An Improved CNF Encoding Scheme for Probabilistic Inference

Anicet Bart[1] and **Frédéric Koriche** and **Jean-Marie Lagniez** and **Pierre Marquis**[2]

Abstract. We present and evaluate a new CNF encoding scheme for reducing probabilistic inference from a graphical model to weighted model counting. This new encoding scheme elaborates on the CNF encoding scheme **ENC4** introduced by Chavira and Darwiche, and improves it by taking advantage of log encodings of the elementary variable/value assignments and of the implicit encoding of the most frequent probability value per conditional probability table. From the theory side, we show that our encoding scheme is faithful, and that for each input network, the CNF formula it leads to contains less variables and less clauses than the CNF formula obtained using **ENC4**. From the practical side, we show that the C2D compiler empowered by our encoding scheme performs in many cases significantly better than when **ENC4** is used, or when the state-of-the-art ACE compiler is considered instead.

1 INTRODUCTION

A number of approaches have been developed during the past fifteen years for improving probabilistic inference, by taking advantage of the local structure (contextual independence and determinism) which may occur in the input graphical model (a weighted constraint network, representing typically a Bayesian network or a Markov network) [4, 34, 22, 3, 28, 18, 9, 16, 23, 36]. Among them are approaches which consist in associating with the input graphical model a weighted propositional formula via a polynomial-time translation [14, 33, 7, 37, 10, 11]. Once the translation has been applied, the problem of computing the probability (or more generally, the weight) of a given piece of evidence (assignments of values to some variables) mainly amounts to solving an instance of the (weighted) model counting problem.

While this problem is still #P-complete, it has received much attention in the past few years, both in theory and in practice; thus, many algorithms have been designed for solving the model counting problem #SAT, either exactly or approximately (see e.g., [2, 19, 20, 31, 5]); search-based model counters (like Cachet [32] and sharpSAT [35]) and preprocessing techniques for #SAT [21] have been developed and evaluated. Propositional languages supporting the (weighted) model counting query in polynomial time have been defined and investigated, paving the way to compilation-based model counters (i.e., when the propositional encoding of the model is first turned into a compiled representation). The most prominent ones are the language Decision-DNNF of decision-based decomposable negation normal form formulas [12] and the language SDD consisting of sentential decision diagrams – a subset of d-DNNF – [17]. Compilers targeting those languages have been developed (many of them are available on line); let us mention the top-down compilers C2D and Dsharp targeting the Decision-DNNF language [12, 15, 25], and the top-down compiler and the bottom-up compiler targeting the SDD language [27, 17].

In the following, we present and evaluate a new CNF encoding scheme for weighted constraint networks. This new encoding scheme elaborates on the CNF encoding scheme **ENC4** introduced in [10], and improves it by taking advantage of two supplementary "ideas": the log encodings of the elementary variable/value assignments and of the implicit encoding of the most frequent probability value per table. While log encodings of variables is a simple idea which has been considered before for credal networks (and defined as "binarization") [1], as far as we know, it had never been tested before for encoding weighted constraint networks into CNF. Furthermore, using an implicit encoding of the most frequent probability value per table seems brand new in this context.

Interestingly, unlike the formulae obtained using **ENC4**, the CNF formulae generated by our encoding scheme can be compiled into Decision-DNNF representations which do not need to be minimized (thanks to a specific handling of the weights given to the negative parameter literals). As such, they can also be exploited directly by any weighted model counter. From the theory side, we show that our encoding scheme is faithful, and that for each weighted constraint network, the CNF formula it leads to contains less variables and less clauses than the CNF formula obtained using **ENC4**. From the practical side, we performed a large-scale evaluation by compiling 1452 weighted constraint networks from 6 data sets. This evaluation shows our encoding scheme valuable in practice. More in detail, we have compared the compilation times and the sizes of the compiled representations produced by C2D (reasoning.cs.ucla.edu/c2d/) when using our encoding scheme, with the corresponding measures when **ENC4** is considered instead. Our encoding scheme appeared as a better performer than **ENC4** since it led most of the time to improved compilation times and improved compilation sizes. We have also done a differential evaluation which has revealed that each of the two "ideas" considered in our encoding is computationally fruitful. We finally compared the performance of C2D empowered by our encoding scheme with those of ACE, a state-of-the-art compiler for Bayesian networks based on **ENC4**, see http://reasoning.cs.ucla.edu/ace. Again, our empirical investigation also showed that C2D equipped with our encoding scheme challenges ACE in many cases. More precisely, it was able to compile more instances given the time and memory resources allocated in our experiments, and it led often to compiled representations significantly smaller than the ones computed using ACE.

[1] LINA-CNRS-INRIA and Ecole des Mines de Nantes, F-44000 Nantes, France, email: anicet.bart@univ-nantes.fr
[2] CRIL, Univ. Artois and CNRS, F-62300 Lens, France, email: {koriche, lagniez, marquis}@cril.univ-artois.fr

2 FORMAL PRELIMINARIES

A (finite-domain) *weighted constraint network* (alias WCN) is a triple $\mathcal{WCN} = (\mathcal{X}, \mathcal{D}, \mathcal{R})$ where $\mathcal{X} = \{X_1, \ldots, X_n\}$ is a finite set of variables; each variable X from $\mathcal{X}$ is associated with a finite set, its domain D_X, and $\mathcal{D}$ is the set of all domains of the variables from $\mathcal{X}$; $\mathcal{R} = \{R_1, \ldots, R_m\}$ is a finite set of (possibly partial) functions over the reals; with each R in $\mathcal{R}$ is associated a subset $scope(R)$ of $\mathcal{X}$, called the scope of R and gathering the variables involved in R; each R is a mapping from its domain $Dom(R)$, a subset of the Cartesian product D_R of the domains of the variables of $scope(R)$, to $\mathbb{R}$; the cardinality of $scope(R)$ is the arity of R. In the following, each function R is supposed to be represented in extension (i.e., as a table associating weights with assignments).

An *assignment* $\mathtt{a}$ of $\mathcal{WCN}$ over a subset $\mathcal{S}$ of $\mathcal{X}$ is a set $\mathtt{a} = \{(X, d) \mid X \in \mathcal{S}, d \in D_X\}$ of elementary asignments (X, d), where for each $X \in \mathcal{S}$ there exists a unique pair of the form (X, d) in $\mathtt{a}$. If $\mathtt{a}$ is an assignment of $\mathcal{WCN}$ over $\mathcal{S}$ and $\mathcal{T} \subseteq \mathcal{S}$, then the restriction $\mathtt{a}[\mathcal{T}]$ of $\mathtt{a}$ over $\mathcal{T}$ is given by $\mathtt{a}[\mathcal{T}] = \{(X, d) \in \mathtt{a} \mid X \in \mathcal{T}\}$. Given two subsets $\mathcal{S}$ and $\mathcal{T}$ of $\mathcal{X}$ such that $\mathcal{T} \subseteq \mathcal{S}$, an assignment $\mathtt{a}_1$ of $\mathcal{WCN}$ over $\mathcal{S}$ is said to *extend* an assignment $\mathtt{a}_2$ of $\mathcal{WCN}$ over $\mathcal{T}$ when $\mathtt{a}_1[\mathcal{T}] = \mathtt{a}_2$. A *full assignment* of $\mathcal{WCN}$ is an assignment of $\mathcal{WCN}$ over $\mathcal{X}$.

A full assignment $\mathtt{s}$ of $\mathcal{WCN}$ is a *solution* of $\mathcal{WCN}$ if and only if for every $R \in \mathcal{R}$, we have $\mathtt{s}[scope(R)] \in Dom(R)$. The *weight* of a full assignment $\mathtt{s}$ of $\mathcal{WCN}$ is $w_{\mathcal{WCN}}(\mathtt{s}) = 0$ when $\mathtt{s}$ is not a solution of $\mathcal{WCN}$; otherwise, $w_{\mathcal{WCN}}(\mathtt{s}) = \Pi_{R \in \mathcal{R}} R(\mathtt{s}[scope(R)])$. Finally, the *weight* $w_{\mathcal{WCN}}(\mathtt{a})$ of an assignment $\mathtt{a}$ of $\mathcal{WCN}$ over $\mathcal{S}$ is the sum over all full assignments $\mathtt{s}$ extending $\mathtt{a}$ of the values $w_{\mathcal{WCN}}(\mathtt{s})$.

Example 1 *Let us consider as a running example the following "toy"* $\mathcal{WCN} = (\mathcal{X} = \{X_1, X_2\}, \mathcal{D} = \{D_{X_1}, D_{X_2}\}, \mathcal{R} = \{R\})$*, where* $D_{X_1} = \{0, 1, 2\}$*,* $D_{X_2} = \{0, 1\}$*, and* R *such that* $scope(R) = \{X_1, X_2\}$ *is given by Table 1.*

X_1	X_2	R
0	0	0
0	1	$8/30$
1	0	$1/10$
1	1	$1/10$
2	0	$8/30$
2	1	$8/30$

Table 1: A tabular representation of R.

$\mathtt{a} = \{(X_2, 1)\}$ *is an assignment of* $\mathcal{WCN}$ *over* $\mathcal{S} = \{X_2\}$*. We have* $w_{\mathcal{WCN}}(\mathtt{a}) = 8/30 + 1/10 + 8/30 = 19/30$.

3 ON CNF ENCODING SCHEMES

Our objective is to be able to compute the *weight* of any assignment $\mathtt{a}$ of a given $\mathcal{WCN}$. Typically, the WCN under consideration will be derived without any heavy computational effort (i.e., in linear time) from a given random Markov field or a Bayesian network, and the assignment $\mathtt{a}$ under consideration will represent some available piece of evidence. In such a case, $w(\mathtt{a})$ simply is the probability of this piece of evidence.

In order to achieve this goal, an approach consists in translating first the input $\mathcal{WCN} = (\mathcal{X}, \mathcal{D}, \mathcal{R})$ into a *weighted propositional formula* $\mathcal{WPROP} = (\Sigma, w, w_0)$. In such a triple, Σ is a propositional representation built up from a finite set of propositional variables PS, w is a weight distribution over the literals over PS, i.e., a mapping from $L_{PS} = \{x, \neg x \mid x \in PS\}$ to $\mathbb{R}$, and w_0 is a real number (a scaling factor). The *weight* of an interpretation ω over PS given $\mathcal{WPROP}$ is defined as $w_{\mathcal{WPROP}}(\omega) = w_0 \times \Pi_{x \in L_{PS} | \omega(x)=1} w(x) \times \Pi_{\neg x \in L_{PS} | \omega(x)=0} w(\neg x)$ if ω is a model of Σ, and $w_{\mathcal{WPROP}}(\omega) = 0$ in the remaining case. Furthermore, the *weight* of any consistent term γ over PS given $\mathcal{WPROP}$ is given by $w_{\mathcal{WPROP}}(\gamma) = \Sigma_{\omega \models \gamma} w_{\mathcal{WPROP}}(\omega)$. Given a CNF formula Σ, we denote by $\#var(\Sigma)$ the number of variables occurring in Σ, and by $\#cl(\Sigma)$, the number of clauses in Σ. Finally, a canonical term over a subset V of PS is a consistent term where each variable of V occurs.

Computing $w(\gamma)$ from a consistent term γ and a $\mathcal{WPROP} = (\Sigma, w, w_0)$ is a computationally hard problem in general (it is #P-complete). Weighted model counters (such as Cachet [32]) can be used in order to perform such a computation when Σ is a CNF formula. Reductions from the weighted model counting problem to the (unweighted) model counting problem, as the one reported in [6], can also be exploited, rendering possible the use of (unweighted) model counter, like sharpSAT [35]. Interestingly, when Σ has been compiled first into a Decision-DNNF representation (and more generally into a d-DNNF representation[3]), the computation of $w(\gamma)$ can be done in time linear in the size of the input, i.e., the size of γ, plus the size of the explicit representation of the weight distribution w over L_{PS}, the size of the representation of w_0, and the size of the d-DNNF representation of Σ. Stated otherwise, the problem of computing $w(\gamma)$ from a consistent term γ and a $\mathcal{WPROP} = (\Sigma, w, w_0)$ where Σ is a d-DNNF representation can be solved efficiently.

Whatever the targeted model counter (direct or compilation-based), the approach requires a notion of translation function (the formal counterpart of an encoding scheme):

Definition 1 (translation function) *A mapping* τ *associating any* $\mathcal{WCN} = (\mathcal{X}, \mathcal{D}, \mathcal{R})$ *with a weighted propositional formula* $\tau(\mathcal{WCN}) = (\Sigma, w, w_0)$ *and any assignment* $\mathtt{a}$ *of* $\mathcal{WCN}$ *over a subset* $\mathcal{S}$ *of* $\mathcal{X}$ *with a term* $\tau(\mathtt{a})$ *over the set of propositional variables* PS *on which* Σ *is built, is a* translation function*.*

Valuable translation functions are those for which the encoding scheme is correct. We say that they are faithful:

Definition 2 (faithful translation) *A translation function* τ *is* faithful *when it is such that for any* $\mathcal{WCN} = (\mathcal{X}, \mathcal{D}, \mathcal{R})$ *and any assignment* $\mathtt{a}$ *of* $\mathcal{WCN}$ *over a subset* $\mathcal{S}$ *of* $\mathcal{X}$*,* $w_{\mathcal{WCN}}(\mathtt{a}) = w_{\tau(\mathcal{WCN})}(\tau(\mathtt{a}))$.

Some faithful translation functions have already been identified in the literature, see [13, 33, 8, 10, 11]. Typically, the set PS of propositional variables used in the translation is partitioned into two subsets: a set of indicator variables λ_i used to encode the assignments, and a set of parameter variables θ_j used to encode the weights. Formally let us denote by Λ_X the set of indicator variables used to encode assignments of variable $X \in \mathcal{X}$ and Θ_R be the set of parameter variables introduced in the encoding of $R \in \mathcal{R}$. Every literal l over all those variables has weight 1 (i.e., $w_1(l) = w_4(l) = 1$), except for the (positive) literals θ_j. Translations functions are typically *modular ones*, where "modular" means that the representation Σ to be generated is the conjunction of the representations $\tau(X)$ corresponding to the encoding of the domain D_X of each $X \in \mathcal{X}$, with the representations $\tau(R)$ corresponding to each mapping R in $\mathcal{R}$:

$$\Sigma = \bigwedge_{X \in \mathcal{X}} \tau(X) \wedge \bigwedge_{R \in \mathcal{R}} \tau(R).$$

[3] But existing d-DNNF compilers actually target the Decision-DNNF language [26].

As a matter of example, let us consider the translation functions τ_1 and τ_4 associated respectively with the encoding schemes **ENC1** [13] and **ENC4** reported in [8]. In **ENC1** and **ENC4**, direct encoding is used for the representation of elementary assignments (X, d). This means that every (X, d) is associated by τ_1 (and similarly by τ_4) in a bijective way with an indicator variable $\tau_1((X, d)) = \tau_4((X, d))$, and every assignment $\mathtt{a}$ is associated with the term $\tau_1(\mathtt{a}) = \tau_4(\mathtt{a})$ which is the conjunction of the indicator variables $\tau_1((X, d))$ for each $(X, d) \in \mathtt{a}$. The encoding $\tau_1(X) = \tau_4(X)$ consists of the following CNF formula:

$$(\bigvee_{d \in D_X} \tau_1((X, d))) \wedge (\bigwedge_{d_1, d_2 \in D_X \mid d_1 \neq d_2} \neg\tau_1((X, d_1)) \vee \neg\tau_1((X, d_2))).$$

Finally, in τ_1 and τ_4, the scaling factor $(w_1)_0 = (w_4)_0$ is 1.

Contrastingly, **ENC1** and **ENC4** differ in the way mappings R are encoded. In **ENC1**, each $\tau_1(R)$ is a CNF formula, consisting for each $\mathtt{a} \in Dom(R)$ of the following CNF formulae: $(\bigvee_{(X,d)\in\mathtt{a}} \neg\tau_1((X, d)) \vee \theta_\mathtt{a}) \wedge \bigwedge_{(X,d)\in\mathtt{a}} (\tau_1((X, d)) \vee \neg\theta_\mathtt{a})$. This formula contains $c \times (a + 1)$ clauses where c is the cardinality of $Dom(R)$ and a is the arity of R. Here, $\theta_\mathtt{a}$ is a parameter variable which is specific to $\mathtt{a}$. For each $\mathtt{a}$, the corresponding CNF formula actually states an equivalence between $\tau_1(\mathtt{a})$ and $\theta_\mathtt{a}$. Finally, $w_1(\theta_\mathtt{a}) = R(\mathtt{a})$.

In **ENC4**, for each $R \in \mathcal{R}$, one parameter variable θ_j per non-null weight in R is introduced, only. Thus, no parameter variable is introduced for the $\mathtt{a} \in Dom(R)$ such that $R(\mathtt{a}) = 0$. Furthermore, all the assignments $\mathtt{a} \in Dom(R)$ which are associated with the same value $R(\mathtt{a})$ are associated with the same parameter variable θ_j which is such that $w_4(\theta_j) = R(\mathtt{a})$. Each $\tau_4(R)$ is a CNF formula, obtained first by computing a compressed representation of R in a way similar to the way a simplification of a Boolean function f is computed using Quine/McCluskey algorithm, i.e., as a minimal number of prime implicants of f the disjunction of which being equivalent to f (see [29, 30, 24] and [10] for details). Once R has been compressed, $\tau_4(R)$ is computed as the conjunction for each $\mathtt{a} \in Dom(R)$ of the following clauses:
$\bigvee_{(X,d)\in\mathtt{a}} \neg\tau_4((X, d))$ if $R(\mathtt{a}) = 0$,
$\bigvee_{(X,d)\in\mathtt{a}} \neg\tau_4((X, d)) \vee \theta_j$ if $R(\mathtt{a}) \neq 0$.

Note that τ_4 by itself is not a faithful translation: the generated formula Σ_4 (the conjunction of all $\tau_4(X)$ for $X \in \mathcal{X}$ and of all $\tau_4(R)$ for $R \in \mathcal{R}$) must be *minimized* first w.r.t. its parameter variables in order to get a faithful translation. Such a "cardinality minimization", noted $min_\theta(\Sigma_4)$, leads to a strengthening of Σ_4, obtained by removing every model of it assigning to true more than one parameter variable associated with a given R. Now, for each variable $X \in \mathcal{X}$, given the CNF formula $\tau_4(X)$, exactly one of the indicator variables $\tau_4((X, d))$ for $d \in D_X$ can be set to true in a model of Σ_4. Accordingly, the "global cardinality minimization" $min(\Sigma_4)$ of Σ_4 (i.e., when "cardinality minimization" is w.r.t. *all* the variables) can be done instead, since we have $min(\Sigma_4) = min_\theta(\Sigma_4)$. The main point is that the mapping τ_4^{min} associating $\mathcal{WCN} = (\mathcal{X}, \mathcal{D}, \mathcal{R})$ with the WPROP $(min(\Sigma_4), w_4, (w_4)_0)$ is faithful. Interestingly, when Σ_4 has been turned first into an equivalent d-DNNF representation, such a "global cardinality minimization" process leading to a minimized d-DNNF representation $min(\Sigma_4)$ can be achieved in linear time [12].

Example 2 (Example 1 continued) *As a matter of illustration, let us present the encodings obtained by applying τ_1 and τ_4 to our running example. τ_1 and τ_4 are based on the same set consisting of* 5 *indicator variables, λ_i^j, where λ_i^j corresponds to the elementary assignment (X_i, j), and on the same set of indicator clauses:*

$\lambda_1^0 \vee \lambda_1^1 \vee \lambda_1^2$, $\neg\lambda_1^0 \vee \neg\lambda_1^2$, $\lambda_2^0 \vee \lambda_2^1$,
$\neg\lambda_1^0 \vee \neg\lambda_1^1$, $\neg\lambda_1^1 \vee \neg\lambda_1^2$, $\neg\lambda_2^0 \vee \neg\lambda_2^1$.

τ_1 and τ_4 differ as to their parameter variables, and as to their parameter clauses. For τ_1, one parameter variable per element of $Dom(R)$ (hence per line in Table 1) is introduced: each θ_i corresponds to line i, thus 6 *variables are introduced. For τ_4, one parameter variable per non-null value taken by R is considered, hence two parameter variables θ_1 (corresponding to $1/10$) and θ_2 (corresponding to $8/30$) are introduced. On this ground, $\tau_1(R)$ consists of the following parameter clauses:*

$\neg\lambda_1^0 \vee \neg\lambda_2^0 \vee \theta_1$, $\neg\lambda_1^1 \vee \neg\lambda_2^0 \vee \theta_3$, $\neg\lambda_1^2 \vee \neg\lambda_2^0 \vee \theta_5$,
$\lambda_1^0 \vee \neg\theta_1$, $\lambda_1^1 \vee \neg\theta_3$, $\lambda_1^2 \vee \neg\theta_5$,
$\lambda_2^0 \vee \neg\theta_1$, $\lambda_2^0 \vee \neg\theta_3$, $\lambda_2^0 \vee \neg\theta_5$,
$\neg\lambda_1^0 \vee \neg\lambda_2^1 \vee \theta_2$, $\neg\lambda_1^1 \vee \neg\lambda_2^1 \vee \theta_4$, $\neg\lambda_1^2 \vee \neg\lambda_2^1 \vee \theta_6$,
$\lambda_1^0 \vee \neg\theta_2$, $\lambda_1^1 \vee \neg\theta_4$, $\lambda_1^2 \vee \neg\theta_6$,
$\lambda_2^1 \vee \neg\theta_2$, $\lambda_2^1 \vee \neg\theta_4$, $\lambda_2^1 \vee \neg\theta_6$,

with $w_1(\theta_1) = 0$, $w_1(\theta_2) = w_1(\theta_5) = w_1(\theta_6) = 8/30$, $w_1(\theta_3) = w_1(\theta_4) = 1/10$, and every other literal has weight 1. *Σ_1 contains* 24 *clauses, over* 11 *variables.*

Contrastingly, with τ_4, R is first compressed into

X_1	X_2	R
0	0	0
0	1	$8/30$
1		$1/10$
2		$8/30$

As a consequence, $\tau_4(R)$ consists of the following parameter clauses:

$\neg\lambda_1^0 \vee \neg\lambda_2^0$, $\neg\lambda_1^1 \vee \theta_1$,
$\neg\lambda_1^0 \vee \neg\lambda_2^1 \vee \theta_2$, $\neg\lambda_1^2 \vee \theta_2$,

with $w_4(\theta_1) = 1/10$, $w_4(\theta_2) = 8/30$, and every other literal has weight 1. *Σ_4 contains* 10 *clauses, over* 7 *variables.*

4 A NEW, IMPROVED CNF ENCODING SCHEME

We present a new translation function τ_{4linp}, which is modular as τ_1 and τ_4. τ_{4linp} elaborates on τ_4 in two directions: the way elementary assignments are encoded, and the implicit handling of one parameter variable per mapping R.

Thus, within the translation function τ_{4linp}, *log encoding* (aka bitwise encoding) is used for the representation of elementary assignments (X, d). The corresponding $\tau_{4linp}(X)$ CNF formula aims at forbidding the interpretations which do not correspond to any elementary assignment. Thus, there is no such constraint (i.e., it is equivalent to $\top$) when the cardinality of the domain of X is a power of 2.

As to the parameter variables and the parameter clauses, our translation function τ_{4linp} is reminiscent to τ_4. However, there are some important differences. First, log encoding is used to define the indicator variables within the parameter clauses. Second, one parameter

variable θ_R per R is kept *implicit* once R has been compressed; it is selected as one of those θ_j such that $w_4(\theta_j) \neq 0$ is one of the most frequent weight in R once compressed. Then we take the scaling factor $(w_{4linp})_0$ to be equal to the product of all the weights $w_4(\theta_R)$ when R varies in $\mathcal{R}$, and we replace the weight $w_4(\theta_j)$ of all the remaining parameter variables θ_j associated with R by $w_{4linp}(\theta_j) = {}^{w_4(\theta_j)}/_{w_4(\theta_R)}$. The benefits achieved by this scaling come from the fact that there is no need to add any clause into $\tau_{4linp}(R)$ for the assignments $\mathtt{a}$ such that $R(\mathtt{a}) = w_4(\theta_R)$. More formally, for each $R \in \mathcal{R}$, R is first compressed as in **ENC4**; then we define $\tau_{4linp}(R)$ as a CNF formula, consisting of the conjunction for each $\mathtt{a} \in Dom(R)$ such that $R(\mathtt{a}) \neq w_4(\theta_R)$ of the following clauses:
$\bigvee_{(X,d)\in\mathtt{a}} \neg\tau_{4linp}((X,d))$ if $R(\mathtt{a}) = 0$,
$\bigvee_{(X,d)\in\mathtt{a}} \neg\tau_{4linp}((X,d)) \vee \theta_j$ if $R(\mathtt{a}) \neq 0$.
Here $\neg\tau_{4linp}((X,d))$ is the clause which is obtained as the disjunction of the negations of all literals occurring in $\tau_{4linp}((X,d))$.

Finally, considering the same weight distribution $w_{4linp} = w_4$ as the one considered in **ENC4** would not make the translation faithful; in order to ensure it, we now assign a specific weight to the negative parameter literals, so that $w_{4linp}(\neg\theta_j) = 1 - w_{4linp}(\theta_j)$ for every parameter variable θ_j considered in the parameter clauses of R, for every $R \in \mathcal{R}$.

As we will show it later, no minimization step is mandatory with τ_{4linp}; furthermore, this translation is modular (like τ_4 but unlike τ_4^{min}); more importantly, we obtain as a side effect that any weighted model counter can be considered downstream (unlike τ_4, which requires a minimization step).

Example 3 (Example 1 continued) *For the WCN considered in Example 1, one just needs to consider two indicator variables for encoding the elementary assignments associated with X_1 (let us say, λ_1^0 and λ_1^1) and one indicator variable for encoding the elementary assignments associated with X_2 (λ_2). The correspondances between elementary assignments and their representation as terms over the indicator variables are as follows for X_1:*

X_1	λ_1^1	λ_1^0
0	0	0
1	0	1
2	1	0

and for X_2, λ_2 corresponds to $(X_2, 1)$ (thus, $\neg\lambda_2$ corresponds to $(X_2, 0)$). We have $\tau_{4linp}(X_1) = \neg\lambda_1^1 \vee \neg\lambda_1^0$ and $\tau(X_2) = \top$. The most frequent value achieved by $R(\mathtt{a})$ is $w_4(\theta_R) = {}^8/_{30}$. Since $\mathcal{R} = \{R\}$, we get that $(w_{4linp})_0 = {}^8/_{30}$. $\tau_{4linp}(R)$ consists of the two following clauses:

$$\lambda_1^0 \vee \lambda_1^1 \vee \lambda_2, \qquad \lambda_1^1 \vee \neg\lambda_1^0 \vee \theta_1.$$

The first clause aims at ensuring that the weight corresponding to the full assignment $\{(X_1, 0), (X_2, 0)\}$ is 0. The purpose of the second clause is to enforce the parameter variable θ_1 to be true when any assignment extending $\{(X_1, 1)\}$ is considered. We finally have $w_{4linp}(\theta_1) = {}^3/_8$ and $w_{4linp}(\neg\theta_1) = {}^5/_8$, while every other literal has weight 1. Σ_{4linp} contains only 3 clauses, over 4 variables.
Table 2 makes precise for each interpretation over the variables λ_1^0, λ_1^1, λ_2, and θ_1 the corresponding full assignment of WCN over $\{X_1, X_2\}$ (if any) and the associated weight w_{4linp}.

Proposition 1 *τ_{4linp} is faithful.*

λ_1^1	λ_1^0	λ_2	θ_1	X_1	X_2	w_{4linp}
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	1	$1/6$
0	0	1	1	0	1	$1/10$
0	1	0	0	1	0	0
0	1	0	1	1	0	$1/10$
0	1	1	0	1	1	0
0	1	1	1	1	1	$1/10$
1	0	0	0	2	0	$1/6$
1	0	0	1	2	0	$1/10$
1	0	1	0	2	1	$1/6$
1	0	1	1	2	1	$1/10$
1	1	0	0	-	-	0
1	1	0	1	-	-	0
1	1	1	0	-	-	0
1	1	1	1	-	-	0

Table 2: The full assignment of WCN over $\{X_1, X_2\}$ and the the weight w_{4linp} associated with each interpretation over the variables of Σ_{4linp}.

Proof: By definition of log encoding, every (partial) assignment $\mathtt{a}$ of $\mathcal{WCN}$ over a subset $\mathcal{S}$ of $\mathcal{X}$ is associated with a term $\tau_{4linp}(\mathtt{a})$ over $\Lambda_{\mathcal{WCN}} = \bigcup_{X\in\mathcal{X}} \Lambda_X$. Furthermore, every full assignment $\mathtt{s}$ of $\mathcal{WCN}$ is associated in a bijective way with a term $\tau_{4linp}(\mathtt{s})$ over $\Lambda_{\mathcal{WCN}}$ which implies $\bigwedge_{X\in\mathcal{X}} \tau_{4linp}(X)$.

Let us recall that the weight of any full assignment $\mathtt{s}$ is $w_{\mathcal{WCN}}(\mathtt{s}) = 0$ when $\mathtt{s}$ is not a solution of $\mathcal{WCN}$ and is $w_{\mathcal{WCN}}(\mathtt{s}) = \Pi_{R\in\mathcal{R}} R(\mathtt{s}[scope(R)])$ otherwise.

Assume first that $w_{\mathcal{WCN}}(\mathtt{s}) = 0$. Then either $\mathtt{s}$ is not a solution of $\mathcal{WCN}$ or $\mathtt{s}$ is such that $R(\mathtt{s}[scope(R)]) = 0$ for at least one $R \in \mathcal{R}$. Hence there exists $R \in \mathcal{R}$ such that either $\mathtt{s}[scope(R)] \notin Dom(R)$ or $R(\mathtt{s}[scope(R)]) = 0$. Subsequently, there exists a clause in Σ_{4linp} such that $\tau_{4linp}(\mathtt{s})$ falsifies it. This implies that every interpretation over $\Lambda_{\mathcal{WCN}} \cup \Theta_R$ which extends $\tau_{4linp}(\mathtt{s})$ falsifies Σ_{4linp}. Accordingly, $w_{4linp}(\tau_{4linp}(\mathtt{s})) = 0$ as expected.

Assume now that $\mathtt{s}$ is such that $w_{\mathcal{WCN}}(\mathtt{s}) \neq 0$. By construction, for every $R \in \mathcal{R}$, the contribution of R to $w_{\mathcal{WCN}}(\mathtt{s})$ is equal to the factor $R(\mathtt{s}[scope(R)])$. Suppose that k parameter variables $\theta_1, \ldots, \theta_k$ have been introduced in $\tau_{4linp}(R)$. Then there are two cases to be considered: (1) $R(\mathtt{s}[scope(R)]) = w_4(\theta_R)$ and (2) $R(\mathtt{s}[scope(R)]) \neq w_4(\theta_R)$.

In case (1), by construction, $\tau_{4linp}(\mathtt{s})$ satisfies every clause of $\tau_{4linp}(R)$. Hence each of the 2^k canonical terms extending $\tau_{4linp}(\mathtt{s})$ over the k parameter variables implies $\tau_{4linp}(R)$. Therefore, the contribution of R to $w_{4linp}(\tau_{4linp}(\mathtt{s}))$ is equal to the sum, for each canonical term, of the products of the parameter literals occurring in it. But this sum is also equal to $\Pi_{i=1}^k (w_{4linp}(\theta_i) + w_{4linp}(\neg\theta_i)) = 1 = {}^{w_4(\theta_R)}/_{w_4(\theta_R)} = {}^{R(\mathtt{s}[scope(R)])}/_{w_4(\theta_R)}$.

In case (2), by construction, there is a clause $\neg\tau_{4linp}(\mathtt{s}[scope(R)]) \vee \theta_j$ in $\tau_{4linp}(R)$, so that the parameter variable θ_j is set to true in every model of Σ_{4linp} extending $\tau_{4linp}(\mathtt{s})$. As above, each of the 2^{k-1} canonical terms extending $\tau_{4linp}(\mathtt{s})$ over the $k-1$ remaining parameter variables (i.e., all of them but θ_j) implies $\tau_{4linp}(R)$. Therefore, the contribution of R to $w_{4linp}(\tau_{4linp}(\mathtt{s}))$ is equal to the sum, for each canonical term, of the products of the parameter literals occurring in it. But this sum is also equal to ${}^{R(\mathtt{s}[scope(R)])}/_{w_4(\theta_R)} \times \Pi_{i=1|i\neq j}^k (w_{4linp}(\theta_i) + w_{4linp}(\neg\theta_i)) = {}^{R(\mathtt{s}[scope(R)])}/_{w_4(\theta_R)}$.

Whatever the case (1) or (2), since $(w_{4linp})_0$ is equal to $\Pi_{R\in\mathcal{R}} w_4(\theta_R)$, the factor $w_4(\theta_R)$ of this product balances the denominator of the ratio ${}^{w_4(\theta_R)}/_{w_4(\theta_R)}$, so that finally, $w_{4linp}(\tau_{4linp}(\mathtt{s})) = \Pi_{R\in\mathcal{R}} R(\mathtt{s}[scope(R)]) = w_{\mathcal{WCN}}(\mathtt{s})$ as expected. ■

Our purpose was also to compare the efficiency of τ_{4linp} w.r.t. the

efficiency of τ_4, where the efficiency is measured as the number of variables and/or as the number of clauses in the corresponding CNF encodings Σ_{4linp} and Σ_4. We obtained that τ_{4linp} is more efficient than τ_4 for both measures:

Proposition 2 *Given $\mathcal{WCN} = (\mathcal{X}, \mathcal{D}, \mathcal{R})$, let $\tau_4(\mathcal{WCN}) = (\Sigma_4, w_4, (w_4)_0)$, and $\tau_{4linp}(\mathcal{WCN}) = (\Sigma_{4linp}, w_{4linp}, (w_{4linp})_0)$. Then we have:*

- $\#var(\Sigma_{4linp}) < \#var(\Sigma_4)$,
- $\#cl(\Sigma_{4linp}) < \#cl(\Sigma_4)$.

Proof:

- $\#var$. When the cardinality of D_X is k, $\tau_4(X)$ uses k indicator variables, while $\tau_{4linp}(X)$ requires only $\lceil log_2(k) \rceil$ indicator variables. As to the parameter variables, by construction, $\tau_{4linp}(R)$ requires one variable less than $\tau_4(R)$.
- $\#cl$. By construction, $\tau_4(X)$ contains $k \times (k-1)+1$ clauses when k is the cardinality of D_X. Contrastingly, $\tau_{4linp}(X)$ contains at most $k-2$ "blocking clauses" (this worst case is obtained when $k = 2^l+1$ for some l). Hence, the number of clauses in $\tau_{4linp}(X)$ is strictly lower than the number of clauses in $\tau_4(X)$. Furthermore, by construction, $\tau_{4linp}(R)$ contains at least one clause less than $\tau_4(R)$ (this worst case situation is obtained when all the values $w(\theta_j) \neq 0$ of the parameter variables θ_j considered by $\tau_4(R)$ are distinct).

■

5 EXPERIMENTS

Our benchmarks consist of 1452 WCNs downloaded from http://www.hlt.utdallas.edu/~vgogate/uai14-competition/index.html and http://reasoning.cs.ucla.edu/ace/. Those instances correspond to Bayesian networks or random Markov fields in the UAI competition format. They are gathered into 6 data sets, as follows: Diagnose (100), UAI (377), Grids (320), Pedigree (22), Promedas (238), Relational (395).

We translated each input WCN into a WPROP, using both the τ_4 and the τ_{4linp} translation function. Downstream to the encoding, we took advantage of the C2D compiler which targets the Decision-DNNF language [12, 15] to compute, for each instance, a minimized Decision-DNNF representation of the CNF formula generated by τ_4, and a Decision-DNNF representation of the CNF formula generated by τ_{4linp}. C2D has been run with its default parameters. Note that we could also consider a model counter (like Cachet, which supports weights) downstream to the CNF encoding produced by τ_{4linp}. For space reasons, we refrain from reporting the corresponding empirical results here because C2D performs often much better that Cachet on CNF instances issued from graphical models (the dtree computed to guide the Decision-DNNF computation achieved by C2D has a major positive impact on the process).

Our experiments have been conducted on a Quad-core Intel XEON X5550 with 32GiB of memory. A time limit of 900s for the compilation phase (including the translation time and the minimization time when τ_4 has been used[4]), and a total amount of 8GiB of memory for storing the resulting Decision-DNNF representations have been considered for each instance. Both the instances used in our experiments, the run-time code of our translator bn2Cnf implementing the τ_4 encoding scheme and the τ_{4linp} encoding scheme, and some detailed empirical results are available on line from http://www.cril.fr/KC.

In order to figure out the reductions in the number of variables and in the number of clauses done by τ_{4linp} compared to τ_4, we computed the number of variables $\#var$ and the number of clauses $\#cl$ of Σ_{4linp} and Σ_4 for each instance. Some of our empirical results are depicted using scatter plots with logarithmic scales. Thus, the scatter plots (a) and (b) from Figure 1 report respectively the relative performances of τ_4 and τ_{4linp} w.r.t. the measurements $\#var$ and $\#cl$. They cohere with Proposition 2 and show that τ_{4linp} leads in practice to CNF encodings which are exponentially smaller w.r.t. both the number of variables and the number of clauses than those produced by τ_4.

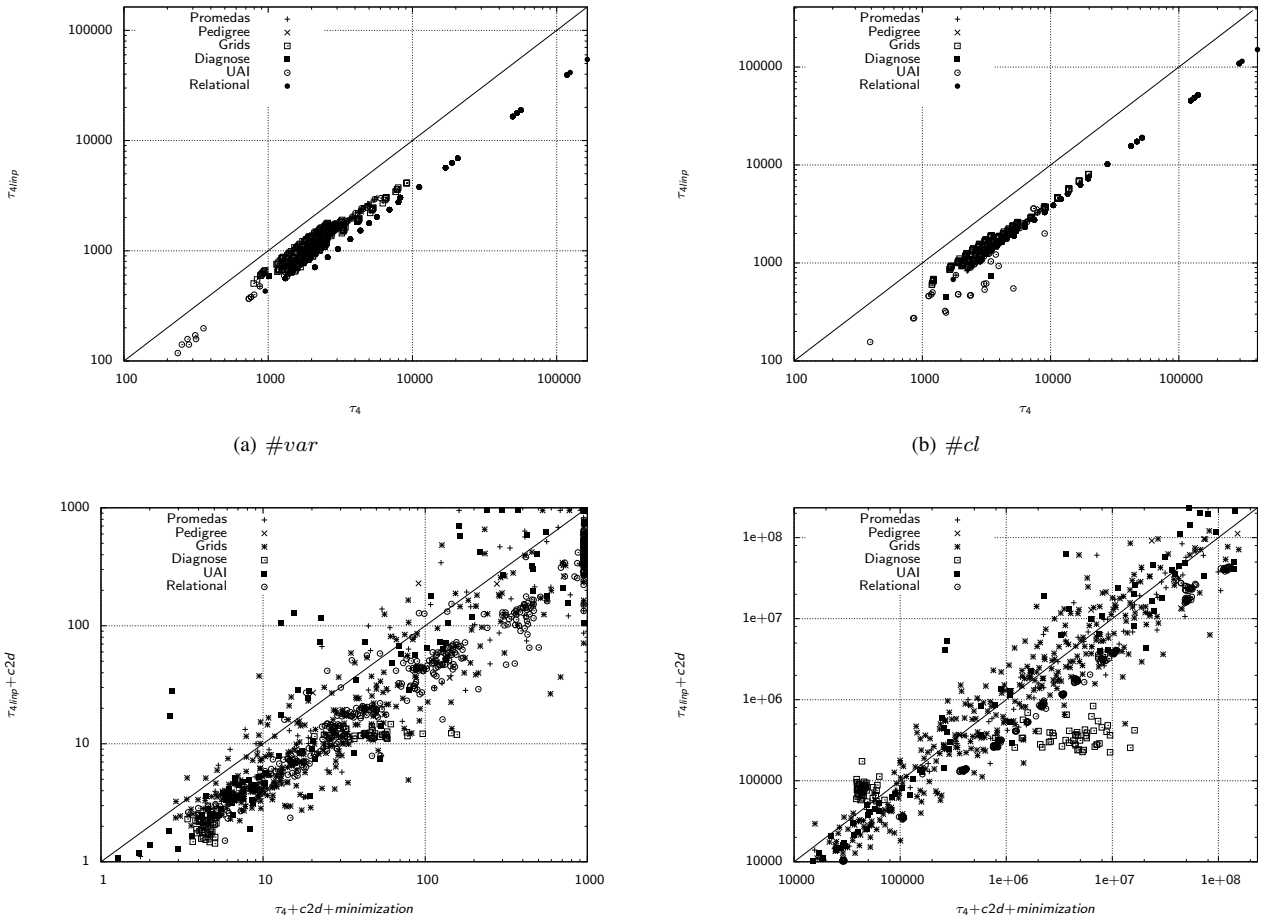
The two scatter plots (c) and (d) from Figure 1 report respectively the CPU times (in seconds) needed to compute the Decision-DNNF representations associated with the input WCNs (for each of the two encoding schemes τ_4 and τ_{4linp}) and make precise the sizes (in number of arcs) of those Decision-DNNF representations.

Table 3 presents a selection of the results available from http://www.cril.fr/KC and used in the scatter plots from Figure 1. The columns of the table make precise, from the leftmost one to the rightmost one:

- data about the input instance, namely:
  - the family of the input WCN, among the six families considered in the experiments;
  - the type of the instance (Bayes net or Markov net);
  - the name of the instance;
  - the number of variables of the instance;
  - the number of tables of the instance;
  - the cardinality of (one of) the largest domain(s) of a variable of the instance;
  - the arity of (one of) the relations of the instance, of largest arity;
  - the total number of tuples in the instance (i.e., the sum of the cardinalities of the relations);
  - the sum of the cardinalities of the domains of the variables;
- and for each of the two encoding schemes τ_4 and τ_{4linp} under consideration:
  - the number of variables in the CNF encoding of the instance;
  - the number of clauses in the CNF encoding of the instance;
  - the time (in seconds) required to generate the CNF encoding, plus the time needed by C2D to generate a Decision-DNNF representation from it (and to minimize it when τ_4 has been used);
  - the size (in number of arcs) of the resulting Decision-DNNF representation produced by C2D (after minimization when τ_4 has been used).

Clearly enough, the scatter plots (c) and (d) from Figure 1 as well as Table 3 illustrate the benefits that can be achieved by considering τ_{4linp} instead of τ_4 when C2D is used downstream. Indeed, τ_{4linp} led most of the time to improved compilation times and improved compilation sizes. To be more precise, as to the compilation times, τ_{4linp} proved strictly better than τ_4 for 911 instances (while τ_4 proved strictly better than τ_{4linp} for 87 instances). As to the sizes

[4] Minimization can be achieved in linear time on d-DNNF representations [12]. It may have a valuable reduction effect on the size of the compiled form.



(a) $\#var$

(b) $\#cl$

(c) Compilation times: τ_4+C2D +minimization vs. τ_{4linp}+C2D

(d) Compiled form sizes: τ_4+C2D +minimization vs. τ_{4linp}+C2D

Figure 1: Comparing τ_{4linp} with τ_4.

Instance									τ_4				τ_{4linp}			
Family	Name	Type	#var	#Rel	max dom.	max ari.	#tuples	#values	#var	#cl	time C2D	size C2D	#var	#cl	time C2D	size C2D
Promedas	or_chain_96.fg	MARKOV	719	719	2	3	4260	1438	3058	4663	216.4	3058	1620	1942	111.5	1620
Promedas	or_chain_223.fg	MARKOV	988	988	2	3	5754	1976	?	?	?	?	2268	2639	1427.4	2268
Promedas	or_chain_178.fg	MARKOV	1021	1021	2	3	5936	2042	?	?	?	?	2314	2715	816.3	2314
Promedas	or_chain_132.fg	MARKOV	723	723	2	3	4058	1446	3009	4522	95.6	3009	1563	1818	199.6	1563
Promedas	or_chain_86.fg	MARKOV	892	891	2	3	5020	1784	3789	5602	499.6	3789	?	?	?	?
Pedigree	pedigree23	MARKOV	402	402	5	4	5025	784	1479	2933	525.4	1479	737	1276	174.4	737
Pedigree	pedigree30	MARKOV	1289	1289	5	5	12819	2491	4802	8860	1836.2	4802	2468	3802	1282.6	2468
Pedigree	pedigree18	MARKOV	1184	1184	5	5	12198	2291	4407	8252	927.7	4407	2262	3560	1140.9	2262
Grids	50-20-8	BAYES	400	400	2	3	3042	800	2556	3428	872.8	2556	1756	1887	1073.0	1756
Grids	90-46-1	BAYES	2116	2116	2	3	16562	4232	?	?	?	?	3503	6727	87.9	3503
Grids	90-42-2	BAYES	1764	1764	2	3	13778	3528	6228	13756	57.2	6228	2700	5513	48.6	2700
Grids	90-50-7	BAYES	2500	2500	2	3	19602	5000	9222	19846	420.3	9222	?	?	?	?
Grids	90-50-8	BAYES	2500	2500	2	3	19602	5000	?	?	?	?	4131	8048	145.7	4131
Grids	75-26-4	BAYES	676	676	2	3	5202	1352	3020	5446	496.7	3020	1668	2468	376.4	1668
Diagnose	3073	BAYES	329	329	6	12	34704	763	1695	3436	151.5	1695	1020	741	27.0	1020
UAI	404.wcsp	MARKOV	100	710	4	3	4538	258	1678	3421	1653.5	1678	839	1037	777.1	839
UAI	moissac4.pre	BAYES	462	462	3	3	7308	1386	2593	7338	39.7	2593	1669	3585	32.5	1669
UAI	linkage_21	MARKOV	437	437	5	4	6698	941	1722	3638	1136.4	1722	?	?	?	?
UAI	prob005.pddl	MARKOV	2701	29534	2	6	125726	5402	?	?	?	?	2701	29534	249.7	2701
UAI	log-1	MARKOV	939	3785	2	5	16266	1878	5663	13393	45.0	5663	939	3785	11.4	939
UAI	CSP_13	MARKOV	100	710	4	3	4538	258	?	?	?	?	839	1037	468.9	839
Relational	blockmap_15_03-0003	BAYES	18787	18787	2	3	132436	37574	56451	141138	473.2	56451	18877	51827	152.4	18877
Relational	blockmap_20_01-0009	BAYES	39297	39297	2	3	278138	78594	?	?	?	?	39334	108649	303.5	39334
Relational	blockmap_22_02-0006	BAYES	56873	56873	2	3	405240	113746	?	?	?	?	56955	157979	625.8	56955
Relational	mastermind_10_08_03-0004	BAYES	2606	2606	2	3	18658	5212	8250	19699	277.7	8250	3038	7446	176.5	3038
Relational	blockmap_20_01-0008	BAYES	39297	39297	2	3	278138	78594	?	?	?	?	39334	108649	364.7	39334
Relational	blockmap_22_03-0008	BAYES	59404	59404	2	3	423452	118808	?	?	?	?	59533	165085	490.0	59533

Table 3: Comparing τ_{4linp} with τ_4. Each '?' means that the process aborted with a time-out or a memory-out.

of the compiled representations, τ_{4linp} proved strictly better than τ_4 for 759 instances (while τ_4 proved strictly better than τ_{4linp} for 239 instances). Using the τ_4 encoding scheme, C2D has been able to generate a Decision-DNNF for 903 instances over 1452 within the time and memory limits. Contrastingly, when equipped with τ_{4linp}, C2D has been able to generate a Decision-DNNF for 1007 instances using the same computational resource bounds.

In order to evaluate the impact of the two "ideas" used in our encoding, we also performed a differential evaluation. Table 4 reports the number of instances for which the whole process – encoding+compilation+minimization (when needed) – terminated before the time limit, when the input encoding scheme is, respectively, τ_4, τ_{4l} (log encoding of the indicator variables), τ_{4inp} (implicit encoding of the most frequent probability value per table), and τ_{4linp}.

τ_4	903
τ_{4l}	975
τ_{4inp}	982
τ_{4linp}	1007

Table 4: Number of instances compiled within a time limit of 900s.

The cactus plot given at Figure 2 makes precise for each of those four encodings, the number of instances processed successfully depending on the allocated time. Both Table 4 and Figure 2 show that each of the two "ideas" used in our encoding has a positive influence on the time needed to "compile" the input WCN.[5]

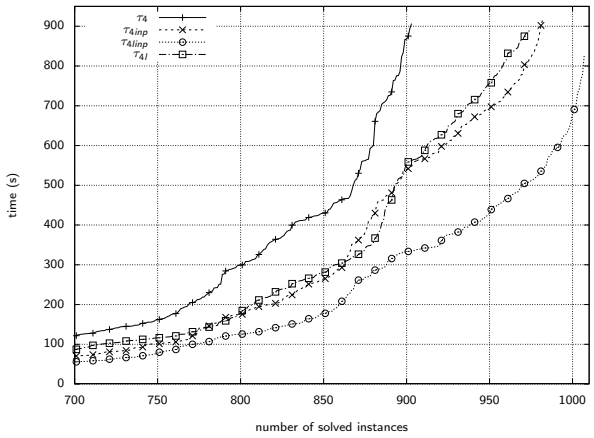


Figure 2: Number of instances compiled depending on the allocated time.

Finally, we also compared the performance of C2D empowered by our encoding scheme with those of ACE (version 3.0), a package that compiles a graphical model into an arithmetic circuit (AC) and then uses the AC to answer multiple queries with respect to the model, see http://reasoning.cs.ucla.edu/ace. In our experiments, logical model counting is used as a basis for compilation (we used the -forceC2d option of ACE for ensuring it). In this case, compilation proceeds by encoding the model into a propositional formula, compiling it into Decision-DNNF (using the C2D knowledge compiler), and extracting the AC from the compiled Decision-DNNF.

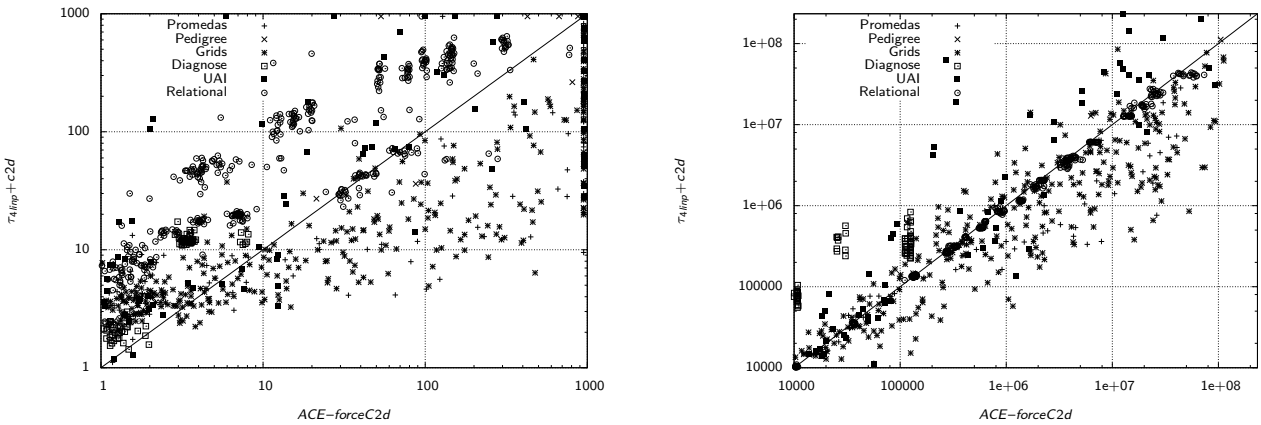
ACE is mainly based on **ENC4**, but incorporates several improvements; thus, *exactly_one* constraints (alias Eclauses) are generated in the encoding used by C2D (so that this encoding is not exactly a CNF encoding); such constraints are useful for representing the domains of the variables (they can replace the indicator clauses); furthermore, no parameter variable and no parameter clause are introduced for the $\mathtt{a} \in Dom(R)$ such that $R(\mathtt{a}) = 1$.

Like in the previous experiments reported in the paper, the comparison between τ_{4linp}+C2D and ACE -forceC2d mainly concerns the generation (using C2D) of a Decision-DNNF representation from an input WCN. However, there is a fundamental difference: in the previous experiments, nothing changed but the encoding under consideration; for this reason, it was possible to draw firm conclusions about the relative efficiency of the encodings; in the comparison with ACE, the situation is different because the input of C2D when run within ACE does not simply consist of the encoding of the given WCN. Indeed, a dtree *derived from the input WCN* (using the well-known minfill heuristic) is considered as well for guiding the compilation process. This dtree may easily be distinct from the one considered by C2D when computed from the encoding, only, and may lead to improved compilation times and compilation sizes. Accordingly, one must keep in mind that the empirical protocol used for comparing τ_{4linp}+C2D with ACE -forceC2d is not favorable to τ_{4linp}+C2D.

The scatter plots (a) and (b) from Figure 3 show respectively the compilation times and the compiled form sizes obtained by using τ_{4linp}+C2D on the one hand, and ACE -forceC2d on the other hand. As to the compilation times, τ_{4linp}+C2D proved strictly better than ACE -forceC2d for 335 instances (while ACE -forceC2d proved strictly better than τ_{4linp}+C2D for 667 instances). As to the sizes of the compiled representations, τ_{4linp}+C2D proved strictly better than ACE -forceC2d for 676 instances (while ACE -forceC2d proved strictly better than τ_{4linp}+C2D for 326 instances). Overall, ACE -forceC2d has been able to generate a Decision-DNNF for 922 instances over 1452 within the time and memory limits. Contrastingly, τ_{4linp}+C2D has been able to generate a Decision-DNNF for 1007 instances using the same computational resource bounds.

Empirically, ACE -forceC2d appeared as a better performer than τ_{4linp}+C2D w.r.t. the compilation times. Here are two possible explanations for it. Firstly, the dtree computed derived from the input WCN can lead to a better decomposition, as explained above (this looks particularly salient for instances from the "Relational" family). Secondly, there are numerous instances for which ACE -forceC2d terminated within 10s, while τ_{4linp}+C2D did not. This can be explained by the fact that each reported time actually covers all the computation time required by the process starting from the input WCN and finishing with the generation of the resulting Decision-DNNF representation. Especially, it includes the time required to generate the dtree used by C2D, and this dtree generation time can be much smaller when the generation process exploits the structure of the given WCN than when its input is just an encoding of the WCN. On the other hand, the combination τ_{4linp}+C2D solved more instances than ACE -forceC2d within the time and memory limits and led to significantly smaller compiled representations in many cases. This is a further illustration of the practical benefits which can be achieved by taking advantage of our encoding τ_{4linp}.

[5] The computation times reported for τ_{4l} are lower bounds, since they do not include the times required for achieving the minimization step w.r.t. the parameter variables. Indeed, this step has not been implemented. Especially, given that $min(\Sigma_{4l}) \neq min_\theta(\Sigma_{4l})$, it was not possible to take advantage of the "global cardinality minimization" functionality offered by C2D to compute $min_\theta(\Sigma_{4l})$. Nevertheless, since cardinality minimization of a specific subset of variables is feasible efficiently from a Decision-DNNF representation, the approximation done does not question the conclusions drawn about the impact of the two "ideas" used in our encoding.



(a) Compilation times: ACE `-forceC2d` vs. τ_{4linp}+C2D

(b) Compiled form sizes: ACE `-forceC2d` vs. τ_{4linp}+C2D

Figure 3: Comparing ACE `-forceC2d` vs. τ_{4linp}+C2D.

6 OTHER RELATED WORK

Interestingly, the key ideas used in τ_{4linp} are not specific to the CNF encoding pointed out, but could also be exploited to define a CDNF encoding (i.e., a conjunction of DNF representations), which can serve as an input to the bottom-up SDD compiler [17]. This can prove useful since bypassing intermediate representations in CNF can lead in some cases to a more efficient compilation algorithm (sometimes by orders of magnitude) [11].

Let $\tau_{4linp-sdd}$ be the translation leading to the WPROP $(\Sigma_{4linp-sdd}, w_{4linp-sdd}, (w_{4linp-sdd})_0)$ where $\Sigma_{4linp-sdd} = \bigwedge_{X\in\mathcal{X}} \tau_{4linp-sdd}(X) \wedge \bigwedge_{R\in\mathcal{R}} \tau_{4linp-sdd}(R)$. We define $\tau_{4linp-sdd}(X) = \tau_{4linp}(X)$ for every $X \in \mathcal{X}$. Then for every $R \in \mathcal{R}$, $\tau_{4linp-sdd}(R)$ is a simplified DNF formula computed from the compressed representation of R as the disjunction of all terms $\tau_{4linp-sdd}(\mathtt{a})$ for $\mathtt{a} \in Dom(R)$ such that $R(\mathtt{a}) = w(\theta_R)$, and all terms $\tau_{4linp-sdd}(\mathtt{a}) \wedge \theta_j$ for $\mathtt{a} \in Dom(R)$ such that $R(\mathtt{a}) \neq 0$ and $R(\mathtt{a}) \neq w(\theta_R)$. The simplification step is achieved using Quine/McCluskey algorithm.[6] Let us finally define $w_{4linp-sdd}$ as w_{4linp} (and $(w_{4linp-sdd})_0 = (w_{4linp})_0$).

Proposition 3 *$\tau_{4linp-sdd}$ is faithful.*

Proof: The result comes easily from the fact that τ_{4linp} is faithful and that under $\bigwedge_{X\in\mathcal{X}} \tau_{4linp-sdd}(X)$ (equivalent to $\bigwedge_{X\in\mathcal{X}} \tau_{4linp}(X)$), each DNF formula $\tau_{4linp-sdd}(R)$ is equivalent to the CNF formula $\tau_{4linp}(R)$. ■

Example 4 (Example 1 continued) *$\tau_{4linp-sdd}(R)$ is computed by considering first the DNF representation reported in the next table (left part), where the last line corresponds to a don't care.*

λ_1^1	λ_1^0	λ_2	θ_1
0	0	1	
1	0		
0	1		1
1	1		

λ_1^1	λ_1^0	λ_2	θ_1
	0	1	
1			
	1		1

This DNF representation is then simplified, leading to the DNF representation reported in the table (right part), equivalent to

$$\lambda_1^1 \vee (\neg\lambda_1^0 \wedge \lambda_2) \vee (\lambda_1^0 \wedge \theta_1).$$

[6] Terms conflicting with $\bigwedge_{X\in scope(R)} \tau_{4linp-sdd}(X)$ can also be added as don't cares prior to the simplification step; this may lead to smaller DNF representations.

This DNF representation is also equivalent under $\bigwedge_{X\in\mathcal{X}} \tau_{4linp}(X) = \neg\lambda_1^1 \vee \neg\lambda_1^0$ to

$$\tau_{4linp}(R) = (\lambda_1^0 \vee\lambda_1^1 \vee\lambda_2) \wedge(\lambda_1^1 \vee \neg\lambda_1^0 \vee \theta_1).$$

7 CONCLUSION

We have presented a new CNF encoding scheme τ_{4linp} for reducing probabilistic inference from a graphical model to weighted model counting. This scheme takes advantage of log encodings of the elementary variable/value assignments and of the implicit encoding of the most frequent probability value per conditional probability table. We have proved that τ_{4linp} is faithful. Experiments have shown that τ_{4linp} can be useful in practice; especially, the C2D compiler empowered by it performs in many cases significantly better than when **ENC4** is used, or when ACE is considered instead.

This work opens several perspectives for further research. From the practical side, we set a time limit to 900s in our experiments and we did not repeat the computations with C2D because the number of instances considered (1452) was large. However, default settings of C2D uses randomization to generate dtrees, which guide the compilation process and may have a huge impact on the total process. Thus we plan to repeat the experiments a few times with a greater time limit, averaging the obtained results to minimize the effect of randomization. On a different, yet empirical perspective, we plan also to compare the performances of τ_{4linp}+C2D with those of ACE `-forceC2d`, when C2D is guided in both cases by a dtree derived from the input network.

On the other hand, instead of associating specific weights with the negative parameter literals, it would be enough to ask (via the introduction of a further constraint) that at most one parameter variable for any relation $R \in \mathcal{R}$ is set to true. Our preliminary investigation showed that, empirically, this approach is less efficient than τ_{4linp} when one considers the compilation times obtained by C2D used downstream, but also that it leads to compiled representations which are typically of smaller sizes. It would be interesting to look for a trade-off by taking advantage of the two approaches (introducing specific weights for negative parameter literals for some R and introducing *at_most_one* constraints for other R). In the future, we plan also to evaluate in practice the benefits offered by such approaches when Decision-DNNF is targeted, and by the $\tau_{4linp-sdd}$ translation when SDD is targeted.

REFERENCES

[1] A. Antonucci, Y. Sun, C. Polpo de Campos, and M. Zaffalon, 'Generalized loopy 2u: A new algorithm for approximate inference in credal networks', *Int. J. Approx. Reasoning*, **51**(5), 474–484, (2010).

[2] F. Bacchus, S. Dalmao, and T. Pitassi, 'Algorithms and complexity results for #sat and Bayesian inference', in *Proc. of FOCS'03*, pp. 340–351, (2003).

[3] F. Bacchus, S. Dalmao, and T. Pitassi, 'Value elimination: Bayesian interence via backtracking search', in *Proc. of UAI'03*, pp. 20–28, (2003).

[4] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller, 'Context-specific independence in bayesian networks', in *Proc. of UAI'96*, pp. 115–123, (1996).

[5] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi, 'Distribution-aware sampling and weighted model counting for SAT', in *Proc. of AAAI'14*, pp. 1722–1730, (2014).

[6] S. Chakraborty, D. Fried, K.S. Meel, and M.Y. Vardi, 'From weighted to unweighted model counting', in *Proc. of IJCAI'15*, pp. 689–695, (2015).

[7] M. Chavira and A. Darwiche, 'Compiling bayesian networks with local structure', in *Proc. of IJCAI'05*, pp. 1306–1312, (2005).

[8] M. Chavira and A. Darwiche, 'Encoding CNFs to empower component analysis', in *Proc. of SAT'06*, pp. 61–74, (2006).

[9] M. Chavira and A. Darwiche, 'Compiling Bayesian networks using variable elimination', in *Proc. of IJCAI'07*, pp. 2443–2449, (2007).

[10] M. Chavira and A. Darwiche, 'On probabilistic inference by weighted model counting', *Artificial Intelligence*, **172**(6-7), 772–799, (2008).

[11] A. Choi, D. Kisa, and A. Darwiche, 'Compiling probabilistic graphical models using sentential decision diagrams', in *Proc. of ECSQARU'13*, pp. 121–132, (2013).

[12] A. Darwiche, 'Decomposable negation normal form', *Journal of the ACM*, **48**(4), 608–647, (2001).

[13] A. Darwiche, 'A compiler for deterministic decomposable negation normal form', in *AAAI'02*, pp. 627–634, (2002).

[14] A. Darwiche, 'A logical approach to factoring belief networks', in *Proc. of KR'02*, pp. 409–420, (2002).

[15] A. Darwiche, 'New advances in compiling CNF into decomposable negation normal form', in *Proc. of ECAI'04*, pp. 328–332, (2004).

[16] A. Darwiche, *Modeling and Reasoning with Bayesian Networks*, Cambridge University Press, 2009.

[17] A. Darwiche, 'SDD: A new canonical representation of propositional knowledge bases', in *Proc. of IJCAI'11*, pp. 819–826, (2011).

[18] F. J. Díez and S. F. Galán, 'Efficient computation for the noisy MAX', *Int. J. of Intelligent Systems*, **18**(2), 165–177, (2003).

[19] C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman, 'From sampling to model counting', in *Proc. of IJCAI'07*, pp. 2293–2299, (2007).

[20] C. P. Gomes, A. Sabharwal, and B. Selman, 'Model counting', in *Handbook of Satisfiability*, 633–654, (2009).

[21] J.-M. Lagniez and P. Marquis, 'Preprocessing for propositional model counting', in *Proc. of AAAI'14*, pp. 2688–2694, (2014).

[22] D. Larkin and R. Dechter, 'Bayesian inference in the presence of determinism', in *Proc. of AISTATS'03*, (2003).

[23] W. Li, P. Poupart, and P. van Beek, 'Exploiting structure in weighted model counting approaches to probabilistic inference', *J. of Artificial Intelligence Research*, **40**, 729–765, (2011).

[24] E.J. McCluskey, 'Minimization of Boolean functions', *Bell System Technical Journal*, **35**(6), 1417–1444, (1956).

[25] Ch.J. Muise, Sh.A. McIlraith, J.Ch. Beck, and E.I. Hsu, 'Dsharp: Fast d-DNNF compilation with sharpSAT', in *Proc. of AI'12*, pp. 356–361, (2012).

[26] U. Oztok and A. Darwiche, 'On compiling CNF into Decision-DNNF', in *Proc. of CP'14*, pp. 42–57, (2014).

[27] U. Oztok and A. Darwiche, 'A top-down compiler for sentential decision diagrams', in *Proc. of IJCAI'15*, pp. 3141–3148, (2015).

[28] D. Poole and N.L. Zhang, 'Exploiting contextual independence in probabilistic inference', *J. of Artificial Intelligence Research*, **18**, 263–313, (2003).

[29] W.V.O. Quine, 'The problem of simplifying truth functions', *American Mathematical Monthly*, **59**, 521–531, (1952).

[30] W.V.O. Quine, 'A way to simplify truth functions', *American Mathematical Monthly*, **62**, 627–631, (1955).

[31] M. Samer and S. Szeider, 'Algorithms for propositional model counting', *J. Discrete Algorithms*, **8**(1), 50–64, (2010).

[32] T. Sang, F. Bacchus, P. Beame, H.A. Kautz, and T. Pitassi, 'Combining component caching and clause learning for effective model counting', in *Proc. of SAT'04*, (2004).

[33] T. Sang, P. Beame, and H. A. Kautz, 'Performing Bayesian inference by weighted model counting', in *Proc. of AAAI'05*, pp. 475–482, (2005).

[34] M. Takikawa and B. D'Ambrosio, 'Multiplicative factorization of noisy-max', in *Proc. of UAI'99*, pp. 622–630, (1999).

[35] M. Thurley, 'sharpSAT - counting models with advanced component caching and implicit BCP', in *Proc. of SAT'06*, pp. 424–429, (2006).

[36] J. Vomlel and P. Tichavský, 'Probabilistic inference in BN2T models by weighted model counting', in *Proc. of SCAI'13*, pp. 275–284, (2013).

[37] M. Wachter and R. Haenni, 'Logical compilation of Bayesian networks with discrete variables', in *Proc. of ECSQARU'07*, pp. 536–547, (2007).