A Relaxed Tseitin Transformation for Weighted Model Counting

Wannes Meert, Jonas Vlasselaer
Computer Science Department
KU Leuven, Belgium

Guy Van den Broeck
Computer Science Department
University of California, Los Angeles

Abstract

The task of Weighted Model Counting is to compute the sum of the weights of all satisfying assignments of a propositional sentence. One recent key insight is that, by allowing negative weights, one can restructure the sentence to obtain a representation that allows for more efficient counting. This has been shown for formulas representing Bayesian networks with noisy-OR structures (Vomlel and Savicky 2008; Li, Poupart, and van Beek 2011) and for first-order model counting (Van den Broeck, Meert, and Darwiche 2014). In this work, we introduce the relaxed Tseitin transformation and show that the aforementioned techniques are special cases of this relaxation.

Introduction

Weighted model counting (WMC) is a generalization of model counting (Gomes, Sabharwal, and Selman 2009). Where in model counting the task is to count the number of satisfying assignments of a propositional sentence, in WMC each of these assignments has a weight and the task is to return the sum of these weights. One application of WMC is to exact inference in probabilistic models. For example, Bayesian networks as well as relational models (e.g. ProbLog, Primula) can reduce the task of exact probabilistic inference into one of WMC (Chavira and Darwiche 2008; Chavira, Darwiche, and Jaeger 2006; Fierens et al. 2015). One typical approach is to first encode the model as a propositional formula and then compile this formula into a more tractable target representation.

While most standard encodings only employ positive weights, the use of negative weights allows one to restructure the sentence for more efficient weighted model counting. This has been shown for formulas representing Bayesian networks with noisy-OR structures, where (negative) weights correspond to (negative) probabilities (Vomlel and Savicky 2008; Li, Poupart, and van Beek 2011). This is particularly relevant for relational models because probabilistic logic programming languages such as ProbLog (De Raedt, Kimmig, and Toivonen 2007), PRISM (Sato 1995), ICL (Poole 2008) and Primula (Jaeger 1997) use noisy-OR as default or optional combination rule. More recently, negative weights have also been used for first-order model counting (Van den Broeck, Meert, and Darwiche 2014) and allows for lifted inference in first-order theories with existential quantifiers. In general, the complex logical relationships encoded in statistical relational models can be difficult to represent as a (flat) WMC problem. Negative weights alleviate this, provide novel reductions between relational representations (Jha and Suciu 2012), and can even lead to increased expressiveness (Buchman and Poole 2016).

Our first contribution is that we introduce the *Relaxed Tseitin Transformation* (RTT) which allows for more efficient WMC on complex sentences. We show that the aforementioned techniques are special cases of this relaxation operation. Our second contribution is that we observe how *smoothing* can play a key-role when performing WMC. Smoothing is an operation that accounts for the weight of variables not represented by a subsentence during WMC. In the context of knowledge compilation, smoothing can be applied during compilation, i.e. encoded in the representation, or on-the-fly, i.e. during counting. While the first approach can lead to an explosion of the compiled representation, the overhead of the second approach is only minimal.

Background

We start by reviewing the necessary background on propositional logic, weighted model counting and the Tseitin transformation.

Propositional Logic

A propositional variable is a Boolean or binary variable, for example, P and Q. With lowercase letters we indicate the truth value assigned to a variable, $P = \mathit{true}$ is written as p and $P = \mathit{false}$ as $\neg p$. An interpretation is a truth-value assignment to all variables. A propositional sentence is a propositional variable, the negation of a sentence ($\neg P$), the conjunction of two sentences ($P \wedge Q$), or the disjunction of two sentences ($P \vee Q$). Negation, conjunction and disjunction are called logical connectives. Other logical connectives such as implication ($P \rightarrow Q$) or equivalence ($P \leftrightarrow Q$) can be defined in terms of the three primitive connectives. A

propositional literal is either a propositional variable P, called a positive literal, or the negation of a propositional variable $\neg P$, called a negative literal. A clause is a disjunction of literals and a propositional sentence is in conjunctive normal form (CNF) if it is a conjunction of clauses. An interpretation ω that satisfies a sentence Δ (defined in the usual way), is denoted $\omega \models \Delta$ and called a *model* of that sentence.

Weighted Model Counting

Given a propositional sentence Δ, model counting returns the number of interpretations that satisfy the sentence. For weighted model counting (WMC), a weight is associated with every model, and the task is to compute the sum of all model weights. The weight of one model is the product of weights associated with literals:

$$WMC(\Delta) = \sum_{\omega \models \Delta} \prod_{l \in \omega} w(l)$$

where l are the literals in the model ω. The weight function $w(\cdot)$ assigns a weight to each literal. The input for WMC is typically in conjunctive normal form (CNF). Note that we permit literal weights to be *negative* numbers, which is crucial for the relaxation algorithm.

Tseitin Transformation

The Tseitin transformation takes as input an arbitrary logic sentence and produces a boolean formula in CNF (Tseitin 1970). For each logical connective, a new variable representing the result of the connective is introduced and replaces the subsentence. An equivalence is introduced between each subsentence and the newly introduced variable. Finally, these equivalences representing the substitutions can be transformed independently into CNF.

For example, the sentence $\phi := (P \vee Q) \wedge R$ would be translated to $T(\phi) := (X_1 \leftrightarrow P \vee Q) \wedge (X_2 \leftrightarrow X_1 \wedge R) \wedge X_2$. Next, each of the conjuncts can be transformed into CNF.

Relaxed Tseitin Transformation

Following and generalizing the approach introduced by Van den Broeck, Meert, and Darwiche (2014) for Skolemization for first-order weighted model counting, we can replace a subsentence with a Tseitin variable and relax the equivalence connective introduced by the Tseitin transformation. This allows us to negate any arbitrary subsentence while maintaining the weighted model count.

At a high level, the *relaxed Tseitin transformation* takes as input a tuple (Δ, w) where Δ is an arbitrary sentence, and w is the weight function. It returns as output a tuple (Δ', w') which has the same weighted model count as the input (Δ, w). Moreover, one subsentence ϕ in Δ is replaced by a new Tseitin variable in Δ', and Δ' contains additional sentences that involve the negation of the original subsentence ϕ. This encoding of the subsentence is different from the standard Tseitin transformation, which would add an equivalence between the subsentence ϕ and the Tseitin variable. After repeatedly applying this relaxed Tseitin transformation, Δ' can easily be turned into CNF and passed on to a WMC solver that requires CNF inputs.

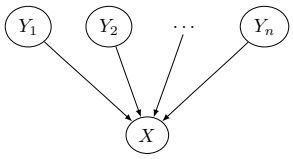


Figure 1: Noisy-OR Bayesian network

More specifically, sentence Δ' is obtained as follows (proof in Appendix). Suppose that Δ contains an arbitrary subsentence ϕ. First, we introduce two new variables: the *Tseitin variable* Z and the *Relaxation variable* R. Second, Δ' is a copy of Δ in which we replace the sentence ϕ in Δ by the atom Z, and instead of adding the sentence $Z \leftrightarrow \phi$ as in the standard Tseitin transformation, we append the sentences

$$\begin{aligned} Z &\vee \neg\phi \\ R &\vee \quad Z \\ R &\vee \neg\phi. \end{aligned}$$

The function w' is equal to w, except that $w'(z) = w'(\neg z) = w'(r) = 1$ and $w'(\neg r) = -1$. In the resulting theory Δ', the sentence ϕ is now present in the formula in its negated form. For a detailed intuition about this transformation, we refer to Van den Broeck, Meert, and Darwiche (2014) where the relaxed Tseitin transformation is applied to subsentences that are scoped by an existential quantifier.

Use Case: Noisy-OR

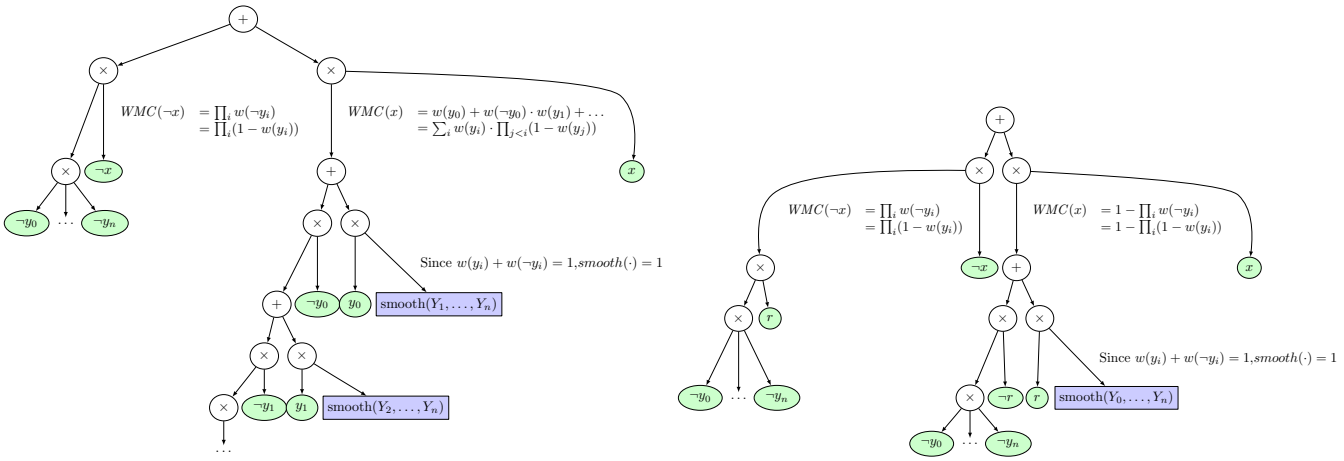
A Bayesian network can be represented as a propositional logic formula and the inference task can be formulated as a WMC task (Darwiche 2003). We will show this representation on a specific type of Bayesian network for which the relaxed Tseitin transformation is beneficial, a Noisy-OR structure. Suppose a Bayesian network with Boolean variables $X, Y_1, \ldots, Y_n$ and the structure shown in Figure 1. X is a noisy-OR combination of the Y_i if the conditional probability distribution of X is given by

$$Pr(x \mid Y_i, \ldots, Y_n) = 1 - \prod_{i:Y_i=true} (1 - p_i)$$

with $p_i = Pr(x \mid \neg y_1, \ldots, y_i, \ldots, \neg y_n)$. If all $p_i = 1$ the structure is a deterministic OR. The conditional probability distribution for X, $Pr(X \mid Y_1, \ldots, Y_n)$, can be encoded as:

$$X \leftrightarrow \bigvee_{i:Y_i=true} Y_i'$$

with weights $w(y_i') = Pr(x|\neg y_1, \ldots, y_i, \ldots, \neg y_n)$, $w(\neg y_i') = 1 - w(y_i')$ and $w(x) = w(\neg x) = 1$. In case



(a) Without relaxed Tseitin transformation.

(b) With relaxed Tseitin transformation.

Figure 2: Arithmetic circuits for noisy-OR. To improve the visualization, literal nodes with multiple incoming edges are duplicated per incoming edge and smoothing is summarized in one node per call.

all parents are observed to be true the probability distribution can be represented by the sentence Δ:

$$X \leftrightarrow Y'_0 \vee Y'_1 \vee \cdots \vee Y'_n$$

If we write this as a CNF, which is the expected format for WMC, Δ becomes:

$$\begin{aligned} &X \vee \neg Y'_0 \\ &X \vee \ldots \\ &X \vee \neg Y'_n \\ &\neg X \vee Y'_0 \vee Y'_1 \vee \cdots \vee Y'_n \end{aligned}$$

Instead of translating the original formula to CNF, we can first replace the disjunction $\phi := \bigvee_i Y'_i$ by a Tseitin variable Z. Second, we introduce the extra clauses that negate ϕ. The resulting Δ' is

$$\begin{aligned} &X \leftrightarrow Z \\ &Z \vee (\neg Y'_0 \wedge \cdots \wedge \neg Y'_n) \\ &R \vee \quad Z \\ &R \vee (\neg Y'_0 \wedge \cdots \wedge \neg Y'_n). \end{aligned}$$

After simplification and translation to CNF:

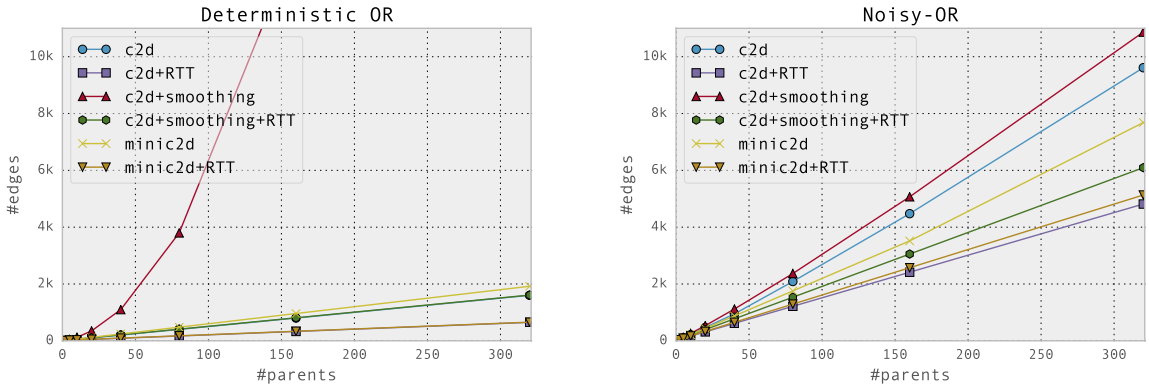
$$\begin{aligned} &X \vee \neg Y'_0 \\ &X \vee \ldots \\ &X \vee \neg Y'_n \\ &X \vee \quad R \\ &R \vee \neg Y'_0 \\ &R \vee \ldots \\ &R \vee \neg Y'_n \end{aligned}$$

with $w(r) = 1$ and $w(\neg r) = -1$.

If we compile this into the most compact arithmetic circuit we obtain the circuit depicted in Fig. 2a for Δ and the computation expressed by this circuit is equivalent to computing noisy-OR as $Pr(x \mid y_1, \ldots, y_n) = \sum_i w(y_i) \cdot \prod_{j<i} w(\neg y_j)$. For sentence Δ' obtained by applying the Tseitin transformation and negating the subsentence we obtain the circuit shown in Fig. 2b. This circuit represents the computation $Pr(x \mid y_1, \ldots, y_n) = 1 - \prod_i w(\neg y_i)$.

Link with multiplicative factorization. The resulting change in computation for a noisy-OR structure is identical to the transformation that is obtained by applying multiplicative factorization before performing an inference technique such as variable elimination or junction trees for Bayesian networks (Takikawa and D'Ambrosio 1999; Díez and Galán 2003) or for probabilistic logic programs (Meert, Struyf, and Blockeel 2010). The same technique has also been applied directly on the equivalent propositional formula (Li, Poupart, and van Beek 2011) or the arithmetic circuit (Vomlel and Savicky 2008). As such, multiplicative factorization can be considered as a special case of the relaxed Tseitin transformation. The common conclusion of the work on multiplicative factorization is that the representation after applying multiplicative factorization results in more efficient inference or a more compact circuit because it is less sensitive to a suboptimal variable orderings.

The role of smoothing. An arithmetic circuit used by model counting is derived from a logical formula in negation normal form that satisfies decomposability (conjunctions do not share variables), determinism (disjuncts must be logically incompatible), and smoothness (disjuncts must mention the same sets of variables) (Chavira and Darwiche 2008). The last property, smoothness, can be established easily by a post-processing step. Smoothness is required, together with



(a) Arithmetic circuit size for Determinstic OR

(b) Arithmetic circuit size for Noisy-OR

Figure 3: Results for Deterministic and Noisy-OR examples

determinism, to allow for correct counting when using the formula $WMC(\vee_i \alpha_i) = \sum_i WMC(\alpha_i)$ where α_i are the children of a disjunction in the circuit. The result of smoothing is shown separately in Fig. 2 because different strategies are used to deal with smoothing and this can have a significant impact on the efficiency of weighted model counting. Originally, smoothing was dealt with in a static manner and encoded as part of the circuit (Darwiche 2003) but recent methods tackle smoothing dynamically while performing weighted model counting (Oztok and Darwiche 2015). The main advantage of dynamic smoothing is that it allows for more efficient implementations. For example, by using division as in the Hugin architecture instead of only multiplication and addition when expressed as an arithmetic circuit (Darwiche 2009, Sec. 7.7). The static approach makes that the circuit in Fig. 2a has quadratic complexity in the number of parents while the dynamic approach has potentially only linear complexity.

Experiments

For the experiments we focus on substituting disjunctions that are present in the encoding of two popular structures in probabilistic logic programs: a regular deterministic OR structure and a noisy-OR structure. We use the CNF encoding as proposed by Fierens et al. (2015) and implemented the relaxed Tseitin transformation as part of the ProbLog package[1].

To perform weighted model counting we rely on the `c2d` d-DNNF compiler that is also part of the ACE package for Bayesian networks (Darwiche 2004) and the `miniC2D` SDD compiler (Oztok and Darwiche 2015). For `c2d` we use the reduce setting, the default elimination order and run the compiler with and without smoothing. For `miniC2D` we use the minfill elimination order as it returned the smallest circuits for the experiments in this section. Since `miniC2D` performs dynamic smoothing we only show circuits without smoothing. Inference has linear complexity in the number of edges in the arithmetic circuit, therefore we report the number of edges.

[1] `https://dtai.cs.kuleuven.be/problog`

Deterministic OR

To analyze inference for a deterministic OR structure we build an arithmetic circuit for the following probabilistic logic program (with three parents):

```
0.5::y(a).  0.5::y(b).  0.5::y(c).
x :- y(V).
query(x).
```

The arithmetic circuits found by `c2d` and `miniC2D` for the CNFs without the relaxed Tseitin transformation is almost identical to the linear circuit shown in Figure 2a. The sizes of the circuits without smoothing grow slightly faster than linear ($O(\#p^{1.1})$, see Figure 3a) because the compilers fail to find exactly one long linear branch but split this branch in two or more parts. For `c2d+smoothing`, the part of the circuit that performs smoothing, however, results in a circuit with a quadratic complexity in the number of parents.

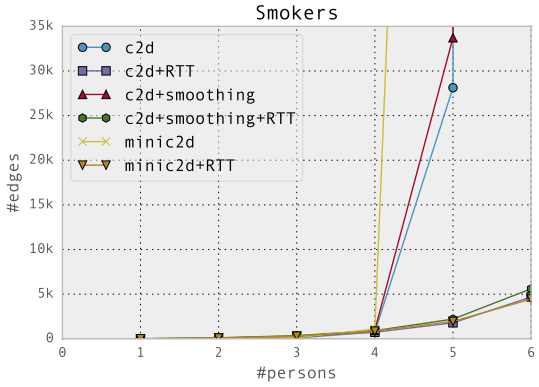
After applying the relaxed Tseitin transformation both methods find smaller circuits because the circuits shown in Figure 2b needs fewer nodes and edges per additional parent. Also these circuits are exactly linear in the number of parents because the circuit is less impacted by a suboptimal elimination ordering.

Noisy-OR

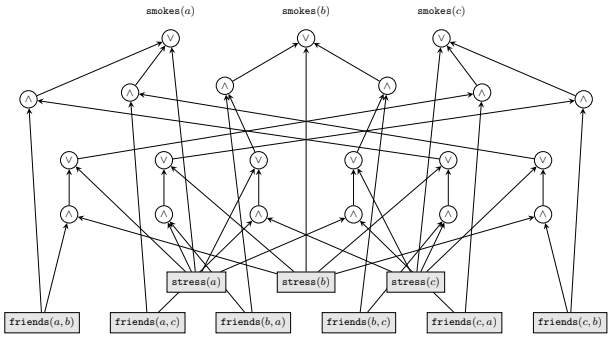
To perform inference for a noisy-OR structure we build an arithmetic circuit for the following program (with three parents):

```
0.5::y(a).  0.5::y(b).  0.5::y(c).
0.5::x :- y(V).
query(x).
```

Also for this program, `c2d` and `miniC2D` manage to find circuits that are almost linear in the number of parents if we ignore smoothing. Interestingly, the penalty



(a) Arithmetic circuit size for Smokers



(b) Boolean sentence for Smokers that is given to WMC

Figure 4: Results for Smokers example

of smoothing is less for noisy-OR. The smoothed circuit only grows with a rate of $O(\#p^{1.12})$. This is because `c2d` does not find the long linear branch and smoothing is split over a number of shorter linear branches. This leads to a larger circuit if we ignore smoothing but it reduces the impact of smoothing significantly and thus the overall size of the circuit.

After applying the relaxed Tseitin transformation both methods, with or without smoothing, return a circuit that is linear in the number of parents and with a smaller number of edges.

Smokers

A frequently encountered statistical relational example is the smokers program. We follow the example and methodology shown in Vlasselaer et al. (2014) to transform the following program to a WMC problem:

```
person(a).  person(b).  person(c).
0.2::stress(X) :- person(X).
0.1::friends(X,Y) :- person(X), person(Y).
smokes(X) :- stress(X).
smokes(X) :- friends(X,Y), smokes(Y).
query(smokes(a)).
```

We use the worst-case setting where everybody is potentially friends with everybody. The last rule expresses a deterministic OR where the parents are cyclically dependent on each other. Such programs are translated to a complex sentence (see Figure 4b) that is given to a WMC system.

Figure 4a shows that both `c2d` and `miniC2D` have difficulties finding a compact circuit. After applying the relaxed Tseitin transformation, the size of the circuits decreases by a factor of 10.

Discussion

These experiments confirm the results shown in previous research on multiplicative factorization for variable elimination (Takikawa and D'Ambrosio 1999; Díez and Galán 2003) and for arithmetic circuits (Vomlel and Savicky 2008; Li, Poupart, and van Beek 2011) when we consider circuits that implement smoothing. When a noisy-OR structure is considered, multiplicative factorization results in a lower tree-width and thus a more efficient computation. When smoothing is dealt with separately, the impact of multiplicative factorization is less pronounced for a noisy-OR structure.

Conclusion

We introduce a Tseitin transformation with relaxation which allows for more efficient WMC by introducing negative weights. This operation is a generalization of popular transformations such as multiplicative factorization for noisy-OR structures and Skolemization for first-order weighted model counting. In general, it allows one to replace any subsentence in a theory by its negation.

Acknowledgments

Jonas Vlasselaer is supported by IWT (agentschap voor Innovatie door Wetenschap en Technologie).

Appendix: Proof of Relaxed Tseitin Transformation

We prove correctness of the replacement of a subsentence ϕ in Δ to obtain Δ', and w'.

Introduce the Tseitin variable Introduce a new *Tseitin* variable Z. Set $w'(Z) = w'(\neg Z) = 1$ and for all other variables V, set $w'(V) = w(V)$ and $w'(\neg V) = w(\neg V)$. Construct Δ' by replacing the expression ϕ in Δ by the variable Z, and appending the equivalence $Z \Leftrightarrow \phi$.

Split the Equivalence Rewrite equivalence $Z \Leftrightarrow \phi$ as two implications, $Z \Rightarrow \phi$ and $Z \Leftarrow \phi$. In clausal form, these become

$$\begin{aligned} \neg Z \vee \quad & \phi \\ Z \vee & \neg\phi. \end{aligned}$$

Convert to a Feature Introduce a new *Relaxation variable* R. Set $w'(R) = 1$ and $w'(R) = 0$ and replace the sentence $\neg Z \vee \phi$ by

$$R \Leftrightarrow \neg Z \vee \phi.$$

In all models of the resulting theory where $\neg Z \vee \phi$ is not satisfied R has to be false, which means that the weight of those models is multiplied by 0. The weight of all other models remains the same.

Convert to an Implication Set $w'(R) = -1$ and turn the equivalence $R \Leftrightarrow \neg Z \vee \phi$ into an implication $R \Leftarrow \neg Z \vee \phi$, which in clausal form becomes

$$\begin{aligned} R \vee &\quad Z \\ R \vee &\neg\phi. \end{aligned}$$

Replacing the equivalence by an implication and changing $w'(R)$ to -1 is correct for the following reason. Let $R \Leftrightarrow \Sigma$ be the above equivalence which is in Δ, and let Γ represent all other sentences in Δ (i.e., $\Delta \equiv (\Sigma \Leftrightarrow R) \wedge \Gamma$). Our goal is now to construct a tuple (Δ', w'), where $\Delta' \equiv (\Sigma \Rightarrow R) \wedge \Gamma$, such that $WMC(\Delta, w) = WMC(\Delta', w')$.

A case analysis on the values of Σ and R shows that $WMC(\Delta, w)$ and $WMC(\Delta', w')$ consist of

Σ	R	$WMC(\Delta, w)$
1	1	$w(r) \cdot WMC(\Gamma \wedge \sigma, w)$
1	0	0
0	1	0
0	0	$w(\neg r) \cdot WMC(\Gamma \wedge \neg\sigma, w)$

Σ	R	$WMC(\Delta', w')$
1	1	$w'(r) \cdot WMC(\Gamma \wedge \sigma, w')$
1	0	0
0	1	$w'(r) \cdot WMC(\Gamma \wedge \neg\sigma, w')$
0	0	$w'(\neg r) \cdot WMC(\Gamma \wedge \neg\sigma, w')$

Note that $w(\neg r) = 0$ in the encoding of Δ, and thus

$$\begin{aligned} WMC(\Delta, w) &= w(r) \cdot WMC(\Gamma \wedge \sigma, w) \\ &\quad + 0 \cdot WMC(\Gamma \wedge \neg\sigma, w) \\ WMC(\Delta', w') &= w'(r) \cdot WMC(\Gamma \wedge \sigma, w') \\ &\quad + [w'(r) + w'(\neg r)] \cdot WMC(\Gamma \wedge \neg\sigma, w') \end{aligned}$$

Setting $w'(V) = w(V)$ for all variables V except for R ensures that $WMC(\Gamma \wedge \sigma, w) = WMC(\Gamma \wedge \sigma, w')$, that $WMC(\Gamma \wedge \neg\sigma, w) = WMC(\Gamma \wedge \neg\sigma, w')$. Furthermore, set $w(r) = w'(r)$. What remains for $WMC(\Delta, w)$ to equal $WMC(\Delta', w')$ is that $w'(r) + w'(\neg r) = w(r) + w'(\neg r) = 0$, which is achieved by setting $w'(r) = -w(r) = -1$.

References

Buchman, D., and Poole, D. 2016. Negation without negation in probabilistic logic programming. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Chavira, M., and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artif. Intell.* 172(6-7):772–799.

Chavira, M.; Darwiche, A.; and Jaeger, M. 2006. Compiling relational bayesian networks for exact inference. *International Journal of Approximate Reasoning* 42(1):4–20.

Darwiche, A. 2003. A Differential Approach to Inference in Bayesian Networks. *Journal of the ACM* 50:280–305.

Darwiche, A. 2004. New advances in compiling CNF to decomposable negation normal form. In *Proceedings of ECAI*, 328–332.

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

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. ProbLog: A Probabilistic Prolog and its Application in Link Discovery. In *20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2468–2473.

Díez, F. J., and Galán, S. F. 2003. Efficient computation for the noisy MAX. *International Journal of Intelligent Systems* 18(2):165–177.

Fierens, D.; Van den Broeck, G.; Thon, I.; Gutmann, B.; and De Raedt, L. 2015. Inference in probabilistic logic programs using weighted CNFs. *Theory and Practice of Logic Programming* 15.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2009. Model Counting. In *Handbook of Satisfiability*. 633–654.

Jaeger, M. 1997. Relational Bayesian networks. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI)*.

Jha, A., and Suciu, D. 2012. Probabilistic databases with MarkoViews. *Proceedings of the VLDB Endowment* 5(11):1160–1171.

Li, W.; Poupart, P.; and van Beek, P. 2011. Exploiting Structure in Weighted Model Counting Approaches to Probabilistic Inference. *Journal of Artificial Intelligence Research* 40:729–765.

Meert, W.; Struyf, J.; and Blockeel, H. 2010. Contextual variable elimination with overlapping contexts. In *Proceedings of the 5th European workshop on Probabilistic Graphical Models (PGM)*, 193–210.

Oztok, U., and Darwiche, A. 2015. A top-down compiler for sentential decision diagrams. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)*.

Poole, D. 2008. The independent choice logic and beyond. *Probabilistic inductive logic programming* LNAI 4911:222–243.

Sato, T. 1995. A Statistical Learning Method for Logic Programs with Distribution Semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP)*, 715–729.

Takikawa, M., and D'Ambrosio, B. 1999. Multiplicative factorization of noisy-max. In Laskey, K. B., and Prade, H., eds., *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI)*, 622–630.

Tseitin, G. 1970. On the complexity of derivation in propositional calculus. In Slisenko, A., ed., *Studies in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics*, 115–125. Steklov Mathematical Institute. Translated from Russian: Zapiski Nauchnykh Seminarov LOMI 8 (1968), pp. 234–259.

Van den Broeck, G.; Meert, W.; and Darwiche, A. 2014. Skolemization for weighted first-order model counting. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Vlasselaer, J.; Renkens, J.; Van den Broeck, G.; and De Raedt, L. 2014. Compiling probabilistic logic programs into sentential decision diagrams. In *Workshop on Probabilistic Logic Programming (PLP)*.

Vomlel, J., and Savicky, P. 2008. Arithmetic circuits of the noisy-or models. In *Proceedings of the Fourth European Workshop on Probabilistic Graphical Models (PGM 2008)*, 297–304.