Pseudo Polynomial-Time Top-k Algorithms for d-DNNF Circuits

#3103

Abstract

We are interested in computing k most preferred models of a given d-DNNF circuit C, where the preference relation is based on an algebraic structure called a monotone, totally ordered, semigroup $(K, \otimes, <)$. In our setting, every literal in C has a value in K and the value of an assignment is an element of K obtained by aggregating using $\otimes$ the values of the corresponding literals. We present an algorithm that computes k models of C among those having the largest values w.r.t. $<$, and show that this algorithm runs in time polynomial in k and in the size of C. We also present a pseudo polynomial-time algorithm for deriving the top-k values that can be reached, provided that an additional (but not very demanding) requirement on the semigroup is satisfied. Under the same assumption, we present a pseudo polynomial-time algorithm that transforms C into a d-DNNF circuit C' satisfied exactly by the models of C having a value among the top-k ones. Finally, focusing on the semigroup $(\mathbb{N}, +, <)$, we compare on a large number of instances the performances of our compilation-based algorithm for computing k top solutions with those of an algorithm tackling the same problem, but based on a partial weighted MaxSAT solver.

1 Introduction

In this paper, we are interested in *optimization problems under compiled constraints*. Roughly, the goal is to derive a subset of most preferred solutions among the feasible ones, where the set of feasible solutions is of combinatorial nature and represented implicitly as valid assignments, i.e., those truth assignments satisfying some given constraints. Such optimization questions are key issues in a number of applications about configuration, recommendation, and e-commerce (see e.g., [Chen and Feng, 2018; Jannach *et al.*, 2021; Ricci *et al.*, 2015; Khabbaz and Lakshmanan, 2011]).

Unlike the preference relation at hand that is user-specific, the set of constraints representing the valid assignments is typically independent of the user, so that it does not often change. In such a case, taking advantage of a knowledge compilation approach can be useful, since compiling the constraints during an offline phase may allow polynomial-time algorithms for optimization tasks, whilst getting a single optimal solution already is NP-hard when no assumptions are made on the representations of the constraints. The discrepancy between the two approaches is amplified when computing multiple best solutions. Indeed, in practice, the computation of several best solutions for a set of uncompiled constraints generally requires successive calls to an NP oracle (see e.g., [Jabbour *et al.*, 2013]), while one can expect polynomial-time algorithms for this task when the constraints have been compiled first.

When dealing with propositional constraints, the language of deterministic, Decomposable Negation Normal Form circuits (d-DNNF) [Darwiche, 2001a] appears as a valuable language for compiling constraints because it supports in polynomial time a number of queries and transformations that are NP-hard in general [Darwiche and Marquis, 2002]. Among them are queries and transformations about optimization. Thus, [Darwiche and Marquis, 2004] has shown how to derive in polynomial time a most preferred, yet feasible solution where the set of feasible solutions is the set of models of a given d-DNNF circuit and the value of a solution is the sum of the weights (numbers representing utilities or penalties) associated with the literals it satisfies. The authors have also presented a polynomial-time transformation that returns a d-DNNF circuit whose models are precisely the preferred, feasible solutions of the d-DNNF circuit one started with.

Such an approach has been extended to a much more general, algebraic model counting setting in [Kimmig *et al.*, 2017]. The extension that has been achieved is threefold: first, the value of a solution is not necessarily a number, but an element of an abstract set, the carrier K of an algebraic structure called a commutative semiring; then, the aggregation operator used to define the value of a solution is not restricted to summation, but can be any abstract binary operator $\otimes$ over K; finally, the authors take advantage of an additional aggregation operator, $\oplus$, which is not necessarily equal to max or min. The goal is to compute the algebraic model count of a given d-DNNF circuit C, defined as the aggregation using $\oplus$ of the values of all models of C, where the value of a solution is the aggregation using $\otimes$ of the values of the literals satisfied by the solution (such values are elements of K). Algebraic model counting generalizes a number of problems of interest, including satisfiability (SAT), (possibly weighted) model counting (#SAT-WMC), and probabilistic inference (PROB) (see Theorem 1 in [Kimmig *et al.*, 2017]).

Following [Kimmig *et al.*, 2017], we extend the approach to optimization under d-DNNF constraints considered in [Darwiche and Marquis, 2004] but our extension relies on a different perspective, as reflected by the queries and transformation we focus on. Whilst [Darwiche and Marquis, 2004] aims to compute a single, most preferred solution, i.e., a top-1 solution (and the corresponding value), we are interested in computing k *most preferred models* (alias top-k solutions) of a given d-DNNF circuit C, where k is a preset bound given by the user. The returned assignments must be valid (i.e., they correspond to feasible solutions) and their values must be among the largest possible ones, i.e., for any top-k assignment ω, there cannot exist k (or more) valid assignments having a value strictly greater than the one of ω.

Considering top-k solutions is important to handle situations when the user is not satisfied by the top-1 solution that is provided (maybe he/she would finally prefer another solution reaching the top value, or even a solution with value slightly smaller than the value of a top-1 solution). We are also interested in computing the k most preferred values, thus extending the issue of computing the top-1 value as considered in [Darwiche and Marquis, 2004]. Finally, we investigate the corresponding transformation problem.

As [Kimmig *et al.*, 2017], we consider a more general algebraic setting than the one in [Darwiche and Marquis, 2004] where, implicitly, $\otimes$ is the summation operator and K is the set of real numbers. We focus here on an algebraic structure called a *monotone, totally ordered, semigroup* $(K, \otimes, <)$. Notwithstanding the $\oplus$ operator (which is implicitly max in our case) the structure used is less demanding than commutative semirings; especially, in the general case, the existence of a neutral element for $\oplus = max$ (i.e., a least element in K w.r.t. $<$) that is an annihilator for $\otimes$ is not required. Given a mapping ν associating with every literal ℓ of C an element of K, the value $\nu(\omega)$ of an assignment ω is defined as $\nu(\omega) = \otimes_{\ell \in Var(C) | \omega \models \ell} \nu(\ell)$. A top-$k$ value of C given $(K, \otimes, <)$ and ν is then one of the k-largest values v of K w.r.t. $<$ such that $v = \nu(\omega)$ is the value of a valid assignment ω of C. A top-k solution of C given $(K, \otimes, <)$ and ν is a model ω of C such that there is strictly less than k valid assignments of C having a value strictly greater w.r.t. $<$ than $\nu(\omega)$. Note that the set of values met by a set of top-k solutions does not span the full set of top-k values in general (e.g., the values of all top-k solutions can be equal to the top-1 value).

Our contribution is as follows. We first present an algorithm that computes top-k solutions of C given $(K, \otimes, <)$ and ν and that runs in time polynomial in k and in the size of the d-DNNF circuit C. We then outline a pseudo polynomial-time algorithm for deriving the top-k values of C given $(K, \otimes, <)$ and ν, provided that an additional (but not very demanding) requirement on the semigroup $(K, \otimes, <)$, namely almost strict monotony, is satisfied. Under the same assumption, we also outline a pseudo polynomial-time algorithm that transforms C into a d-DNNF circuit C' satisfied exactly by the models of C having a value among the top-k values of C given $(K, \otimes, <)$ and ν. Whenever k is small enough so that it can be considered as bounded by a constant (which is a reasonable assumption in practice), each of our top-k algorithms runs in time linear in the size of the d-DNNF circuit C. Finally, focusing on the semigroup $(\mathbb{N}, +, <)$, we present the results of an empirical comparison of our compilation-based algorithm for computing top-k solutions with an algorithm tackling the same problem, but based on the partial weighted MaxSAT solver MaxHS [Davies and Bacchus, 2011; Davies, 2013; Davies and Bacchus, 2013a; Davies and Bacchus, 2013b]. The obtained results show that, in practice, taking advantage of the compilation-based algorithm makes sense for many instances.

The rest of the paper is organized as follows. We first give some preliminaries. Then we present our algorithm for computing k top solutions of a d-DNNF circuit and outline our algorithm for computing its top-k values. Our algorithm for achieving the top-k transformation of a d-DNNF circuit is outlined afterwards. The results of the empirical evaluation are then presented, before the concluding section. Proofs, the code of our algorithms, and the data used in the experiments are provided as a supplementary material.

2 Preliminaries

Let X be a set of propositional variables. The set of $\mathrm{Lit}(X)$ of *literals* over X is the union of X with the set of negated variables over X. An *assignment* ω is a mapping from X to $\{1, 0\}$. A *Boolean function* f over X is a mapping from the assignments over X to $\{1, 0\}$. An assignment ω such that $f(\omega) = 1$ is called a *valid* assignment, or a model of f. The set of valid assignments for f is noted $\mathrm{ValidA}(f)$.

d-DNNF **circuits.** Circuits are convenient representations of Boolean functions. A *deterministic, Decomposable Negation Normal Form (*d-DNNF*) circuit* [Darwiche, 2001a] is a directed acyclic graph (DAG) where internal nodes are labelled by connectives in $\{\wedge, \vee\}$ and leaves are labelled by literals from $\mathrm{Lit}(X)$ or Boolean constants. The two main properties of d-DNNF circuits are that the sets of variables appearing in the subcircuits of any $\wedge$ node are pairwise disjoint (decomposability) and the valid assignments of the subcircuits of any $\vee$ node are pairwise disjoint (determinism). Figure 1a gives an example of a d-DNNF circuit.

In the remainder of the paper, regarding the d-DNNF circuits that are considered as inputs, we make the following three assumptions:

- First, internal nodes are binary (each of them has two children).
- Second, d-DNNF circuits are smooth, i.e., the sets of variables associated with the two children of any $\vee$ node are identical.
- Finally, those d-DNNF circuits are reduced, in the sense that no leaf node labelled by a Boolean constant occurs in the circuit, unless the circuit is such a leaf (in which case the top-k problems trivialize).

Those three assumptions are computationally harmless: any d-DNNF circuit can be binarized in linear time (every internal N node having $m > 2$ children $N_1, \ldots, N_m$ can be replaced by a binary tree with $m-1$ internal nodes of the same type as N and $N_1, \ldots, N_m$ as children; and every internal node with a single child can be replaced by its child), smoothed in quadratic time [Darwiche, 2001b] (and even more efficiently for structured d-DNNF [Shih *et al.*, 2019]), and reduced in linear time (just applying the elementary rules of Boolean calculus).

Top-k problems. In order to present in formal terms the three main top-k computation problems over d-DNNF circuits C we are interested in, we first need to make precise the algebraic structure over which the values of the satisfying assignments of C are evaluated:

Definition 1. *A* monotone, totally ordered semigroup *is a triple $(K, \otimes, <)$ where K is a set that is totally ordered by $<$ (a strict, total ordering), $\otimes$ is a binary operator over K that is commutative, associative, and* monotone*, i.e., for any $p, q, r, s \in K$, if $p \leq q$ and $r \leq s$ then $p \otimes r \leq q \otimes s$ (where $x \leq y$ iff $x < y$ or $x = y$). The semigroup is* strictly monotone *iff it is monotone and for any $p, q, r, s \in K$, if $p \leq q$ and $r < s$ then $p \otimes r < q \otimes s$. $(K, \otimes, <)$ is said to have a* least absorptive element *a whenever a is the least element of K w.r.t. $<$ and a is absorptive for $\otimes$, i.e., $\forall x \in K, x \otimes a = a \otimes x = a$. $(K, \otimes, <)$ is said to be* almost strictly monotone *if either $(K, \otimes, <)$ is strictly monotone or K has a least absorptive element a and $(K \setminus \{a\}, \otimes, <)$ is strictly monotone.*

Clearly enough, whenever $(K, \otimes, <)$ has a least absorptive element a, a is neutral for $\oplus = max$. Furthermore, when $\otimes$ is monotone, it distributes over $\oplus = max$ (which is obviously commutative). Thus, in this case, provided that $\otimes$ has a neutral element n, $(K, max, \otimes, a, n)$ is a commutative semiring. However, the existence of such a neutral element n is not mandatory in our setting.

Here are two examples. $(\mathbb{R}, +, <)$ and $([0,1], \times, <)$ are monotone, totally ordered semigroups. In $(\mathbb{R}, +, <)$, the elements of $\mathbb{R}$ may denote utilities and in $([0,1], \times, <)$, the elements of $[0,1]$ may denote probabilities. It is easy to check that $(\mathbb{R}, +, <)$ is strictly monotone (just like its restriction $(\mathbb{N}, +, <)$), which implies that it does not have a least absorptive element, and that $([0,1], \times, <)$ has a least absorptive element (namely, 0) and it is almost strictly monotone.

Provided a monotone, totally ordered semigroup $(K, \otimes, <)$, evaluating an assignment ω over X requires to indicate how the literals from $\mathrm{Lit}(X)$ are interpreted in K. This calls for a notion of *value function* over X onto K:

Definition 2. *Given a set X of propositional variables and a monotone, totally ordered semigroup $(K, \otimes, <)$, a* value function *ν over X onto K is a mapping from the literals over X to K, assigning to each literal ℓ an element from K noted $\nu(\ell)$ and called the* value *of ℓ.*

When $\otimes$ is a binary operator over K, the value of a literal ℓ as given by a value function ν over X onto K can then be extended to the *value of an assignment* ω over X, defined as the $\otimes$-aggregation of the values of the literals (as given by ν) satisfied by ω (the order with which they are taken does not matter as soon as $\otimes$ is commutative and associative). This value is noted $\nu(\omega)$. We denote by $\mathrm{ValidV}(C)$ the subset of values from K that are reached by the valid assignments of C. Formally, $\mathrm{ValidV}(C) = \{\nu(\omega) \mid \omega \in \mathrm{ValidA}(C)\}$.

In the following, we assume that the representation of the values in K are bounded by a constant and the costs of computing $\otimes$ and of comparing elements of K using $<$ are constant-time operations.

We are now ready to formally define the three top-k problems we focus on:

Definition 3. *Let $(K, \otimes, <)$ be a monotone, totally ordered semigroup. Let X be a set of propositional variables. Let ν be a value function over X onto K. Let C be a Boolean circuit over X. The problem $\mathrm{TopVal}_k(\nu, C)$ consists in computing the set of the k largest values w.r.t. $<$ in $\mathrm{ValidV}(C)$. When $\mathrm{ValidV}(C)$ contains less than k elements, the set is defined as $\mathrm{ValidV}(C)$.*

Note that the set of top-k values of C given $(K, \otimes, <)$ and ν is unique since K is totally ordered by $<$.

To define the problem of generating top-k solutions of C given $(K, \otimes, <)$, one first lifts the notion of value of a model ω of C to the notion of value of a set S of models of C, as follows: the value of a set $S = \{\omega_1, \ldots, \omega_m\}$ of models of C is the list of values $(\nu(\omega_{\pi(1)}), \ldots, \nu(\omega_{\pi(m)}))$ where π is a permutation over $\{1, \ldots, m\}$ such that $\nu(\omega_{\pi(1)}) \geq \ldots \geq \nu(\omega_{\pi(m)})$. The values of such sets S can then be compared w.r.t. the lexicographic ordering $\succ$ induced by $>$.

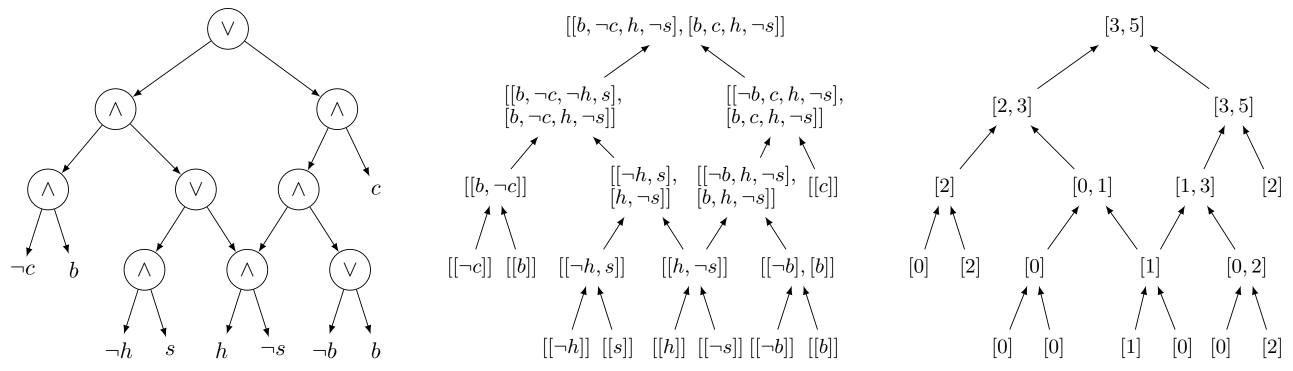
Definition 4. *Let $(K, \otimes, <)$ be a monotone, totally ordered semigroup. Let X be a set of propositional variables. Let ν be a value function over X onto K. Let C be a Boolean circuit over X. A set S of k models of C is a set of top-k models of C if and only if its value is the maximal value w.r.t. $\succ$ reached by sets of k models of C. As an exception, when $\mathrm{ValidA}(C)$ has less than k elements, the (unique) set of top-k models of C is defined as $\mathrm{ValidA}(C)$. Finally, the problem $\mathrm{TopSol}_k(\nu, C)$ consists in computing a set of top-k models of C.*

Unlike the set of top-k values, the set of top-k models of C is not unique in general (for instance, it may exist strictly more than k models ω of C having a maximal value $\nu(\omega)$).

Definition 5. *Let $(K, \otimes, <)$ be a monotone, totally ordered semigroup. Let X be a set of propositional variables. Let ν be a value function over X onto K. Let C be a Boolean circuit over X from a circuit language $\mathcal{L}$. The problem $\mathrm{TopTra}_k(\nu, C)$ consists in computing from C a circuit C' in the same language $\mathcal{L}$ as C and whose models are precisely those of $\mathrm{ValidA}(C)$ having a value in $\mathrm{TopVal}_k(\nu, C)$.*

Obviously enough, such a circuit C' is not unique in general. Let us now illustrate on a simple example the top-k problems we consider.

Example 1. *Consider the d-DNNF circuit C over $X = \{b, c, h, s\}$ given at Figure 1a. C encodes an E-Shop security system where at least one payment between* `bank transfer (b)` *and* `credit card (c)` *is required, exactly one security policy between* `high (h)` *and* `standard (s)` *must be chosen, with the constraint that* `c` *implies* `h`*. Suppose that ν gives value 2 to literal b and to literal c, value 1 to literal h, and value 0 to every other literal and that $(\mathbb{N}, +, <)$ is the*



a. The d-DNNF circuit C. b. $\mathrm{TopSol}_2(\nu, C)$. c. $\mathrm{TopVal}_2(\nu, C)$.

Figure 1: Top-2 algorithms at work on Example 1.

monotone, totally ordered semigroup under consideration. C has 4 models, reported in the following table, and for each of these models we indicate its value according to ν.

$(b, c, h, s) \in \mathrm{ValidA}(C)$	$\nu(b, c, h, s)$
$(0, 1, 1, 0)$	3
$(1, 0, 0, 1)$	2
$(1, 0, 1, 0)$	3
$(1, 1, 1, 0)$	5

For this example, the set of top-2 values is $\{5, 3\}$. There are two possible sets of top-2 solutions, namely $\{(1, 1, 1, 0), (0, 1, 1, 0)\}$ and $\{(1, 1, 1, 0), (1, 0, 1, 0)\}$. Any circuit in the d-DNNF *language over $X = \{b, c, h, s\}$ having as models $\{(1, 1, 1, 0), (0, 1, 1, 0), (1, 0, 1, 0)\}$ is an admissible result for the top-2 transformation of C.*

Algorithm 1: $\mathrm{TopSol}_k(\nu, N)$.

```
Input: $N$: a node in a d-DNNF circuit, $k$: a positive integer,
       $\nu$: the value function
Result: a list of top-$k$ solutions of the d-DNNF circuit
        rooted at $N$
if $N$ is a leaf node labelled by literal $\ell$ then
    return $[[\ell]]$;
else
    top_c0 = $\mathrm{TopSol}_k(\nu, N.children(0))$;
    top_c1 = $\mathrm{TopSol}_k(\nu, N.children(1))$;
    if $N$ is a $\vee$ node then
        return sorted_fusion(top_c0, top_c1, k, $\nu$);
    else  // $N$ is a $\wedge$ node
        return sorted_product(top_c0, top_c1, k, $\nu$);
    end
end
```

3 Computing Top-k Solutions

To solve $\mathrm{TopSol}_k(\nu, C)$, we present Algorithm 1. This algorithm computes in a bottom-up fashion the values of two synthesized attributes (representing top-k solutions and their values) for the d-DNNF (sub)circuits rooted at the nodes of the input circuit C. When N is an internal node of C, top_c0 (resp. top_c1) denotes the list of top-k solutions (and the corresponding values for ν) that has been computed for the d-DNNF circuit rooted at the left (resp. right) child of N. Depending on the label of N ($\wedge$ or $\vee$), a different procedure is run to derive the values of the two attributes at N: *sorted_fusion(top_c0, top_c1, k, ν)* when N is a $\vee$ node and *sorted_product(top_c0, top_c1, k, ν)* when N is a $\wedge$ node. When called at a $\vee$ node, *sorted_fusion* returns the ordered list of top-k solutions extracted from the sorted union of top_c0 and top_c1. When called at a $\wedge$ node, *sorted_product* returns an ordered list of top-k solutions generated from the cross product of top_c0 and top_c1.

Complexity. The complexity of Algorithm 1 relies on the two subroutines *sorted_fusion* and *sorted_product*. *sorted_fusion* merges two ordered lists (this can be done in linear time in the size of the lists). *sorted_product* returns the k greatest elements from the product of two ordered lists and this can be done in time $\mathcal{O}(m \cdot \log m)$ where m is the minimum between k and the number of assignments in the product.

These results assume a clever representation of the partial assignments by keeping in the ordered lists a pointer to the assignments (not the assignments themselves, otherwise the complexity would increase by a factor $|X|$). We have obtained the following theorem:

Theorem 1. *Let $(K, \otimes, <)$ be a monotone, totally ordered semigroup. Let X be a set of propositional variables. Let ν be a value function over X onto K. Let C be a* d-DNNF *circuit over X. The problem $\mathrm{TopSol}_k(\nu, C)$ can be solved in time $\mathcal{O}(\sum_{N\ \vee\text{-node}} m(N) + \sum_{N\ \wedge\text{-node}} m(N) \cdot \log\ m(N))$ where $m(N)$ is the minimum between the number of valuations at C_N and k, where C_N is the subcircuit rooted at N.*

Example 2. *Let us consider Example 1 again. The nodes N of the DAG reported in Figure 1b correspond in a bijective way with those of the* d-DNNF *circuit in Figure 1a (the nodes and the arcs of the two DAGs are the same ones, only the labels change). The label of each node N of the DAG at Figure 1b is a list of top-2 solutions of the* d-DNNF *circuit in Figure 1a rooted at the same node (for the sake of readability, the values of those solutions are not reported on the figure).*

Performance of our algorithm. In this paragraph, we do not present any strong lower bound to the complexity of TopSol_k. However, we would like to mention that TopSol_k is closely related to the problem of $X+Y$ *sorting*, which consists in sorting the pairs of elements of two sets X and Y following the sum of the scores of the elements. The exact complexity of this problem is still an open question and the best known algorithm to solve it is in $\mathcal{O}(n^2 \cdot \log\ n)$, where n is the maximum of $|X|$ and $|Y|$. Interestingly, it is possible to build a `d-DNNF` circuit $C_{X,Y}$ such that each valuation represents a pair of elements in X and Y and the associated score is the sum of the scores of the two elements. From $C_{X,Y}$ and the associated score function $\nu_{X,Y}$, the problem of $X+Y$ sorting can be reduced to the problem TopSol_{n^2}. By looking at the construction of the circuit, we can deduce that Algorithm 1 solves the $X+Y$ sorting problem in $\mathcal{O}(n^2 \cdot \log\ n)$, which the best known complexity.

4 Computing Top-k Values

From the user perspective, computing top-k solutions requires first to decide which value of k should be retained. To make an informed choice, deriving first top-k values (with possibly another value for k than the one representing the number of solutions) can be very useful.

To this end, we present an adaptation of Algorithm 1. The main modifications that took place are the following ones: the top-k values algorithm keeps an ordered list of k values at each node (and not of k assignments), and the functions sorted_fusion and sorted_product are modified to account for this update. sorted_fusion still sorts two ordered lists and its principle and complexity remain unchanged. Contrastingly, sorted_product cannot benefit from the trick used for generating top-k solutions but in the worst case, it needs to sort the k^2 pairs of values obtained from the two lists. This last part can be done by using any classical sort algorithm in time $\mathcal{O}(k^2 \cdot \log k)$. More precisely, we have obtained the following result:

Theorem 2. *Let $(K, \otimes, <)$ be an almost strictly monotone, totally ordered semigroup. Let X be a set of propositional variables. Let ν be a value function over X onto K. Let C be a `d-DNNF` circuit over X. The problem $\mathrm{TopVal}_k(\nu, C)$ can be solved in time $\mathcal{O}(\sum_{N\ \vee-\mathrm{node}} \min(v(c_0(N)) + v(c_1(N)), k) + \sum_{N\ \wedge-\mathrm{node}} \min(v(c_0(N)) \cdot v(c_1(N)), k^2) \cdot \log\ (\min(v(c_0(N)) \cdot v(c_1(N)), k^2)))$ where $c_0(N)$ (resp. $c_1(N)$) is the left (resp. right) child of N, and $v(N)$ is equal to $\min(k, \mathrm{ValidV}(C_N))$ and C_N is the subcircuit rooted at N.*

Example 3. *Let us step back to Example 1 once more. The nodes N of the DAG reported in Figure 1c correspond in a bijective way with those of the `d-DNNF` circuit in Figure 1a. The label of each node N of the DAG reported in Figure 1c is the list of top-2 values of the `d-DNNF` circuit in Figure 1a rooted at the corresponding node.*

5 Achieving Top-k Transformation

Once the top-k values have been computed, other operations can be done on the assignments with a value among the top-k ones. In order to make them efficiently, we are looking to represent this set of assignments using a `d-DNNF` circuit.

This calls for a top-k transformation algorithm. We now give a brief intuition of this algorithm. Our algorithm for $\mathrm{TopTra}_k(\nu, C)$ starts by running $\mathrm{TopVal}_k(\nu, C)$ so that the top-k values of the `d-DNNF` circuits rooted at the nodes N of C are stored as additional labels of the corresponding nodes. The generation of a `d-DNNF` circuit C' as a result of $\mathrm{TopTra}_k(\nu, C)$ is achieved by parsing the nodes of C (together with the list of values associated with them) in a bottom-up manner.

For simplicity's sake, let us suppose first that $(K, \otimes, <)$ does not have a least absorptive element. For each node N of C and each value v in its top-k list, a new subcircuit noted (N, v) is generated in order to connect all the children (S, w) contributing to the value v in N, where S is a child of n and w is one of its value. For $\vee$-nodes N, (N, v) is the disjunction of the nodes (S, v) and for $\wedge$-nodes, (N, v) is the disjunction of the conjunctions of the nodes (S, w) and (T, u) where S and T are children of N such that $w \otimes u = v$.

If $(K, \otimes, <)$ has a least absorptive element a, then $\wedge$-nodes N must be treated differently whenever a appears in the top-k values of N. In such a case, the subcircuit (N, a) that is generated is the disjunction of the subcircuits (S, a) with (T, u) where S and T are the children of N. Indeed, $a \otimes u$ is equal to a for any value u.

Once the root N of C has been processed, the resulting circuit is simplified in linear time by removing every node and every arc that cannot be reached from the root, and by shunting every node that has a single child.

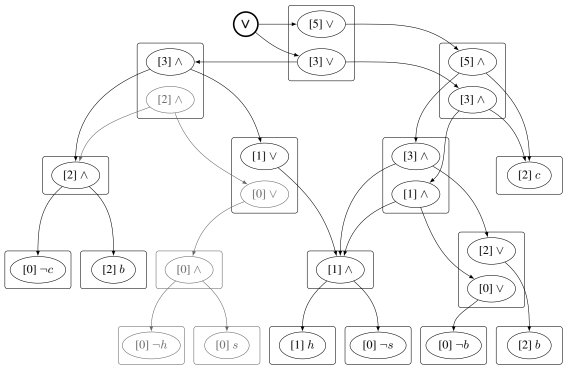
We have obtained the following theorem:

Theorem 3. *Let $(K, \otimes, <)$ be an almost strictly monotone, totally ordered semigroup. Let X be a set of propositional variables. Let ν be a value function over X onto K. Let C be a `d-DNNF` circuit over X. The problem $\mathrm{TopTra}_k(\nu, C)$ can be solved in time $\mathcal{O}(|C| \cdot k^2 \cdot log\ k)$ and the resulting `d-DNNF` circuit has a size in $\mathcal{O}(|C| \cdot k^2)$.*

Example 4. *Let us consider again Example 1 once more. Figure 2a illustrates the computation achieved for deriving $\mathrm{TopTra}_2(\nu, C)$ where C is the `d-DNNF` circuit presented at Figure 1a. An equivalent, yet simplified circuit is reported on Figure 2b.*

One of the motivations of the top-k transformation is to enable various analyses about the top-k values to take place afterwards. Our transformation leads to a deterministic disjunction of subcircuits for each of the top-k values. Each of those circuits being a `d-DNNF` circuit, we can operate a number of tractable operations over them. For instance, we can count the number of assignments having a preset value among the top-k values.

Corollary 1. *Let $(K, \otimes, <)$ be an almost strictly monotone, totally ordered semigroup. Let X be a set of propositional variables. Let ν be a value function over X onto K. Let C be a `d-DNNF` circuit over X. For each value v in $\mathrm{TopVal}_k(\nu, C)$, the problem of counting the number of valid assignments of value v can be solved in $\mathcal{O}(|C| \cdot k^2 \cdot log\ k)$.*


a. $\mathrm{TopTra}_2(\nu, C)$ at work.

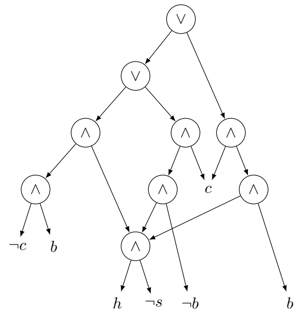

b. The resulting d−DNNF circuit (once simplified).

Figure 2: $\mathrm{TopTra}_2(\nu, C)$ at work on Example 1. Each box in Figure 2a gathers nodes related to the same node in the initial d−DNNF circuit C given at Figure 1a. Each node is labelled by the value given to it according to $\mathrm{TopVal}_2(\nu, C)$ and by a connective or a literal. Faded nodes and arcs are reported to illustrate the construction but they are not parts of the d−DNNF circuit that is generated. Figure 2b presents the resulting d−DNNF circuit (the circuit given at Figure 2a, once simplified).

6 Experimental Evaluation

Setup. We implemented in Java our algorithm for solving the TopSol_k problem. This implementation takes as input a d−DNNF circuit. Since constraints are usually given into the CNF format, we leveraged the state-of-the-art compiler d4[1] [Lagniez and Marquis, 2017] in order to generate a d−DNNF circuit (more precisely, a Decision-DNNF circuit) from a CNF formula. This circuit is then binarized, smoothed, and reduced upstream to its use as an input of our algorithm.

To the best of our knowledge, there does not exist any specific benchmark suited to the TopSol_k problem. Therefore, in order to experiment and validate our approach, we relied on a benchmark originating from a dataset reported in [Sharma *et al.*, 2018]. It contains 1436 CNF formulae encoding problems coming from various fields, including probabilistic reasoning, bounded model checking, circuit, product configuration, SMTLib benchmarks, planning, quantified information flow and bug synthesis. For each run, we created a value function ν for each CNF formula by assigning an integer between 0 and 1,000,000 to each literal of the formula. Those integers have been picked up at random following a uniform distribution. In our experiments, we mainly considered small values of k (less or equal to 20) since the generation of top-k solutions is typically triggered by a human user who will not be able to encompass a large set of solutions as a whole due to his/her cognitive limitations (see e.g., [Miller, 1956]). Nonetheless, we also considered a greater value of k (50) to assess our approach on a different practical scenario.

Each run was computed on a machine based on bi-processors Intel Xeon E5-2680 v4 (2.2 GHz) with 64 GB of memory with a timeout set to 10 minutes. The code of the algorithms and the data used in our experiments are furnished as a supplementary material.

Results. We now present the results obtained by the implementation of our TopSol_k algorithm with k set to 20 so as to showcase to which extent our approach is valuable in practice. d4 succeeded in compiling in due time 1175 instances out of 1436. The execution of the TopSol_{20} algorithm has been successful over 1162 instances out of 1175.

[1]www.cril.univ-artois.fr/KC/d4.html

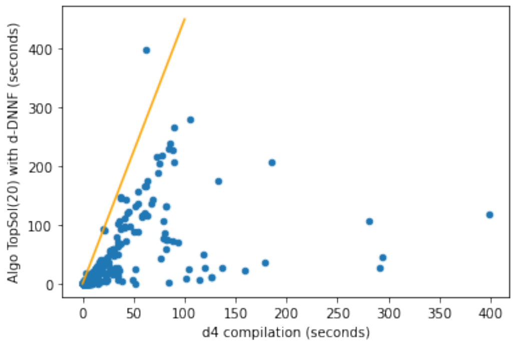


Figure 3: Comparison of the run time required by the compilation step and the average run time required to generate 20 top solutions.

Figures 3 and 4 present scatter plots that are useful for comparing in more detail (at the instance level) the run times required by the compilation step, the binarization-smoothing-reduction step, and finally the generation of 20 top solutions. Each dot corresponds to an input instance (a CNF formula). In Figure 3, the x-coordinate of each dot is the time (in seconds) required by d4 to compile it into a d−DNNF circuit, and its y-coordinate gives the average run time (over the five value functions) required to generate 20 top solutions provided that the d−DNNF circuit has been binarized, smoothed, and reduced. In Figure 4, the x-coordinate of each dot is the time (in seconds) required to binarize, smooth, and reduce the d−DNNF circuit associated with the instance, and its y-coordinate gives the average run time (over the five value functions) required to generate 20 top solutions from this circuit.

Figures 3 and 4 show that the main part of the overall computation time is typically required by the compilation step, but that the time used for smoothing and for finally generating top-k solutions cannot be neglected.

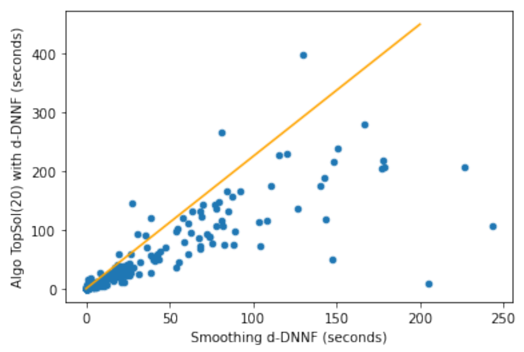


Figure 4: Comparison of the run time required by the binarization-smoothing-reduction step and the average run time required to generate k top solutions.

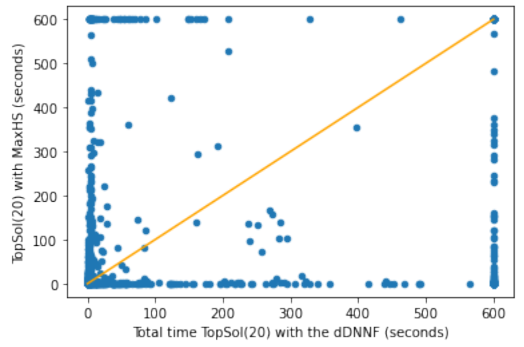


Figure 5: Comparison of the run times of our approach and the direct approach for computing 20 top solutions.

Comparison to a direct approach based on Weighted Partial MaxSAT. For the sake of comparison, we have also considered a *direct* approach to solve TopSol_k, i.e., an approach that does not include any compilation step. This approach consists in looking for a top-1 solution of the problem using a Weighted Partial MaxSAT solver. Whenever such a solution has been found and k solutions have not been enumerated so far, a hard clause is added to block the new solution (i.e., to prevent from generating it afterwards), and the algorithm resumes.

We have implemented such a direct approach in Python and leveraged one of the best solvers from the 2020 MaxSAT competition, namely MaxHS [Davies and Bacchus, 2013b; Davies and Bacchus, 2013a; Davies, 2013] in our implementation. We compared the two approaches on the full dataset (including instances that have not been compiled by d4 before the time limit) and set k to 1, 10, 20 and 50. A timeout of 10 minutes per instance has been considered for each approach (for our approach, it includes the compilation time). The empirical results are described in the following Table 1.

	k=1	k=10	k=20	k=50
Success rate for MaxHS	93.8%	89.6%	88.3%	85%
Success rate for the d-DNNF approach	79.7%	81%	80.6%	78.7%
# instances solved only by MaxHS	217	173	168	180
# instances solved only by the d-DNNF approach	15	49	58	90
# instances where MaxHS was faster	1194	844	596	395
# instances where the d-DNNF approach was faster	168	492	730	915

Table 1: Performances of MaxHS and of the d-DNNF approach for computing k top solutions, depending on k.

As reported in Table 1, the top-k approach based on MaxHS solved 93.8% of the instances of the dataset for $k = 1$ to 85% for $k = 50$, while the compilation-based approach solved between 78.7% to 81%. When k increased, the number of instances solved only by the MaxSAT-based approach diminished from 217 for $k = 1$ to 168 for $k = 20$, while the number of instances solved only by the d-DNNF approach went up from 15 to 90. As to run times, the fastest method was the MaxSAT-based approach for $k = 1$ and $k = 10$, but for $k = 20$ and $k = 50$, the compilation-based approach was clearly faster. This is not surprising since the main computational effort in the compilation-based approach is the time (and space) spent in the compilation phase, but this phase has to be performed once only, and the resources used are independent of the value of k. This contrasts with the MaxSAT-based approach to computing k top solutions which requires to solve k instances of an NP-hard problem (the instances being possibly harder and harder).

Accordingly, the compilation-based approach appears as the more interesting option to derive k top solutions when k is large enough. Our experiments show that even for a small value of k, it can be a challenging method.

To conclude, Figure 5 presents a scatter plot showing the run times required by the two approaches for finding the top-20 first solutions. Each dot corresponds to a CNF formula, its x-coordinate is the average run time required by our approach, and its y-coordinate is the average run time required by the direct approach. Based on this figure, the two approaches look rather complementary in the sense that for a large majority of instances the run times used by the two approaches are significantly different.

7 Conclusion

We have presented three top-k algorithms for d-DNNF circuits C given a totally ordered, semigroup $(K, \otimes, <)$. Provided that some assumptions about monotony w.r.t. $<$ are satisfied by $\otimes$, these algorithms can be used, respectively, to compute in pseudo polynomial-time k top solutions of C, the top-k values met by solutions of C, and a d-DNNF circuit C' satisfied exactly by the models of C having a value among the top-k ones. We have also presented the results of an empirical evaluation, showing that the d-DNNF compilation-based approach can prove valuable to address the top-k solutions problem.

Interestingly, it can be noted that some of the restrictions considered in the previous sections could be questioned. Our top-k algorithms could be extended to DNNF, i.e., removing the determinism condition on circuits C. This would not have any impact on the complexity of the algorithms, especially those for computing the top-k values and making the top-k transformation, and only a slight impact on the complexity of the algorithm for deriving k top solutions (equality tests should be implemented at $\vee$ nodes, thus one could not get rid of the $|X|$ factor in the complexity assessment).

References

[Chen and Feng, 2018] B. Chen and T. Feng. Top-k query for weighted interactive product configuration. In *2018 17th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, pages 326–331, Los Alamitos, CA, USA, oct 2018. IEEE Computer Society.

[Darwiche and Marquis, 2002] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[Darwiche and Marquis, 2004] Adnan Darwiche and Pierre Marquis. Compiling propositional weighted bases. *Artificial Intelligence*, 157(1-2):81–113, 2004.

[Darwiche, 2001a] Adnan Darwiche. Decomposable negation normal form. *Journal of the Association for Computing Machinery*, 48(4):608–647, 2001.

[Darwiche, 2001b] 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(1-2):11–34, 2001.

[Davies and Bacchus, 2011] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011.

[Davies and Bacchus, 2013a] Jessica Davies and Fahiem Bacchus. Exploiting the power of mip solvers in maxsat. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing*, SAT'13, page 166–181, Berlin, Heidelberg, 2013. Springer-Verlag.

[Davies and Bacchus, 2013b] Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up maxsat solving. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, pages 247–262, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Davies, 2013] Jessica Davies. *Solving MaxSAT by decoupling optimization and satisfaction*. PhD thesis, University of Toronto, 2013.

[Jabbour *et al.*, 2013] Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. The top-k frequent closed itemset mining using top-k SAT problem. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Zelezný, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III*, volume 8190 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2013.

[Jannach *et al.*, 2021] Dietmar Jannach, Pearl Pu, Francesco Ricci, and Markus Zanker. Recommender systems: Past, present, future. *AI Mag.*, 42(3):3–6, 2021.

[Khabbaz and Lakshmanan, 2011] Mohammad Khabbaz and Laks V. S. Lakshmanan. Toprecs: Top-k algorithms for item-based collaborative filtering. In Anastasia Ailamaki, Sihem Amer-Yahia, Jignesh M. Patel, Tore Risch, Pierre Senellart, and Julia Stoyanovich, editors, *EDBT 2011, 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21-24, 2011, Proceedings*, pages 213–224. ACM, 2011.

[Kimmig *et al.*, 2017] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. Algebraic model counting. *J. Appl. Log.*, 22:46–62, 2017.

[Lagniez and Marquis, 2017] Jean-Marie Lagniez and Pierre Marquis. An Improved Decision-DNNF Compiler. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 667–673, 2017.

[Miller, 1956] G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, 1956.

[Ricci *et al.*, 2015] Francesco Ricci, Lior Rokach, and Bracha Shapira, editors. *Recommender Systems Handbook*. Springer, 2015.

[Sharma *et al.*, 2018] Shubham Sharma, Rahul Gupta, Subhajit Roy, and Kuldeep S. Meel. Knowledge compilation meets uniform sampling. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 57 of *EPiC Series in Computing*, pages 620–636. EasyChair, 2018.

[Shih *et al.*, 2019] Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. Smoothing structured decomposable circuits. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 11412–11422, 2019.

1. Proof of Theorem 1

An algorithm for the top-k solutions problem. See Algorithm 1 in the paper.

Correctness and complexity. In order to prove the correctness of our bottom-up algorithm (Algorithm 1) for computing k top solutions of a $\texttt{d-DNNF}$ circuit C, one must show that, at any internal node N of C, computing k top solutions of the $\texttt{d-DNNF}$ circuit C_N rooted at N can be achieved when k top solutions of the $\texttt{d-DNNF}$ circuits $C_{c_0(N)}$ and $C_{c_1(N)}$, rooted respectively at the children N_0 and N_1 of N, have been computed first. Note that each of $C_{c_0(N)}$ and $C_{c_1(N)}$ has at least one model since C is supposed to be simplified.

The expected result is rather obvious when N is a $\vee$ node: in this case, by definition, the set of valid assignments of the circuit rooted at N is the union of the sets of valid assignments of the circuits rooted at $c_0(N)$ and at $c_1(N)$. Thus, to derive k top solutions of the $\texttt{d-DNNF}$ circuit rooted at N, it is enough to make the union of the sets of k top solutions associated with its two children, to sort them in decreasing order w.r.t. their values, and to keep the first k elements of the sorted list of solutions.

Things are a bit more tricky when considering $\wedge$ nodes N. In that case, the monotony assumption about $\otimes$ is useful. Towards a contradiction, suppose that the set $\{\omega \cdot \omega' : \omega \in \mathrm{TopSol}_k(\nu, C_{c_0(N)}), \omega \in \mathrm{TopSol}_k(\nu, C_{c_1(N)})\}$ does not include k top solutions of C_N. Thus, there exist $\omega \models C_{c_0(N)}$ and $\omega' \models C_{c_1(N)}$ such that $\omega \notin \mathrm{TopSol}_k(\nu, C_{c_0(N)}) = \{\omega_1, \ldots, \omega_k\}$ and for each $i \in [k]$, $\nu(\omega \cdot \omega') > \nu(\omega_i \cdot \omega'_1)$, where $\omega'_1 \in \mathrm{TopSol}_1(\nu, C_{c_1(N)})$. In such a case, since $\nu(\omega') \leq \nu(\omega'_1)$ and $\nu(\omega) \leq \nu(\omega_i)$ ($i \in [k]$), the monotony property of the semigroup ensures that for each $i \in [k]$, $\nu(\omega) \otimes \nu(\omega') \leq \nu(\omega_i) \otimes \nu(\omega'_1)$, or equivalently $\nu(\omega \cdot \omega') \leq \nu(\omega_i \cdot \omega'_1)$, a contradiction.

As to complexity, let us first consider a simple, yet naive implementation of the two functions sorted_fusion and sorted_product. Function sorted_fusion(top_c0, top_c1, k, ν) computes the sorted union of two disjoint lists of size k. It is well-known that this can be done in time $O(k \cdot \log\ k)$.

Function sorted_product(top_c0, top_c1, k, ν) can be implemented by computing explicitly the cross product of the two lists of k solutions, ordering them and picking up k top elements. Through this implementation, the algorithm runs in time $O(|C| \cdot k^2 \cdot \log\ k \cdot |X|)$.

However, a better implementation can be obtained by avoiding to sort at each $\wedge$ node N all the assignments resulting from the cross product of the two lists, top_c0 and top_c1. Our algorithm takes advantage of a max-heap implementation of a priority queue Q, i.e., a data structure allowing to add elements to Q and remove elements from Q in time logarithmic in the size of Q, and also to retrieve an element of maximal value from Q in constant time. In our implementation of sorted_product, one stores in Q pairs of indices (i, j) together with the corresponding value for ν denoted by $\nu(i, j)$. Each pair (i, j) represents the concatenation of the ith assignment of top_c0 with the jth assignment of top_c1. Because $(K, \otimes, <)$ is monotone, the first pair $(0, 0)$ of Q is one of top value. Then the following treatment is iterated for $k - 1$ steps. At each step, the first pair stored in Q, (i, j) is retrieved, then deleted from Q, and the corresponding assignment is added to the list of top-k solutions under construction for node N. Then the pairs $(i, j + 1)$ and $(i + 1, 0)$ are added to the queue if they were not added yet. We use additional structures to check these in a decent time (this test can be done in constant time using a hashmap or in time $\log\ k$ using B+ trees). By construction, after k steps, the list at node N contains k top assignments of the cross product between top_c0 and top_c1. Since at each step, one element is removed from Q and at most two elements are added, the size of Q increases linearly in the number of steps and therefore the size of Q remains linear in k. Therefore, the time complexity of computing k top assignments for a $\wedge$ node N is in $O(|X| \cdot (\sum_{N\ \wedge\text{-node}} m(N) \cdot \log\ m(N)))$ where $m(N)$ is the minimum between the number of valuations at C_N and k (with C_N the subcircuit rooted at N).

The $|X|$ factor in the complexity evaluation comes from the computational cost of concatenating the two assignments represented in a naive manner. It is possible to remove this factor through a more efficient representation of assignments. In our implementation, sets of literals are represented as binary trees where leaves are labelled by literals. Then, the concatenation of partial assignments at $\wedge$ nodes can be achieved in constant time by taking the roots of the two sets and creating a new root having them as children. The decomposability of $\wedge$ nodes ensures the correctness of the approach (i.e., the resulting tree is guaranteed to correspond to a partial assignment). At $\vee$ nodes, where unions of sets of partial assignments must be done, there is no need for equality tests to avoid duplicates: by construction, the determinism of $\vee$ nodes ensures that those unions are disjoint ones. As a consequence, the construction of solutions can be done efficiently through the tree representation of assignments and the multiplicative factor $|X|$ can be removed from the time complexity of our algorithm.

Finally, the time used by sorted_fusion(top_c0, top_c1, k, ν) can be improved given that top_c0 and top_c1 are sorted. It is well-known that sorting the fusion of two sorted lists can be done in time linear in the sum of the sizes of the lists. This gives the complexity expressed by Theorem 1.

2. Proof of Theorem 2

An algorithm for the top-k values problem. Our algorithm to solve $\mathrm{TopVal}_k(\nu, C)$ is a variant of Algorithm 1 for computing top-k solutions. A main difference is that it is sufficient to store values and thus, the procedures sorted_fusion and sorted_product take as inputs tables of values and not tables of pairs (assignment, value) and they output tables of values. Those procedures must be updated to handle (respectively) $\vee$ nodes and $\wedge$ nodes in a satisfying way since it is possible in the top-k values context that duplicates appear. When computing top-k solutions, the decomposability and the determinism conditions on $\texttt{d-DNNF}$ circuits ensure that the assignments generated at each node of C when applying Algorithm 1 are distinct, but it is not the case for values.

Thus, when calling sorted_fusion(top_c0, top_c1, k, ν), it may happen that the same value appears both in top_c0 and top_c1 so that one of the duplicates has to be removed after sorting. Thus the update of sorted_fusion(top_c0, top_c1, k, ν) simply consists in sorting the values of the union of top_c0 and top_c1 and then removing the duplicates from the resulting sorted table. This has no impact on the complexity of sorted_fusion. When calling sorted_product(top_c0, top_c1, k, ν) at a $\wedge$ node, it is also possible to get duplicates as distinct $\otimes$-combinations of the values of the solutions of its two children. The update of sorted_product(top_c0, top_c1, k, ν) can be done as follows. The values that are obtained are kept in memory, using a binary search tree S allowing us to add an element and to check whether an element is already stored in S in time logarithmic in the size of S. Whenever we pop a pair (i, j), we check using S whether $\nu(i, j)$ has already been outputted. $\nu(i, j)$ is then outputted and added to S iff it has not been outputted before. The remaining instructions of the algorithm, i.e., adding $(i + 1, 0)$ and $(i, j + 1)$ to Q, are the same ones as those in the algorithm for computing top-k solutions. This treatment is repeated until k distinct values have been found or we went through all the pairs (i, j).

Correctness and complexity. The main point for proving the correctness of our algorithm is the correctness of sorted_product. For this, we can prove that if a value v belongs to the top-k values of a subcircuit C_N rooted at a $\wedge$ node N with children $c_0(N)$ and $c_1(N)$, then either v is equal to the least absorptive element of K if it exists or there exist u in the top-k values of $c_0(N)$ and w in the top-k values of $c_1(N)$ such that $u \otimes w = v$. This property is a consequence of the fact that $(K, \otimes, <)$ is almost strictly monotone. More precisely, we consider two cases depending on the connective labelling the root N of the subcircuit C_N under consideration.

$\vee$ To find the top k elements from the union of two ordered tables containing k elements, it is sufficient to merge the two tables and to stop the merging process once k elements have been selected. This operation can be done in time $\mathcal{O}(k)$. Therefore, the computation of the k top values of a subcircuit rooted at a $\vee$ node N can be achieved in time $\mathcal{O}(k)$ by assuming the top-k values of its two subcircuits have already been computed.

$\wedge$ In the context, we prove by contradiction that is sufficient to compute the products of the pairs of the k top values of the two subcircuits rooted at the children of the $\wedge$ node N. Let assume that one of the k top values of the subcircuit rooted at N is obtained as the product of the ith top value v_i of one child of N with the jth top value w_j of the other child of N such that at least one of i or j is strictly greater than k. Suppose without loss of generality that $i > k$. Two cases must be considered: either w_j is the absorptive element of K for $\otimes$ or w_j is not the absorptive element of K for $\otimes$.

- If w_j is not the absorptive element of K for $\otimes$, then due to the fact that $(K, \otimes, <)$ is almost strictly monotone, all the values obtained by $v_l \otimes w_j$, $l < i$ are strictly greater than $v_i \otimes w_j$ and they are all distinct. Thus, $v_i \otimes w_j$ is not among the top-k values of C_N.

Now let us assume that w_j is the absorptive element of K for $\otimes$.

- If w_j is the top-1 value of the second child of N, then since $(K, \otimes, <)$ is almost strictly monotone, w_j is the least element of K, and as a consequence, w_j is the only top-k value associated with the second child of N. Therefore, w_j also is the only top-k value associated with N.
- If w_j is the not the top-1 value of the second child of N, then each combination $v_l \otimes w_{j-1}$ where $l \leq k$ is strictly greater than $v_l \otimes w_j$ and therefore greater than $v_i \otimes w_j$. Thus, $v_i \otimes w_j$ is not among the top-k values associated with N.

With this case analysis, we see that we can always restrict ourselves to compute the k top values for each subcircuit.

The time complexity of our algorithm comes from calling $|C|$ times the procedure sorted_fusion(top_c0, top_c1, k, ν) or the procedure sorted_product(top_c0, top_c1, k, ν). For sorted_fusion, the complexity bound is the same one as for the top-k solutions case. For sorted_product, one may need to consider the full set of pairs of values coming from the $\otimes$-combinations of the top-k values associated with the children of the $\wedge$ node at hand. Therefore, the time complexity of a call to this procedure is in $\mathcal{O}(\min(v(c_0(N)) \cdot v(c_1(N)), k^2) \cdot \log\ \min(v(c_0(N)) \cdot v(c_1(N)), k^2))$.

3. Proof of Theorem 3

An algorithm for the top-k transformation problem. Our algorithm for $\mathrm{TopTra}_k(\nu, C)$ starts by running $\mathrm{TopVal}_k(\nu, C)$ so that the top-k values of the d-DNNF circuits rooted at the nodes N of C are stored as additional labels of the corresponding nodes. The generation of a d-DNNF circuit C' as a result of $\mathrm{TopTra}_k(\nu, C)$ is achieved by parsing the nodes N of C (together with the $\mathrm{TopVal}_k(\nu, C_N)$) in a bottom-up manner. For the sake of simplicity, let us suppose first that $(K, \otimes, <)$ does not have a least absorptive element. Let $c_0(N)$ and $c_1(N)$ be the children of N when N is an internal node. The treatment is as follows. Every leaf node of C is kept unchanged. For each internal node N together with the associated list $\mathrm{TopVal}_k(\nu, C_N)$ of values, we create one new node for each value v in the list $\mathrm{TopVal}_k(\nu, C_N)$. The node created from N with value v is noted (N, v).

If N is a $\vee$ node, then (N, v) is a $\vee$ node, and for each value v, arcs connecting (N, v) to $(c_0(N), v)$ and/or $(c_1(N), v)$ are added if those last nodes exist, i.e., if v belongs to the list of the top-k values associated with $c_0(N)$ and/or $c_1(N)$. If the root N of C is a $\vee$ node, then the root of C' is a new $\vee$ node having as children the nodes (N, v) where v varies in $\mathrm{TopVal}_k(\nu, C_N)$. Now, for a $\wedge$ node N, for each value v in the list of the top-k values associated with N, let $L(v)$ be the list of pairs (u, w) of values from the lists of the top-k values associated respectively with $c_0(N)$ and $c_1(N)$, such that $u \otimes w = v$. We construct a subcircuit rooted at (N, v) that encodes the disjunction over the values v of the conjunctions of $(c_0(N), u)$ and $(c_1(N), w)$ such that $u \otimes w = v$.

Once the root of C has been processed, the resulting circuit is simplified in linear time by removing every node and every arc that cannot be reached from the root, and by shunting every node that has a single child.

When the semigroup has a least absorptive element a, the construction of C' is similar to the previous one, except when a belongs to the top-k values of a subcircuit rooted at a $\wedge$ node N. Indeed, in such a case, a also belongs to the top-k values associated with one of the children $c_0(N)$, $c_1(N)$ of N. Suppose that a is a top-k value associated with $c_0(N)$. Because a is absorptive, for any valid assignment ω of the subcircuit rooted at $c_1(N)$ such that $\nu(\omega) = v$ (whatever v is), we have $a \otimes v = a$. Therefore, ω can be among the valid assignments of the subcircuit rooted at $c_1(N)$ that produce (once concatenated with a valid assignment of the subcircuit rooted at $c_0(N)$) a top-k solution at node N. Accordingly, all the valid assignments ω at $c_1(N)$ must be stored, even if they are not among those having a top-k value at $c_1(N)$. To deal with this case, we copy C into a new circuit C^c so that for each node N of C we have an associated node in C^c, denoted N^c. The subcircuit associated with each node (N, v) where v is different of a is built as explained before. We now explain how to build the subcircuits associated with the nodes (N, a).

The construction depends on the type of N:

- if N is a $\vee$ node, then the subcircuit associated with (N, a) is equal to the disjunction of the nodes $(c_0(N), a)$ and $(c_1(N), a)$ if they exist. If there exists only one such node, (N, a) is equal to this node.
- if N is a $\wedge$ node, then there are three cases: if both $(c_0(N), a)$ and $(c_1(N), a)$ exist, then the subcircuit associated with (N, a) is the disjunction of the conjunction of $(c_0(N), a)$ and $c_1(N)^c$ with the conjunction of $c_0(N)^c$ and $(c_1(N), a)$; if only $(c_0(N), a)$ exists, then the subcircuit associated with (N, a) is the conjunction of $(c_0(N), a)$ and $c_1(N)^c$; in the remaining case, i.e., when $(c_1s(N), a)$ exists, then the subcircuit associated with (N, a) is the conjunction of $c_0(N)^c$ and $(c_1(N), a)$.

Correctness and complexity. By construction, the decomposability of the $\wedge$ nodes is preserved by the transformation algorithm. Furthermore, the $\vee$ nodes that are created in the resulting circuit are deterministic ones (each child of such a node corresponds to a set of satisfying assignments having a value different of those of its sibling or obtained by concatenating distinct partial assignments, therefore those sets of assignments are pairwise disjoint).

The main part of the correctness of the algorithm is to prove that for each $\wedge$-node N and v, one of its top-k value which is not absorptive, it is sufficient to consider the top-k values of the two children of N to consider all the possible manners to compute v.

We prove this property by contradiction. Let v be a top-k value associated with node N. Consider the set of pairs (v_{l_i}, w_{j_i}) of values of the children of N such that $v_{l_i} \otimes w_{j_i}$ is equal to v. The index l_i (respectively j_i) is the rank of the value v_{l_i} (respectively w_{j_i}) in the values associated with the subcircuits rooted at the children of N. Let assume towards a contradiction that there exists i such that l_i is greater than k. Due to the fact that $(K, \otimes, <)$ is almost strictly monotone, all the values obtained by $v_m \otimes w_{j_i}$, $m < l_i$ are strictly greater than $v_{l_i} \otimes w_{j_i}$ and they are all distinct. This contradicts the fact that $v_{l_i} \otimes w_{j_i}$ is one the top-k values associated with N. A similar reasoning can be done for the case when j_i is greater than k.

The multiplicative factor k^2 in the size of the resulting `d-DNNF` circuit comes directly from the fact that in the worst case, for any value v in $\mathrm{TopVal}_k(\nu, C_N)$, the number of pairs of values u and w in $\mathrm{TopVal}_k(\nu, C_{c_0(N)})$ and $\mathrm{TopVal}_k(\nu, C_{c_1(N)})$ such that $u \otimes w = v$ is equal to k^2.