trace of the search [15].

We will now review the CNF encoding of a Bayesian network as given in [12] as the specific encoding will play a role in the remainder of the paper. A Bayesian network is a directed acyclic graph (DAG) and a set of tables called conditional probability tables (CPTs), one table for each node in the DAG. The CPTs are analogous to the truth tables of gates in a circuit. Two major differences are that variables in a CPT can be multi–valued and instead of mapping each row to truth or falsehood, a CPT maps each row to a real–number called a *parameter*.[2] Figure 3(a) depicts an example CPT, where variable A and B each have two values and variable C has three values. When encoding gates of a circuit, global structure allowed Syntax 1 to encode each gate separately. In a similar way, each CPT in a network can be encoded in isolation. When encoding a truth table for a particular logic gate, local structure allowed Syntax 2 to tailor its encoding to the particular gate type. It can be more difficult to utilize local structure in a Bayesian network. Tables are not normally associated with a type, so local structure must be inferred from parameter values.

The encoding that will serve as our starting point, which we will refer to as Baseline Encoding, captures a large amount of local structure and was consequently shown in [8] to vastly improve compilation performance on many benchmark networks. This encoding begins by looking at the network variables. For each value x of each network variable X, we create in the CNF an *indicator variable* λ_x. For example, for network variable C with values c_1, c_2, and c_3, the encoding would generate CNF variables λ_{c_1}, λ_{c_2}, and λ_{c_3}. Next, for each network variable, we generate *indicator clauses*, which assert that in each model, exactly one of the corresponding indicator variables is true. For variable C, these clauses are as follows: $\lambda_{c_1} \vee \lambda_{c_2} \vee \lambda_{c_3}$, $\neg\lambda_{c_1} \vee \neg\lambda_{c_2}$, $\neg\lambda_{c_1} \vee \neg\lambda_{c_2}$, and $\neg\lambda_{c_1} \vee \neg\lambda_{c_1}$. The encoding then looks at each CPT in isolation. For each non–zero parameter value that is unique within its CPT, the encoding generates a CNF *parameter variable*. For example, the parameters in rows 7–9 in the CPT in Figure 3(a), all equal to 0.333, might generate parameter variable θ_4. Finally, for each row in the CPT, the encoding generates a *parameter clause*. A parameter clause asserts that the

[2] There are other restrictions on the CPTs of a Bayesian network that are not important to the current discussion.

A	B	C	$\Pr(c\|a,b)$	
a_1	b_1	c_1	0.7	(θ_1)
a_1	b_1	c_2	0.0	(false)
a_1	b_1	c_3	0.3	(θ_2)
a_1	b_2	c_1	0.4	(θ_3)
a_1	b_2	c_2	0.3	(θ_2)
a_1	b_2	c_3	0.3	(θ_2)
a_2	b_1	c_1	0.333	(θ_4)
a_2	b_1	c_2	0.333	(θ_4)
a_2	b_1	c_3	0.333	(θ_4)
a_2	b_2	c_1	0.2	(θ_5)
a_2	b_2	c_2	0.3	(θ_2)
a_2	b_2	c_3	0.5	(θ_6)

(a)

$\lambda_{a_1} \wedge \lambda_{b_1} \wedge \lambda_{c_1} \rightarrow \theta_1$
$\neg\lambda_{a_1} \vee \neg\lambda_{b_1} \vee \neg\lambda_{c_2}$
$\lambda_{a_1} \wedge \lambda_{b_1} \wedge \lambda_{c_3} \rightarrow \theta_2$
$\lambda_{a_1} \wedge \lambda_{b_2} \wedge \lambda_{c_1} \rightarrow \theta_3$
$\lambda_{a_1} \wedge \lambda_{b_2} \wedge \lambda_{c_2} \rightarrow \theta_2$
$\lambda_{a_1} \wedge \lambda_{b_2} \wedge \lambda_{c_3} \rightarrow \theta_2$
$\lambda_{a_2} \wedge \lambda_{b_1} \wedge \lambda_{c_1} \rightarrow \theta_4$
$\lambda_{a_2} \wedge \lambda_{b_1} \wedge \lambda_{c_2} \rightarrow \theta_4$
$\lambda_{a_2} \wedge \lambda_{b_1} \wedge \lambda_{c_3} \rightarrow \theta_4$
$\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_1} \rightarrow \theta_5$
$\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_2} \rightarrow \theta_2$
$\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_3} \rightarrow \theta_6$

(b)

$\lambda_{a_1} \wedge \lambda_{b_1} \wedge \lambda_{c_1} \rightarrow \theta_1$
$\neg\lambda_{a_1} \vee \neg\lambda_{b_1} \vee \neg\lambda_{c_2}$
$\lambda_{a_1} \wedge \lambda_{c_3} \rightarrow \theta_2$
$\lambda_{b_2} \wedge \lambda_{c_2} \rightarrow \theta_2$
$\lambda_{a_1} \wedge \lambda_{b_2} \wedge \lambda_{c_1} \rightarrow \theta_3$
$\lambda_{a_2} \wedge \lambda_{b_1} \rightarrow \theta_4$
$\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_1} \rightarrow \theta_5$
$\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_3} \rightarrow \theta_6$

(c)

Fig. 3. (a) A CPT over three variables, (b) Clauses generated by the encoding from [8] for the CPT, and (c) an equivalent encoding.

conjunction of the corresponding indicators implies θ, where θ is the parameter variable for the row, or falsehood if the row's parameter is zero. For example, the seventh row in Figure 3(a) generates the clause $\lambda_{a_2} \wedge \lambda_{b_1} \wedge \lambda_{c_1} \Rightarrow \theta_4$, and the second row generates the clause $\neg\lambda_{a_1} \vee \neg\lambda_{b_1} \vee \neg\lambda_{c_2}$. The encoding in [8] uses a few additional optimizations, which are unimportant for the current discussion. The complete set of clauses for the rows of the CPT in Figure 3(a) is shown in Figure 3(b).

5 Encoding with Local Structure

Baseline Encoding capitalizes on *determinism* (zero probabilities) and equal parameters in the network by omitting the generation of parameter variables for certain parameters. The effect on inference can be dramatic, as was shown in [8]. However, Baseline Encoding does not go as far as possible to capitalize on local structure. In this section, we introduce a new encoding method that retains the advantages of Baseline Encoding while further harnessing local structure to uncover semantic independence and improve component analysis.

Consider again Figure 3(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 of variable C. This phenomenon is similar to context–specific–independence (CSI) [7] and can be very powerful. CSI is normally taken to mean that given values of certain parents (A or B in this case), some other parent becomes irrelevant to the probability of the child (C). The phenomenon described here is a more powerful generalization as it also captures cases where (1) setting one or more parents causes the distribution on the child to become

uniform or (2) when setting the child to a certain value makes a parent irrelevant. This type of structure allows the clauses in Figure 3(b) to be simplified to the clauses in Figure 3(c), which will tend to have fewer occurrences of irrelevant variables as we set variables in search process.

Before defining a general procedure for simplifying the clauses of a given CPT, 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 Γ be a disjunction of terms over $\mathbf{X}$. An *implicant* γ of Γ is a term over $\mathbf{X}' \subseteq \mathbf{X}$ that implies Γ. A *prime implicant* γ of Γ 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 Γ.

Algorithm 1 EncodeCPT(ϕ: CPT) Generates a set of clauses for ϕ.

```
Partition the rows of φ into groups so that all rows with the same parameter are in
the same group
for each encoding group Γ do
  M ← terms of Γ
  θ ← consequent of Γ
  P ← the prime implicants of M
  for p in P do
    I ← encoding of p
    if θ = 0 then
      assert clause ¬I
    else
      assert clause I ⇒ θ
    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 Baseline Encoding and by generating clauses for each CPT according to Algorithm 1. This algorithm encodes a CPT ϕ 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 Γ, we find the prime implicants of Γ's terms, and for each prime implicant p,

we assert a clause $I \Rightarrow \theta$, where I is conjunction of indicators corresponding to p, and θ is the consequent of the encoding group. If the parameter θ equals 0, we simply generate the clause $\neg I$. Figure 4 demonstrates this algorithm for the CPT in Figure 3(a).

The algorithm we use to find prime implicants is an extension of the venerable Quine-McCluskey (QM) algorithm (e.g., [14]). 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., [16]). The definition given here was found effective for the purpose at hand.

Encoding Group	Param Value	Terms	Consequent	Prime Implicants	Encoding
Γ_1	.7	$a_1b_1c_1$	θ_1	$a_1b_1c_1$	$\lambda_{a_1} \wedge \lambda_{b_1} \wedge \lambda_{c_1} \Rightarrow \theta_1$
Γ_2	0	$a_1b_1c_2$	false	$a_1b_1c_2$	$\neg\lambda_{a_1} \vee \neg\lambda_{b_1} \vee \neg\lambda_{c_2}$
Γ_3	.3	$a_1b_1c_3, a_1b_2c_2, a_1b_2c_3, a_2b_2c_2$	θ_2	a_1c_3, b_2c_2	$\lambda_{a_1} \wedge \lambda_{c_3} \Rightarrow \theta_2$, $\lambda_{b_2} \wedge \lambda_{c_2} \Rightarrow \theta_2$
Γ_4	.4	$a_1b_2c_1$	θ_3	$a_1b_2c_1$	$\lambda_{a_1} \wedge \lambda_{b_2} \wedge \lambda_{c_1} \Rightarrow \theta_3$
Γ_5	.333	$a_2b_1c_1, a_2b_1c_2, a_2b_1c_3$	θ_4	a_2b_1	$\lambda_{a_2} \wedge \lambda_{b_1} \wedge \Rightarrow \theta_4$
Γ_6	.2	$a_2b_2c_1$	θ_5	$a_2b_2c_1$	$\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_1} \Rightarrow \theta_5$
Γ_7	.5	$a_2b_2c_3$	θ_6	$a_2b_2c_3$	$\lambda_{a_2} \wedge \lambda_{b_2} \wedge \lambda_{c_3} \Rightarrow \theta_6$

Fig. 4. Encoding a CPT using prime implicants.

The new encoding method described defines a structured resolution strategy which we will refer to as Resolution Strategy 2. 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 close this section with a few observations. First, even though computing prime implicants can be expensive in general, Resolution Strategy 2 adds little overhead to Baseline Encoding. This efficiency stems from the small number of variables that are involved in the computation (those appearing in a CPT). This is to be contrasted with our first resolution strategy, which is applied to variables in the whole CNF. Second, there is a strong similarity between the two strategies. In particular, both are capable of removing occurrences of literals, transforming a set of terms into a more minimal set, and in this way revealing semantic independence. Third, the main idea presented might be applied more generally to other domains where a CNF is encoded from a set of functions over finitely valued variables. As we have seen, two examples are truth tables and Bayesian networks. Other examples are Markov networks and influence diagrams. Finally, CNFs created using Resolution Strategy 2 will be smaller than those created using Baseline Encoing. A natural question is how

much of any gains achieved arise from smaller CNFs as opposed to increased decomposability? It is not clear how one would conduct an analysis to answer this question, but the magnitude of improvements obtained clearly demonstrate that reduction in size could not be solely responsible.

6 Experimental Results

In this section, we examine a number of Bayesian networks. For each, we generate a CNF according to Baseline Encoding and another using Resolution Strategy 2. We compile the CNFs into d-DNNF using the c2d compiler [3, 13] and compare performance. Table 2 shows five sets of networks. The first set consists of ISCAS89 circuits converted to Bayesian networks by placing uniform probability distributions on inputs and encoding other gates with deterministic CPTs (all parameters 0 or 1). The blockmap (bm) networks were generated from relational probabilistic models and were first used in [9] to demonstrate the effectiveness of the compilation approach to networks of this type. The OR and grid (gr) networks were used in [18] to also show the effectiveness of weighted model counting for probabilistic inference, this time using search rather than compilation. From the large number of OR and grid networks, which are divided into sets of ten, we selected sets that provided a challenge for c2d, while still possible to compile within 2GB of memory using Resolution Strategy 2. Finally, the last set consists of benchmark networks from various sources that have long been used to compare probabilistic inference algorithms. Experiments ran on a 1.6Ghz Pentium M with 2GB of memory. The implementation of the encoding and compiling algorithms have been packaged in the publicly available tools Ace 1.1 [4] (Resolution Strategy 2 for encoding Bayesian networks) and c2d 2.2 [3] (Resolution Strategy 1 for general CNFs).

For each network, Table 2 first lists the maximum cluster size as computed by a minfill heuristic. This measure is important because inference algorithms that do not use local structure run in time that is exponential in this number. We next list encoding times for the two encoding algorithms. The main point is that the resolution taking place in Resolution Strategy 2 is not adding significant time to the encoding. Compile times then reveal the extent to which Resolution Strategy 2 helps. In particular, we see that, except for one case, 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 Baseline Encoding but compiled successfully using Resolution Strategy 2. The last three columns show the improvement to the size (number of edges) of the resulting compilations. This size is important to demonstrate space requirements and also because online inference, which may be repeated a great many times for a given application, runs in time that is linear in this size. Here, we see that on networks where Baseline Encoding was successful, sizes were sometimes comparable and otherwise significantly reduced.

Network	Max. Clst. Size	Baseline Enc. Time	RS 2 Enc. Time	**Baseline Comp. Time**	**RS 2 Comp. Time**	Improvement	**Baseline Comp. Size**	**RS 2 Comp. Size**	Improvement
s1238	61.0	0.91	0.86	11.32	1.83	6.19	853,987	263,786	3.24
s713	19.0	0.80	0.79	1.40	0.35	4.00	67,428	37,495	1.80
s526n	18.0	0.73	0.73	0.23	0.12	1.92	10,088	10,355	0.97
s1196	54.0	0.86	0.83	6.16	1.33	4.63	685,254	189,381	3.62
s526	18.0	0.69	0.68	0.22	0.14	1.57	13,352	14,143	0.94
s953	70.0	0.79	0.80	13.88	2.19	6.34	691,220	205,043	3.37
s641	19.0	0.84	0.79	2.54	0.38	6.68	78,071	36,555	2.14
s1488	46.0	0.89	0.88	1.65	0.56	2.95	333,629	125,739	2.65
s1494	48.0	0.88	0.90	1.82	0.44	4.14	419,274	85,469	4.91
s832	27.0	0.75	0.76	1.38	0.24	5.75	62,756	32,715	1.92
s838.1	13.0	0.79	0.80	0.43	0.20	2.15	49,856	30,899	1.61
s1423	24.0	0.91	0.91	56.62	14.54	3.89	3,010,821	994,518	3.03
bm-05-03	19.0	1.04	1.10	0.29	0.20	1.45	19,190	10,957	1.75
bm-10-03	51.0	2.90	3.03	19.57	4.97	3.94	938,371	275,089	3.41
bm-15-03	62.0	7.76	7.39	254.96	44.07	5.79	7,351,823	1,460,842	5.03
bm-20-03	90.0	17.96	17.40	1,505.24	388.65	3.87	37,916,087	6,195,000	6.12
bm-22-03	107.0	26.26	25.62	4,869.64	748.13	6.51	72,169,022	14,405,730	5.01
or-60-20-1	24.0	0.69	0.77	338.48	54.47	6.21	6,968,339	7,777,867	0.90
or-60-20-3	25.0	1.04	0.69	1.40	0.77	1.82	104,275	119,779	0.87
or-60-20-5	27.0	0.74	0.70	728.36	118.17	6.16	17,358,747	14,986,497	1.16
or-60-20-7	26.0	1.08	0.71	250.72	97.13	2.58	11,296,613	12,510,488	0.90
or-60-20-9	25.0	0.73	0.70	19.58	7.17	2.73	1,011,193	1,060,217	0.95
gr-50-16-1	24.0	0.76	0.75	137.25	43.95	3.12	14,692,963	5,739,854	2.56
gr-50-16-2	25.0	0.86	4.52	-	292.42	-	-	35,473,955	-
gr-50-16-3	24.0	0.92	0.74	65.03	40.45	1.61	7,755,318	5,280,027	1.47
gr-50-16-4	24.0	1.21	0.80	407.60	46.83	8.70	35,950,912	6,128,859	5.87
gr-50-16-5	25.0	0.88	0.82	-	26.70	-	-	3,431,139	-
gr-50-16-6	25.0	0.85	0.79	44.40	22.99	1.93	4,598,373	3,159,007	1.46
gr-50-16-7	24.0	0.85	0.84	51.68	2.99	17.28	6,413,897	421,060	15.23
gr-50-16-8	24.0	0.84	0.81	86.19	32.29	2.67	10,341,755	4,280,261	2.42
gr-50-16-9	24.0	0.84	0.94	-	60.55	-	-	7,360,872	-
gr-50-16-10	24.0	0.84	0.83	133.70	287.08	0.47	15,144,602	33,561,672	0.45
gr-50-18-1	27.0	1.02	0.87	411.45	48.36	8.51	39,272,847	6,451,916	6.09
gr-50-18-2	28.0	0.94	0.92	-	172.13	-	-	19,037,468	-
gr-50-18-3	27.0	0.91	0.86	362.90	29.18	12.44	32,120,267	2,507,215	12.81
gr-50-18-4	28.0	1.62	0.98	-	139.81	-	-	15,933,651	-
gr-50-18-5	27.0	1.26	1.07	-	158.13	-	-	18,291,116	-
gr-50-18-6	28.0	1.05	0.86	403.96	52.55	7.69	37,411,619	7,111,893	5.26
gr-50-18-7	27.0	0.98	0.98	-	79.97	-	-	9,439,318	-
gr-50-18-8	28.0	0.93	0.89	-	42.17	-	-	5,036,670	-
gr-50-18-9	27.0	0.96	0.87	-	68.51	-	-	7,890,645	-
gr-50-18-10	28.0	1.00	1.00	-	188.66	-	-	22,387,841	-
water	20.8	1.04	0.95	2.81	1.73	1.62	101,009	103,631	0.97
pathfinder	15.0	2.97	1.86	12.45	2.86	4.35	36,024	33,614	1.07
diabetes	18.2	10.76	7.77	6,281.23	3,391.18	1.85	15,426,793	15,751,044	0.98
mildew	20.7	13.37	8.16	6,245.45	1,869.92	3.34	1,693,750	1,696,139	1.00
barley	23.4	4.36	6.75	-	14,722.19	-	-	37,321,497	-

Table 2. Results for compiling a number of networks using Baseline Encoding and the Resolution Strategy 2 encoding. All times are in seconds..

Before closing this section, we place these results into a broader perspective. The first critical point is that on many of these networks, inference approaches that do not utilize local structure would simply fail, because of large cluster sizes. The second point is that the gains that Resolution Strategy 2 achieves are particularly noteworthy, since they are being compared to a state–of–the–art technique for utilizing local structure [8]. The approach described in [18] harnesses local structure within the Bayesian network, applies the Cachet model counter to a different CNF encoding, and has been shown to be successful on some of the networks considered here. Table 3 repeats some of the results reported in [18] with regards to networks in Table 2. In particular, for each of several sets of networks, search times running on a dual 2.8GHz processor with 4GB of memory are shown. Each time represents the median over ten networks. Also shown in the table are median compile times we achieved for the two encodings considered in this paper. As can be seen from the table, the times are comparable for the OR networks, but both Baseline Encoding and Resolution Strategy 2 allow compilation to run more efficiently on grid networks (even though compilation would normally require much more overhead than search). We note here that the grid networks in Table 3 were chosen from a large number of such networks because they represent some of the hardest of the group (they contain the least amount of determinism and any larger grids having the same degree of determinism cause Cachet to fail).

Network Set	Cachet Search Time (s)	Baseline Encoding Compile Time (s)	Resolution Strategy 2 Compile Time (s)
grid-50-16	890	135	42
grid-50-18	13,111	592	74
or-60-5	1.7	3.9	1.9
or-60-10	3.9	24.9	8.7
or-60-20	54	294.6	64.8

Table 3. Median times for Cachet search and for c2d compilation using Baseline Encoding and Resolution Strategy 2.

7 Conclusion

We observe in this paper that the particular syntax of a CNF can be critical for the performance of search–with–decomposition algorithms, as it can lead to a gap between semantic and syntactic independence that can hinder the identification of semantically independent components. We provide two resolutions strategies, one general and one more structured, for pre–processing a CNF with the aim of reducing the gap between syntactic and semantic independence. We apply our proposed techniques to general CNF encodings, and to more specific ones corresponding to Bayesian networks. Experimental results show large improvements

when applying state of the art search–with–decomposition algorithms, including the Cachet model counter and the c2d compiler, allowing us to solve some problems that have previously exhausted available resources.

References

1. The cachet model counter, http://www.cs.washington.edu/homes/kautz/Cachet.
2. ISCAS89 Benchmark Circuits, http://www.cbl.ncsu.edu/www/CBL_Docs/iscas89.html.
3. The c2d compiler, http://reasoning.cs.ucla.edu/c2d.
4. The Ace compiler, http://reasoning.cs.ucla.edu/ace.
5. Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Dpll with caching: A new algorithm for #SAT and Bayesian inference. *Electronic Colloquium on Computational Complexity (ECCC)*, 10(003), 2003.
6. R. Bayardo and J. Pehoushek. Counting models using connected components. In *AAAI*, pages 157–162, 2000.
7. Craig Boutilier, Nir Friedman, Moisés Goldszmidt, and Daphne Koller. Context–specific independence in Bayesian networks. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 115–123, 1996.
8. Mark Chavira and Adnan Darwiche. Compiling Bayesian networks with local structure. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1306–1312, 2005.
9. Mark Chavira, Adnan Darwiche, and Manfred Jaeger. Compiling relational Bayesian networks for exact inference. In *Proceedings of the Second European Workshop on Probabilistic Graphical Models (PGM)*, pages 49–56, 2004.
10. Adnan 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):11–34, 2001.
11. Adnan Darwiche. A compiler for deterministic, decomposable negation normal form. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, pages 627–634, Menlo Park, California, 2002. AAAI Press.
12. Adnan Darwiche. A logical approach to factoring belief networks. In *Proceedings of KR*, pages 409–420, 2002.
13. Adnan Darwiche. New advances in compiling CNF to decomposable negational normal form. In *Proceedings of European Conference on Artificial Intelligence*, pages 328–332, 2004.
14. John P. Hayes. *Introduction to Digital Logic Design*. Addison Wesley, 1993.
15. Jinbo Huang and Adnan Darwiche. Dpll with a trace: From sat to knowledge compilation. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 156–162, 2005.
16. M. M. Mirsalehi and T. K. Gaylord. Logical minimization of multilevel coded functions. *Applied Optics*, 25:3078–3088, September 1986.
17. Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *SAT*, 2004.
18. Tian Sang, Paul Beame, and Henry Kautz. Solving Bayesian networks by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, volume 1, pages 475–482. AAAI Press, 2005.