Smoothing Structured Decomposable Circuits

Andy Shih
University of California, Los Angeles
andyshih@cs.ucla.edu

Guy Van den Broeck
University of California, Los Angeles
guyvdb@cs.ucla.edu

Paul Beame
University of Washington
beame@cs.washington.edu

Antoine Amarilli
LTCI, Télécom ParisTech
antoine.amarilli@telecom-paristech.fr

Abstract

We study the task of *smoothing* a circuit, i.e., ensuring that all children of a $\oplus$-gate mention the same variables. Circuits serve as the building blocks of state-of-the-art inference algorithms on discrete probabilistic graphical models and probabilistic programs. They are also important for discrete density estimation algorithms. Many of these tasks require the input circuit to be smooth. However, smoothing has not been studied in its own right yet, and only a trivial quadratic algorithm is known. This paper studies efficient smoothing for structured decomposable circuits. We propose a near-linear time algorithm for this task and explore lower bounds for smoothing general circuits, using existing results on range-sum queries. Further, for the important special case of All-Marginals, we show a more efficient linear-time algorithm. We validate experimentally the performance of our methods.

1 Introduction

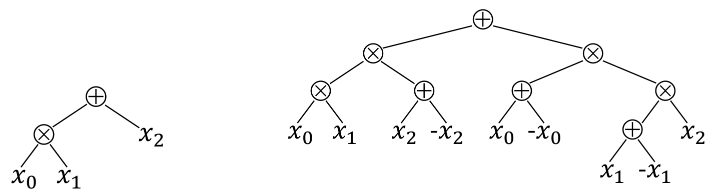
Circuits are directed acyclic graphs that are used throughout logical and probabilistic inference. Their structure captures the computation of reasoning algorithms. In the context of machine learning, state-of-the-art algorithms for exact and approximate inference in discrete probabilistic graphical models [Chavira and Darwiche, 2008; Kisa *et al.*, 2014; Friedman and Van den Broeck, 2018] and probabilistic programs [Fierens *et al.*, 2015; Bellodi and Riguzzi, 2013] are built on circuit compilation. As well, learning tractable circuits is the current method of choice for discrete density estimation [Gens and Domingos, 2013; Rooshenas and Lowd, 2014; Vergari *et al.*, 2015; Liang *et al.*, 2017]. Circuits are also used to enforce logical constraints on deep neural networks [Xu *et al.*, 2018].

Most of the probabilistic inference algorithms on circuits actually require the input circuit to be *smooth* (also referred to as *complete*) [Sang *et al.*, 2005; Poon and Domingos, 2011]. The notion of smoothness was first introduced by Darwiche [2001] to ensure efficient model counting and cardinality minimization and has since been identified as essential to probabilistic inference algorithms. Yet, to the best of our knowledge, no efficient algorithm to smooth a circuit has been proposed beyond the original quadratic algorithm by Darwiche [2001].

The quadratic complexity can be a major bottleneck, since circuits in practice often have hundreds of thousands of edges when learned, and millions of edges when compiled from graphical models. As such, in the latest Dagstuhl Seminar on "Recent Trends in Knowledge Compilation", this task of smoothing a circuit was identified as a major research challenge [Darwiche *et al.*, 2017]. Therefore, a more efficient smoothing algorithm will increase the scalability of circuit-based inference algorithms.

Intuitively, smoothing a circuit amounts to filling in the missing variables under its $\oplus$-gates. In Figure 1a we see that the $\oplus$-gate does not mention the same variables on its left side and right side, so we fill in the missing variables by adding tautological gates of the form $x_i \oplus -x_i$, resulting in the

Preprint. Under review.



(a) A circuit. (b) A smooth circuit.

Figure 1: Two equivalent circuits computing $(x_0 \otimes x_1) \oplus x_2$. The left one is not smooth and the right one is smooth.

smooth circuit in Figure 1b. Filling in these missing variables is necessary for probabilistic inference tasks such as computing marginals, computing probability of evidence, sampling, and approximating Maximum A Posteriori [Sang *et al.*, 2005; Chavira and Darwiche, 2008; Friesen and Domingos, 2016; Friedman and Van den Broeck, 2018; Mei *et al.*, 2018]. The task of smoothing was also explored by Peharz *et al.* [2017], where they look into preserving smoothness when augmenting Sum-Product Networks for computing Most Probable Explanation.

In this paper we propose a more efficient smoothing algorithm. We focus on the commonly used class of *structured decomposable circuits*, which include structured decomposable Negation Normal Form, Sentential Decision Diagrams, and more [Pipatsrisawat and Darwiche, 2008; Darwiche, 2011]. Intuitively, such circuits must always consider their variables in a certain way, which is formalized as a tree structure on the variables called a *vtree*.

Our first contribution (Section 4) is to show a near-linear time algorithm for smoothing such circuits, which is a clear improvement on the naive quadratic algorithm. Specifically, our algorithm runs in time proportional to the circuit size multiplied by the inverse Ackermann function α of the circuit size and number of variables[1] (Theorem 3).

Our second contribution (Section 5) is to show a lower bound of the same complexity, on smoothing general circuits for the restricted class of smoothing algorithms that we call *smoothing-gate algorithms* (Theorem 5). Intuitively, smoothing-gate algorithms are those that retain the structure of the original circuit and can only make them smooth by adding new gates to cover the missing variables. This natural class corresponds to the example in Figure 1 and our near-linear time smoothing algorithm also falls in this class. We match its complexity and show a lower bound on the performance of *any* smoothing-gate algorithm, relying on known results in the field of range-sum queries.

Our third contribution (Section 6) is to focus on the probabilistic inference task of All-Marginals and to propose a novel linear time algorithm for this task which bypasses the need for smoothing, assuming that the weight function supports all four elementary operations of $\oplus, \ominus, \otimes, \oslash$ (Theorem 6). These results are summarized in Table 1.

Our fourth contribution (Section 7) is to study how to make a circuit smooth while preserving structuredness. We show that we cannot achieve a sub-quadratic smoothing algorithm if we impose the same vtree structure on the output circuit (Prop. 7), unless the vtree has low height (Prop. 8).

Our final contribution (Section 8) is to experiment on smoothing and probabilistic inference tasks. We evaluate the performances of our smoothing and of our linear time All-Marginals algorithm.

The rest of the paper is structured as follows. In Section 2 we review the necessary definitions, and in Section 3 we motivate the task of smoothing in more detail. We then present each of our five contributions in order in Sections 4, 5, 6, 7 and 8. We conclude in Section 9.

2 Background

Let us now define the model of circuits that we study (refer again to Figure 1 for an example):

[1] The inverse Ackermann function α is defined in Tarjan [1972]. As the Ackermann function grows faster than any primitive recursive function, the function α grows slower than the inverse of any primitive recursive function, e.g., slower than $\log(\log(\log n)))$.

Table 1: Summary of results. We let n be the number of variables and m be the size of the circuit.

Task	Operations	Complexity
Smoothing	$\oplus, \otimes$	$O(m \cdot \alpha(m, n))$
Smoothing*	$\oplus, \otimes$	$\Omega(m \cdot \alpha(m, n))^*$
All-Marginal	$\oplus, \ominus, \otimes, \oslash$	$\Theta(m)$

* For *smoothing-gate algorithms* on general circuits

Definition 1. *A **logical circuit** is a rooted directed acyclic graph where leaves are variables, and internal gates perform disjunction ($\oplus$-gates) or conjunction ($\otimes$-gates). An **arithmetic circuit** is one where leaves are numeric constants or variables, and internal gates perform addition ($\oplus$-gates) or multiplication ($\otimes$-gates). The **children** of an internal gate are the gates that feed into it. We impose that the circuit has been preprocessed in linear time to ensure that each $\otimes$-gate has 0 or 2 inputs.*

We focus on circuits that are *decomposable*, and more precisely which are *structured*.

Definition 2. *For any gate p, we call $vars_p$ the set of variables that appears at or below gate p. A circuit is **decomposable** if these sets of variables are disjoint between the two children of every $\otimes$-gate. Formally, for every $\otimes$-gate p with children c_1 and c_2, we have $vars_{c_1} \cap vars_{c_2} = \emptyset$.*

We then define structuredness, by introducing the notion of vtree on a set of variables:

Definition 3. *A **vtree** on a set of variables S is a full binary tree whose leaves have a one-to-one correspondence with variables in S. We denote the set of variables under a vtree node p as u_p.*

Definition 4. *A circuit **respects** a vtree V if there is a mapping ρ from its gates to V such that:*

- *For every variable c, the node $\rho(c)$ is mapped to the leaf of V corresponding to c.*
- *For every $\oplus$-gate c and child c' of c, the node $\rho(c')$ is $\rho(c)$ or a descendant of $\rho(c)$ in V.*
- *For every $\otimes$-gate c with children c_1, c_2, letting v_l and v_r be the left and right children of $\rho(c)$, the node $\rho(c_1)$ is v_l or a descendant of v_l and $\rho(c_2)$ is v_r or a descendant of v_r.*

*A circuit is **structured decomposable** if it respects some vtree V. The circuit is also decomposable.*

Structured decomposability was introduced in the context of logical circuits, and it is also enforced in Sentential Decision Diagrams, a widely used tractable representation of Boolean functions [Darwiche, 2011]. This property allows for a polytime conjoin operation on logical circuits [Pipatsrisawat and Darwiche, 2008]. For circuits that represent distributions, structured decomposability allows multiplication of these distributions [Shen *et al.*, 2016] and efficient computation of the KL-divergence between two distributions [Liang and Van den Broeck, 2017].

Next, we review another property of circuits that will be relevant for probabilistic inference tasks [Choi and Darwiche, 2017].

Definition 5. *A circuit is **deterministic** if the children of each $\oplus$-gate are pairwise logically disjoint.*

In the rest of this paper, we will let n denote the number of variables in a circuit and let $m \geq n$ denote the size of a circuit, measured by the number of edges in the circuit.

3 Smoothing

We focus on the probabilistic inference tasks of weighted model counting and computing All-Marginals [Sang *et al.*, 2005; Chavira and Darwiche, 2008]. We will refer to weighted model counting as its more general form of Algebraic Model Counting (AMC) [Kimmig *et al.*, 2016]. To describe these tasks, we define knowledge bases and models.

Definition 6. *Given a set of variables $\mathbf{X}$, a set f of instantiations of $\mathbf{X}$ is called a **knowledge base**, and each element of f is called a **model**.*

The task of AMC on a knowledge base f and a weight function w is to compute s from Equation 1. The task of All-Marginals is to compute the partial derivative of s with respect to the weight of each variable. The weights are usually defined over a semiring, an important distinction we highlight later.

On probabilistic models, s is often the partition function or the probability of evidence, where the partial derivatives of these quantities correspond to all (conditional) marginals in the distribution. Computing All-Marginals efficiently significantly speeds up probabilistic inference, and is used as a subroutine in the collapsed compilation algorithm in our later experiments.

$$s = \bigoplus_{\mathbf{x} \in f} \bigotimes_{x \in \mathbf{x}} w(x) \quad \text{AMC} \quad (1) \qquad\qquad \left\{ \frac{\partial s}{\partial w(x)}, \frac{\partial s}{\partial w(-x)} \,\middle|\, X \in \mathbf{X} \right\} \quad \text{All-Marginals} \quad (2)$$

These tasks are difficult in general, unless we have a tractable representation of the knowledge base f. The following fact highlights the importance of smoothing. If f is represented as a logical circuit that is only deterministic and decomposable but not smooth, then there is in general no known technique to compute the AMC and All-Marginals tasks in linear time. If f is represented as a logical circuit that is deterministic, decomposable and smooth, then the AMC and All-Marginals tasks can be completed in time $O(m)$. For example, the AMC task is done by converting the deterministic, decomposable and smooth logical circuit into an arithmetic circuit, attaching the weights of the variables as numeric constants in the circuit, and then evaluating the circuit.

Given the necessity of smoothness for efficiently computing these inference tasks, we are interested in studying the complexity of smoothing a circuit. To do so, we formally define the task of smoothing.

Definition 7. *Two logical circuits on variables $\mathbf{X}$ are **equivalent** if they evaluate to the same output on any input $\mathbf{x}$.*

Definition 8. *A circuit is **smooth** if for every pair of children c_1 and c_2 of a $\oplus$-gate, $vars_{c_1} = vars_{c_2}$. The task of **smoothing** a logical circuit g is to output a smooth logical circuit that is equivalent to g.*

Note that we are only defining the smoothing task over logical circuits. This is because the probabilistic inference tasks are performed by smoothing a logical circuit and then converting it into an arithmetic circuit, so it is easier for the reader to only consider smoothing on logical circuits. For the rest of the paper, we will refer to logical circuits simply as circuits.

When the weight function allows division, there exists a renormalization technique that can compute the AMC in linear time without smoothing the initial circuit [Kimmig *et al.*, 2016]. However, this restriction is limiting, since even if the weight function is defined over a field, division by zero may raise an issue. For example, in practice division by zero may be unavoidable [Van den Broeck *et al.*, 2014] or the weight function may be defined over a semiring [Friesen and Domingos, 2016], in which case there is no known technique to bypass smoothing. As such, we explore efficient smoothing algorithms in Sections 4 & 5.

On the other hand, one may still be interested in settings where all four elementary operations of $\oplus, \ominus, \otimes, \oslash$ on the weight function are allowed. To this end, we also propose in Section 6 a novel technique that computes All-Marginals in linear time in this relaxed setting.

4 Smoothing Algorithm

We present our algorithm on smoothing structured decomposable circuits, based off of semigroup range-sum literature. First, we define a class of common strategies to smoothing a circuit, which encompasses both the previously-known algorithm and our new algorithm.

The existing quadratic algorithm on smoothing a circuit goes to each $\oplus$-gate and inserts missing variables one by one [Darwiche, 2001]. This algorithm retains the original gates of the circuit, and adds additional gates to fill in missing variables. We will define *smoothing-gate algorithms* as the family of smoothing algorithms that retain the original gates of the circuit.

Definition 9. ***Edge contraction** is the process of removing each $\oplus$-gate or $\otimes$-gate with a single child, and feeding the child as input to the parents of the removed gate.*

Definition 10. *Two circuits g and h with gate sets G and H are **isomorphic** if there exists a bijection $B : G \to H$ between their gates such that the following conditions hold.*

1. *For any gate $p \in G$, $B(p)$ is the same type of gate as p.*
2. *For any gate $p_1 \in G$ and child $p_2 \in G$ of p_1, the gate $B(p_2)$ is a child of $B(p_1)$ in h.*
3. *For any gate $p'_1 \in H$ and child $p'_2 \in H$ of p'_1, the gate $B^{-1}(p'_2)$ is a child of $B^{-1}(p'_1)$ in g.*

4. *The root of g maps to the root of h.*

*An algorithm is a **smoothing-gate algorithm** if for any edge-contracted input circuit g, the output circuit h has a subcircuit that is isomorphic to g after edge contraction.*

Definition 11. *A circuit g is called a **smoothing gate** if it is equivalent to the circuit $\bigotimes_{\mathbf{X}} (x \oplus -x)$ from some $\mathbf{X}$.*

Smoothing-gate algorithms are very intuitive, since the entire task boils down to the efficient computation of a smoothing gate $SG(\mathbf{X})$ given a set of missing variables $\mathbf{X}$. The structure of $SG(\mathbf{X})$ is not specified, and the only requirement is that it is equivalent to $\bigotimes_{\mathbf{X}} (x \oplus -x)$. The quadratic algorithm constructs $SG(\mathbf{X})$ by naively conjoining each variable in $\mathbf{X}$ one at a time, leading to a linear amount of work per gate. In the case of structured decomposable circuits, we can do much better.

Lemma 1. *Consider a structured decomposable circuit, and let π be the sequence of its variables written following the in-order traversal of its vtree. For any two vtree nodes $(\rho(p), \rho(c))$, we have that $u_{\rho(p)} \backslash u_{\rho(c)}$ can be written as the union of at most two contiguous intervals in π.*

Proof. Since v is a binary tree, the in-order traversal of v visits the variables of $u_{\rho(p)}$ consecutively, and the variables of $u_{\rho(c)}$ consecutively. Hence, $u_{\rho(p)}$ and $u_{\rho(c)}$ can each be written as a contiguous interval, and $u_{\rho(p)} \backslash u_{\rho(c)}$ can be written as the union of at most two contiguous intervals. □

We smooth a circuit in one bottom-up pass. If p is a leaf $\otimes$-gate, replace it with $SG(u_{\rho(p)})$. If p is an internal $\otimes$-gate, letting v_l, v_r and c_1, c_2 be the children of $\rho(p)$ and p respectively, replace c_1 with $c_1 \otimes SG(u_{v_l} \backslash u_{\rho(c_1)})$ and c_2 with $c_2 \otimes SG(u_{v_r} \backslash u_{\rho(c_2)})$. If p is a $\oplus$-gate, replace each child c with $c \otimes SG(u_{\rho(p)} \backslash u_{\rho(c)})$. By Lemma 1, each SG can be built with two gates of the form $\bigotimes_{\mathbf{X}}(x \oplus -x)$, where $\mathbf{X}$ is a continuous interval in π. Thus, we can appeal to results from semigroup range-sums.

Semigroup Range-Sum. The semigroup range-sum problem considers n variables, m intervals $[a_1, b_1], \dots, [a_m, b_m]$, and a weight function z over the variables. The task is to compute the sum of the weights of the variables in each interval, i.e. $s_j = \Sigma_{i \in [a_j, b_j]} z(x_i)$ for all $j \in [1, m]$ [Yao, 1982; Chazelle and Rosenberg, 1989]. Since z is only defined over a semigroup, subtraction is not supported. That is, we cannot follow the efficient strategy of precomputing $p_k = \Sigma_{i \in [1,k]} z(x_i)$ and outputting $s_j = p_{b_j} - p_{a_j - 1}$. Still, there is an efficient algorithm to compute all the required sums in time $O(m \cdot \alpha(m, n))$, where α is the inverse Ackermann function. We restate their result here.

Theorem 2. *Given n variables defined over a semigroup and m intervals, the sum of each interval can be computed in time $O(m \cdot \alpha(m, n))$ [Chazelle and Rosenberg, 1989].*

Our smoothing task can be reduced to the semigroup range-sum problem as follows. Smoothing a structured decomposable circuit of size m reduces to constructing smoothing gates for $O(m)$ intervals. We pass these intervals as input to the range-sums algorithm, which will then generate a sequence of additions that computes the sum of each interval. Each addition in the sequence will add two previously pre-computed sums.

We trace this sequence of additions (see Figure 2). For the base case of $z(x_i)$, let $g(z(x_i))$ be the gate $x_i \oplus -x_i$. Then for each addition $s = t + u$, we construct a corresponding $\otimes$-gate $g(s) = g(t) \otimes g(u)$. A sum of an interval then maps to a gate that is a smoothing gate for that interval. This process of smoothing a structured decomposable circuit leads to the following theorem.

Theorem 3. *The task of smoothing a structured decomposable circuit has time complexity $O(m \cdot \alpha(m, n))$, where n is the number of variables and m is the size of the circuit.*

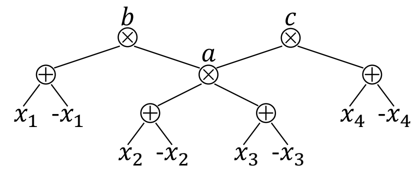
5 Lower Bound

In this section we show a lower bound on the task of smoothing a general circuit, for the family of smoothing-gate algorithms. Chazelle and Rosenberg [1989] show a lower bound on semigroup range-sums, as we state here, but more work is needed to successfully leverage their results.

Theorem 4. *Given n variables defined over a semigroup, there exists a set of $m = n$ intervals of the weights, such that computing the sum of each interval takes $\Omega(m \cdot \alpha(m, n))$ number of additions Chazelle and Rosenberg [1989].*

```
a = z(x_2) + z(x_3)
b = a + z(x_1)
c = a + z(x_4)
output b, c
```

(a) Sequence of additions to compute intervals



(b) Tracing the additions into a circuit

Figure 2: We construct smoothing gates for $\{x_1, x_2, x_3\}$ and $\{x_2, x_3, x_4\}$ by first passing the intervals $[1, 3]$ and $[2, 4]$ to the range-sum algorithm, and then tracing the sequence of additions. The trace is done by replacing $z(x_i)$ with $x_i \oplus -x_i$ and replacing each addition with a $\otimes$-gate.

We cannot immediately assert the same lower bound for smoothing general circuits, for two reasons. First, we must pose the sum of each of the m intervals as a smoothing problem in $O(m)$ time. Second, we must show that no smoothing algorithm is more efficient than smoothing-gates algorithms. We address the first issue, but leave the second open.

Theorem 5. *For the class of smoothing-gate algorithms, the task of smoothing a general circuit has space complexity $\Omega(m \cdot \alpha(m, n))$, where n is the number of variables and m is the size of the circuit.*

Proof. Take any set of m intervals, with $m = n$. For each interval $[a_i, b_i]$, construct the gate $G_i = \bigotimes_{j \notin [a_i, b_i]} x_j$, which is done by first constructing prefix gates $p_k = \bigotimes_{j=1}^{k} x_j, \forall k$ and suffix gates $s_k = \bigotimes_{j=k}^{n} x_j, \forall k$ in linear time, and then constructing each gate $G_i = p_{a_i - 1} \otimes s_{b_i + 1}$ in constant time. Next, let g be the circuit $G_1 \oplus \ldots \oplus G_m \oplus F$, where $F = \bigotimes_{j=1}^{n} x_j$. We need to show that running a smoothing-gate algorithm on g is as hard as computing the sum of each interval in m.

Since g has a top-level $\oplus$-gate with children $G_1, \ldots, G_m, F$, and F mentions all n variables, each gate G_i also needs to mention all n variables to satisfy smoothness. By the construction of G_i, it is missing exactly the variables $\mathbf{X}_i = [X_{a_i}, \ldots, X_{b_i}]$. We will show that constructing the smoothing gates $SG(\mathbf{X}_i)$ for all i is as hard as solving the semigroup range-sum problem on those intervals, by mapping the $\oplus$-operation in the semigroup range-sum problem to the $\otimes$-gates in our circuits.

In particular, consider a smooth circuit h that contains the smoothing gates $SG(\mathbf{X}_i)$ for all i. We use the following relabelling scheme to remove all the $\oplus$-gates. For every $\oplus$-gate p of h, take one of its input wires and reroute a copy of it to each gate that p feeds into. Each remaining $\otimes$-gate is now the product of all the variables that was mentioned by its corresponding gate in the original circuit h, so each gate $SG(\mathbf{X}_i)$ implicitly contained a $\otimes$-gate of the variables $\mathbf{X}_i$. This relabelling scheme shows that every $SG(\mathbf{X}_i)$ must implicitly be computing the $\otimes$-gate of $\mathbf{X}_i$. By setting the inputs to the circuits to be the value of the weights in the range-sum problem, and evaluating the circuits treating $\otimes$ as addition, the value to which each $SG(\mathbf{X}_i)$ gate evaluates is the requested sum. So, the circuit describes a sequence of additions to compute the sum of each interval. We then apply Theorem 4, which implies that the bound of $\Omega(m \cdot \alpha(m, n))$ applies to the size of the smooth circuit h. $\square$

6 Computing All-Marginals

In this section we propose an optimization to the special case of computing All-Marginals on a deterministic and structured decomposable circuit. The goal is to compute the partial derivative of the AMC with respect to the weight of each variable (Equation 2 in Section 3). Recall that computing All-Marginals on a deterministic, decomposable and smooth circuit takes time linear in the size of the circuit. Therefore, using the techniques in Section 4, we can smooth a deterministic and structured decomposable circuit and then convert it into an arithmetic circuit to compute All-Marginals, all in time $O(m \cdot \alpha(m, n))$. For the relaxed setting where the weight function also supports division and subtraction, we propose an even more efficient method to compute All-Marginals that bypasses the smoothing process. Our method takes time $O(m)$, which is not only optimal but also avoids the messy construction of smoothing gates.

The algorithm is a form of backpropagation, and goes as follows (Algorithm 1). First, we compute the AMC using a linear bottom-up pass over the circuit. During this process, we keep track of the AMC of each internal gate. Next, we traverse the circuit top-down in order to compute the partial derivative

Algorithm 1 `all-marginal`(g, w)

We compute partial derivatives of positive literals. The negative literals are handled similarly.

input: A deterministic and structured decomposable circuit g on n variables and a weight function w that supports $\oplus, \ominus, \otimes, \oslash$.

output: Partial derivatives s_j for $1 \leq j \leq n$.

`top-down`(g, w, amc)**:**

$pd \leftarrow \{\}$ // partial derivative
for gates p in g, parents before children **do**
  if p is leaf **then** $s_p \leftarrow pd[p]$
  if p is $\otimes$-gate with children C **then**
    for child k **do**
      $m \leftarrow (\bigotimes_C amc[c]) \otimes pd[p]$
      $pd[k] \leftarrow m \oslash amc[k]$
  if p is $\oplus$-gate with children C **then**
    for child k **do**
      $l_1, r_1, l_2, r_2 \leftarrow$ `getinterval`(p, k)
      $\delta_{l_1} \leftarrow \delta_{l_1} \oplus pd[p]$
      $\delta_{r_1+1} \leftarrow \delta_{r_1+1} \ominus pd[p]$
      $\delta_{l_2} \leftarrow \delta_{l_2} \oplus pd[p]$
      $\delta_{r_2+1} \leftarrow \delta_{r_2+1} \ominus pd[p]$
      $pd[k] \leftarrow pd[p]$
return s, δ

`main`(g, w)**:**

$amc \leftarrow$ `bottom-up`(g, w)
$s, \delta \leftarrow$ `top-down`(g, w, amc)
$s_1 \leftarrow s_1 \oplus \delta_1$
for $i \leftarrow [2, n]$ **do** $s_j \leftarrow s_{j-1} \oplus d_j$
return s

of each gate. At a $\otimes$-gate or $\oplus$-gate, we propagate the partial derivative down to the children as needed. However, since the circuit is not smooth, there may be missing variables in the children of $\oplus$-gates, in which case the propagation is incomplete. The challenge is to efficiently complete the propagation to the missing variables.

Theorem 6. *The All-Marginals task on a deterministic and structured decomposable circuit g and a weight function w that supports $\oplus, \ominus, \otimes, \oslash$ has time complexity $\Theta(m)$, where n is the number of variables and m is the size of g.*

Proof. Recall from Lemma 1 that the set of missing variables of each parent-child pair forms at most two contiguous intervals with respect to the in-order traversal of the vtree. The idea now is that propagating the partial derivative to each interval amounts to a *range increment*, i.e., incrementing a quantity for each variable in the interval. The naive algorithm takes quadratic time to do this for all intervals, but there is a more efficient method to perform all range increments in linear time.

Consider an integer n, a set of m intervals $[a_1, b_1], \ldots, [a_m, b_m]$ ($1 \leq a_i \leq b_i \leq n$), and m numeric constants $c_1, \ldots, c_m$. For each integer $1 \leq j \leq n$, we wish to compute the sum $s_j = \bigoplus_{i:j \in [a_i, b_i]} c_i$. That is, if j belongs to some interval $[a_i, b_i]$, then we increase s_j by c_i. The trick is to keep track of delta variables $\delta_1, \ldots, \delta_n$. For each interval $[a_i, b_i]$, we increase δ_{a_i} by c_i and decrease $\delta_{b_{i+1}}$ by c_i. Finally, we output $s_1 = \delta_1$ and $s_j = s_{j-1} \oplus \delta_j, j > 1$. This process can be done in time $O(m)$. $\square$

7 On Retaining Structuredness

The property of structured decomposability allows for a polytime conjoin operation, multiplication of distributions, and more (see Section 2). For downstream tasks such as computing AMC or All-Marginals, structuredness is not required. Since these downstream tasks are performed after the conjoin/multiply operations, our smoothing algorithm does not sacrifice much, if at all, by losing structuredness. One could also keep a copy of the original circuit if structuredness is needed later on.

Nevertheless, the reason our smoothing algorithm does not retain structuredness is that it interferes with the efficient construction of smoothing gates (Definition 11). In fact, we can show that any smoothing algorithm that maintains the same vtree structure must run in quadratic time.

Proposition 7. *The task of smoothing a structured decomposable circuit g that enforces the same vtree has space complexity $\Omega(nm)$, where n is the number of variables and m is the size of g.*

Proof. We consider a right-linear vtree v with variables $X_1, \ldots, X_n$, in that order. For simplicity, let n be a multiple of 3, and consider the following functions for $y \in [0, 2^{n/3})$:

$$J_y = \bigotimes_{i=1}^{n/3} \beta(i, y) x_i \qquad K_y = \bigotimes_{i=2n/3+1}^{n} \beta(i, y) x_i$$

where $\beta(i, y) = 1$ if the i-th bit of the binary representation of y is set, and -1 otherwise.

Next, consider $f = (\bigotimes_{i=1}^{n} -x_i) \oplus (\bigoplus_{i=1}^{2^{n/3-1}} (J_i \otimes K_i))$. An instantiation satisfies f if all its literals are negative, or if the sign of its literals from $X_1, \ldots, X_{n/3}$ (in order) equals those from $X_{2n/3+1}, \ldots, X_n$, and are not all negative. We can build a circuit g with size $O(2^{n/3})$ that respects v and computes f using an Ordered Binary Decision Diagram representation [Bryant, 1986]. Yet, any smooth circuit h that respects v and computes f has size $\Omega(n \cdot 2^{n/3})$, as we see next.

Let the depth of an internal gate c be $p+1$, where p is the length of the path from the root of v to $\rho(c)$. We use the notion of a *certificate* on a circuit, as defined by Bova *et al.* [2014]. Since h is smooth, every certificate of h must have n literals. Let a be an instantiation satisfying $J_i \otimes K_i$ and certificate T_a, and let b be an instantiation satisfying $J_j \otimes K_j$ and certificate T_b, with $i \neq j$ and $i, j \in [1, 2^{n/3})$. Since any instantiation satisfying $J_k \otimes K_l$ for $k \neq l$ is not a model of f, it follows that T_a and T_b must not share any internal gates from depth $n/3 + 1$ to depth $2n/3$. So, h has size $\Omega(n \cdot 2^{n/3})$. $\square$

In some cases, it is possible to do better: for instance, when the vtree has low height.

Proposition 8. *The task of smoothing a structured decomposable circuit g that enforces the same vtree v has time complexity $O(hm)$, where h is the height of the vtree and m is the size of g.*

Proof. We construct smoothing gates by following the structure of the vtree: for each vtree node p with children p_l and p_r, we build in constant time a structured smoothing gate for the variables that are descendants of p, using the smoothing gate for the variables that are descendants of p_l the one for the variables that are descendants of p_r. Now, we can use these gates to smooth the circuit: any interval of variables in the in-order traversal of the vtree can be written as h intervals corresponding to vtree nodes, so smoothing g has time complexity $O(hm)$. $\square$

8 Experiments

We experiment on our smoothing algorithm in Section 4 and our All-Marginals algorithm in Section 6. Experiments were run on a single Intel(R) Core(TM) i7-3770 CPU with 16GB of RAM.

Smoothing Circuits. We first study the smoothing task on structured decomposable circuits using our new smoothing algorithm (Section 4), which we compare to the naive quadratic smoothing algorithm. We construct hand-crafted circuits for which many smoothing gates are required, each of which covers a large interval. In particular, we pick m large intervals $I_1, ..., I_m$ and for each interval we construct the structured gate $G_i = \bigotimes_{j \notin I_i} x_j$ for a balanced vtree. Then we take each G_i and feed them into one top-level $\oplus$-gate. This triggers the worst-case quadratic behavior of the naive smoothing algorithm, while our new algorithm has near-linear behavior.

The speedup of our smoothing algorithm is captured in Table 2a. The **Size** column reports the size of the circuit. The **Naive** column reports the time taken by the quadratic smoothing algorithm, the **Ours** column reports the same value using our near-linear algorithm, and the **Improve** column reports the relative decrease in time. The values are averaged over 4 runs.

Collapsed Sampling. We next benchmark our method for computing All-Marginals in Section 6 on the task of collapsed sampling, which is a technique for probabilistic inference on factor graphs. The collapsed sampling algorithm performs approximate inference on factor graphs by alternating between *knowledge compilation phases* and *sampling phases* [Friedman and Van den Broeck, 2018]. In the sampling phase, the algorithm computes All-Marginals as a subroutine.

We replace the original quadratic All-Marginals subroutine by our linear time algorithm (Algorithm 1). The requirement of the $\oplus, \ominus, \otimes, \oslash$ operations for Algorithm 1 is satisfied since the weight function w is defined over the reals and $w(x) + w(-x) \neq 0$ in the experiments by Friedman and Van den Broeck [2018]. In Table 2b we report the results on the *Segmentation-11* network. Results for other networks used in Friedman and Van den Broeck [2018] were similar. We see a decrease in the number of $\oplus, \ominus, \otimes, \oslash$ operations needed for each All-Marginal computation. The **Size** column reports the size threshold during the knowledge compilation phase. The **Naive** column reports the number of $\oplus, \ominus, \otimes, \oslash$ operations using the original All-Marginals subroutine, the **Ours** column reports the same

Table 2: Left: Experiments on smoothing hand-crafted circuits. Right: Experiments on computing All-Marginals as part of the collapsed sampling algorithm. Sizes are reported in thousands (k).

(a) Time (in seconds) taken to smooth circuits.

Size	Naive	Ours	Improve %
40k	0.82 ± 0.01	0.04 ± 0.01	95.2 ± 1.3
416k	50 ± 0.3	0.31 ± 0.01	99.4 ± 0.1
1,620k	293 ± 2	0.74 ± 0.04	99.7 ± 0.1
8,500k	6050 ± 20	4.13 ± 0.09	99.9 ± 0.1

(b) Number of $\oplus, \ominus, \otimes, \oslash$ operations to compute All-Marginals when sampling the Segmentation-11 network.

Size	Naive	Ours	Improve %
100k	$28{,}494 \pm 598$	$20{,}207 \pm 411$	29.1 ± 3.0
200k	$55{,}875 \pm 1{,}198$	$36{,}101 \pm 1{,}522$	35.4 ± 5.2
400k	$86{,}886 \pm 6{,}330$	$56{,}094 \pm 817$	35.4 ± 6.1

value using Algorithm 1, and the **Improve** column reports the relative decrease in operations. The values are averaged over 4 runs.

9 Conclusion

In this paper we consider the task of smoothing a circuit. Circuits are widely used for inference algorithms for discrete probabilistic graphical models, and for discrete density estimation. The input circuits are required to be smooth for many of these probabilistic inference tasks, such as Algebraic Model Counting and All-Marginals. We provide a near-linear time smoothing algorithm for structured decomposable circuits and prove a matching lower bound within the class of smoothing-gate algorithms, for general circuits. We introduce a technique to compute All-Marginals in linear time without smoothing the circuit, when the weight function supports division and subtraction. As well, we show that smoothing a circuit while maintaining the same vtree structure cannot be sub-quadratic, unless the vtree has low height. Finally, we empirically evaluate our algorithms and show a speedup over the existing smoothing algorithm.

References

Elena Bellodi and Fabrizio Riguzzi. Expectation maximization over binary decision diagrams for probabilistic logic programs. *Intell. Data Anal.*, 17:343–363, 2013.

Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Expander cnfs have exponential dnnf size. *CoRR*, abs/1411.1995, 2014.

Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172:772–799, 2008.

Bernard Chazelle and Burton Rosenberg. Computing partial sums in multidimensional arrays. In *Symposium on Computational Geometry*, 1989.

Arthur Choi and Adnan Darwiche. On relaxing determinism in arithmetic circuits. In *ICML*, 2017.

Adnan Darwiche, Pierre Marquis, Dan Suciu, and Stefan Szeider. Recent trends in knowledge compilation (Dagstuhl Seminar 17381). *Dagstuhl Reports*, 7:62–85, 2017.

Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11:11–34, 2001.

Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI*, 2011.

Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Sht. Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *TPLP*, 15:358–401, 2015.

Tal Friedman and Guy Van den Broeck. Approximate knowledge compilation by online collapsed importance sampling. In *NeurIPS*, December 2018.

Abram L. Friesen and Pedro M. Domingos. The sum-product theorem: A foundation for learning tractable models. In *ICML*, 2016.

Robert Gens and Pedro M. Domingos. Learning the structure of sum-product networks. In *ICML*, 2013.

Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. Algebraic model counting. *International Journal of Applied Logic*, November 2016.

Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *KR*, 2014.

Yitao Liang and Guy Van den Broeck. Towards compact interpretable models: Shrinking of learned probabilistic sentential decision diagrams. In *IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI)*, August 2017.

Yitao Liang, Jessa Bekker, and Guy Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *UAI*, 2017.

Jun Mei, Yong Jiang, and Kewei Tu. Maximum a posteriori inference in sum-product networks. In *AAAI*, 2018.

Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro M. Domingos. On the latent variable interpretation in sum-product networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:2030–2044, 2017.

Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *AAAI*, 2008.

Hoifung Poon and Pedro M. Domingos. Sum-product networks: A new deep architecture. *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690, 2011.

Amirmohammad Rooshenas and Daniel Lowd. Learning sum-product networks with direct and indirect variable interactions. In *ICML*, 2014.

Tian Sang, Paul Beame, and Henry A. Kautz. Performing bayesian inference by weighted model counting. In *AAAI*, 2005.

Yujia Shen, Arthur Choi, and Adnan Darwiche. Tractable operations for arithmetic circuits of probabilistic models. In *NIPS*, 2016.

Robert E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22:215–225, 1972.

Guy Van den Broeck, Wannes Meert, and Adnan Darwiche. Skolemization for weighted first-order model counting. In *KR*, 2014.

Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *ECML/PKDD*, 2015.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, 2018.

Andrew Chi-Chih Yao. Space-time tradeoff for answering range queries (extended abstract). In *STOC*, 1982.