mathematics

Article

A Deletion Algorithm for the Marginal Problem in Propositional Logic Based on Boolean Arrays

Efraín Díaz-Macías [1,*,†] **and Serafín Moral** [2,†]

1 Faculty of Engineering Sciences, State Technical University of Quevedo, Quevedo 120301, Ecuador
2 Department of Computer Science and Artificial Intelligence, University of Granada, 18071 Granada, Spain
* Correspondence: efraindiaz@uteq.edu.ec
† These authors contributed equally to this work.

Abstract: This paper proposes a deletion algorithm for the marginal problem in propositional logic. The algorithm is based on the general Davis and Putnam deletion algorithm DP, expressed as a bucket elimination algorithm, representing sets of clauses with the same set of variables employing a Boolean array. The main contribution is the development of alternative procedures when deleting a variable which allow more efficient computations. In particular, it takes advantage of the case in which the variable to delete is determined by a subset of the rest of the variables. It also provides a set of useful results and tools for reasoning with Boolean tables. The algorithms are implemented using Python and the `NumPy` library. Experiments show that this procedure is feasible for intermediate problems and for difficult problems from hard Bayesian networks cases.

Keywords: marginal problem; satisfiability problem; propositional logic; propagation algorithm; calculus with potentials

MSC: 03-08

check for updates

Citation: Díaz-Macías, E.; Moral, S. A Deletion Algorithm for the Marginal Problem in Propositional Logic Based on Boolean Arrays. *Mathematics* **2023**, *11*, 2748. https://doi.org/10.3390/math11122748

Academic Editor: Ioannis G. Tsoulos

Received: 16 May 2023
Revised: 14 June 2023
Accepted: 15 June 2023
Published: 17 June 2023

1. Introduction

The marginal problem consists of computing the consequences of a set of propositional formulae in a reduced subset of variables. The basic algorithm to solve it has been the so-called Davis and Putnam (DP) [1] deletion algorithm. This algorithm is a particular case of the Shenoy–Shafer deletion algorithm [2,3] or the bucket elimination scheme [4,5]. The problem of this algorithm is the space complexity (it tends to produce too many clauses of large size). The time complexity is exponential in the tree-width of its connectivity graph [4]. The computations can be organized in a join tree [6] in the same way as probabilistic computations in Bayesian networks [2,7]. However, in the case of propositional logic, we are in a case of computation in an idempotent valuation system or valuation algebra [3,8], which has some special features which can be exploited in order to build efficient algorithms.

This paper proposes new algorithms for the marginal problem by applying the Shenoy–Shafer abstract framework [2], but it has some differences compared to the DP algorithm. First, it represents sets of clauses as Boolean arrays. This is a semantic representation of the set of true assignments satisfying a given set of clauses, i.e., a truth table. Boolean arrays can be a very efficient representation of a large set of clauses, as the values on the array are simple 0–1 values, though its size is exponential in the number of variables. We define the basic operations and show that the set of Boolean arrays with these operations has the structure of an information algebra [3]. Secondly, we also give alternative deletion procedures and a set of possible optimizations of the full procedure. In this sense, a very important contribution is the study of the cases in which a variable is functionally determined by a set of variables and the exploitation of this fact in the marginalization algorithms. The Boolean array representation is especially appropriate for this improvement.

The final result is a family of tools and algorithms that can solve moderate-sized problems, even in cases where the associated connectivity graph has a large tree width (more than 100), as is shown in the experiments.

A related problem is the satisfiability problem (SAT) [9], which consists of determining if a set of propositional formulae is satisfiable, i.e., there is a true assignment such that every formula becomes true. It is the first problem proven to be NP-complete [10]. This implies that any NP problem can be efficiently reduced to it, and therefore, any good algorithm to solve the SAT could also be used for such an NP problem. In fact, many well-known problems are solved nowadays by encoding them as an SAT [11–13]. Current approaches for SATs are mainly based on the Davis–Putnam–Logemann–Loveland backtracking algorithm (DPLL) [14], and its successors, conflict-driven clause-learning algorithms (CDCL) [15]. The SAT problem can be solved from the marginal problem, by deleting all the variables (marginalizing over the empty set). The result is consistent if and only if this marginalization is vacuous. However, the marginal problem can be used without deleting all the variables for computing the marginal in a given set and it can be used to compute all the solutions (configurations satisfying all the clauses) or to simulate in the space of solutions, in the case of a satisfiable problem [6]. As is said in [5], the marginal problem is a kind of knowledge compilation that can be useful in many other related problems. Furthermore, in a recent survey of SAT-related problems [16], the following was noted: "We see that DP responds to a more difficult problem than the simple problem of satisfiability. Except for a few specific problems of small induced width, DP-60 never really competed with the version of DP-62" (where DP-60 is what we have called DP and DP-62 makes reference to DPLL). For this reason, the basic deletion algorithm has received little attention in the literature. For example, in Knuth's book [17], covering the existing approaches for SAT, there is only a short reference to the DP approach, saying that it works well for small problems, but that it can be very inefficient in the worst case. Our paper will show that it is possible to solve large problems, even problems with a large tree width. The initial DP algorithm was revisited in [4,5], but this approach was also based on a clause representation and the main contributions were the determination of good deletion sequences.

Another related problem is propositional model counting or #-SAT, consisting of computing the number of models satisfying a given formula [18]. This is related to probabilistic computations and it is known to be #P-complete. Similar deletion algorithms have been applied; however, as is said in Gogate and Dechter [19], these algorithms are exponential with the tree-width size. In this paper, we take advantage of using idempotent valuations and develop a computation method that can solve some concrete cases, even if the tree width is large, using representations that are not exponential with such a tree width.

We provide some experiments carried out with our `Python 3.8.8` implementation based on the `NumPy` library for representing arrays (the library together with data to reproduce experiments are available as a Github repository at: https://github.com/serafinmoral/SAT-solver (accessed on 16 May 2023)). In them, we show that it is possible to solve some moderate-sized problems, expanding the class of problems solved by the existing deletion algorithms. The contribution is also significant because of the possibilities it opens for future developments, as proposed in the last section of the paper devoted to the conclusions and future work. We have also shown that it is possible to solve the 0–1 problems associated with hard Bayesian networks inference problems. The rest of the paper is organized as follows: in Section 2, the notation and the problem specification are given; Section 3 is devoted to the table representation of sets of clauses, the introduction of the basic operations (combination, marginalization, and conditioning), and the study of their properties; Section 4 studies the basic deletion algorithm and the alternative procedures, providing formal results of their main properties; in Section 5, a set of additional tools for the deletion algorithm are given, and the final decision procedure for the deletion algorithm is described; Section 6 is devoted to the experimental part; and

Section 7 details the conclusions and future work. All the proofs of the results in the paper are in Appendix A.

2. Problem Specification

Let $V = \{p, q, r, s, \dots\}$ a finite set of variables or propositional symbols. A literal is either a variable, p, (positive literal) or a negated variable $\neg p$ (negative literal). A clause c is a disjunction of literals, $p \vee \neg r \vee s$, which we will represent as a set $c = \{p, \neg r, s\}$. If a clause contains literals p and $\neg p$, then it will be called trivial or tautology. The set of all non-trivial clauses defined for variables V is denoted as $\mathcal{L}(V)$.

Information will be given as a finite set of non-trivial clauses C. The set of variables appearing in clause c will be denoted by $V(c)$. If C is a set of clauses, then $V(C) = \bigcup_{c \in C} V(c)$.

A true assignment, t, is a mapping from V to $\{False, True\}$. A true assignment t satisfies the set of clauses C when for each $c \in C$, there is a positive literal, $p \in c$, with $t(p) = True$ or a negative literal, $\neg r \in c$, with $t(r) = False$.

Two sets of clauses, C_1 and C_2, defined for the same set of variables V, will be considered logically equivalent, $C_1 \equiv C_2$, when for each true assignment t, we have that t satisfies C_1 if and only if t satisfies C_2.

The basic syntactic operation with clauses is resolution: given two clauses c_1 and c_2 such that there is a variable r such that $r \in c_1$ and $\neg r \in c_2$, then the resolution of c_1 and c_2 by r is the clause $R(c_1, c_2, r) = (c_1 \setminus \{r\}) \cup (c_2 \setminus \{\neg r\})$. A set of clauses C is said to be complete under set of variables V if and only if for each $c_1, c_2 \in C$, if $R(c_1, c_2, r)$ is not trivial, then $R(c_1, c_2, r) \in C$ and if $c \in C$ and $c \subseteq c' \in \mathcal{L}(V)$ (i.e., c' is subsumend by c), and then $c' \in C$. Given a set of clauses, C, there is always a minimum set of clauses (in the sense of inclusion) which contains C and is complete (the set obtained by adding to C all the clauses obtained by resolution and subsumption). It will be denoted by $\mathcal{C}(C)$. It is clear that two sets of clauses, C_1 and C_2, defined for the same set of variables V, are equivalent if and only if $\mathcal{C}(C_1) = \mathcal{C}(C_2)$

A set of clauses C implies clause c if and only if $c \in \mathcal{C}(C)$ and this is denoted as $C \vdash c$. It is well known that this is equivalent to the fact that any true assignment t satisfying C also satisfies c. If C_1 and C_2 are two sets of clauses and for each $c \in C_2$ we have that $C_1 \vdash c$, it can be said that C_1 implies C_2, which is denoted by $C_1 \vdash C_2$. This is equivalent to the fact that any satisfying true assignment for C_1 is also a true assignment for C_2. It is also well known that C_1 and C_2 are equivalent if and only if $C_1 \vdash C_2$ and $C_2 \vdash C_1$.

The satisfiability problem (SAT) consists of determining if for a given set of clauses C there is a satisfying true assignment t for it. It is clear that if C is empty, then the answer is positive, and if the empty clause belongs to C, the answer is negative. This is an NP-complete problem [10], and therefore, hard to solve.

The algorithms in this paper will be based on the Davis–Putnam deletion algorithm [1]. The basic step of this algorithm is the deletion of a variable in a set of clauses C, denoted as C^{-v}. This operation is carried out with the following steps:

1. Compute $C_v^+ = \{c \in C \, : \, vs. \in c\}$
   $C_v^- = \{c \in C \,|\, \neg vs. \in c\}$
   $C_v^0 = C \setminus (C_v^+ \cup C_v^-)$
2. The result is
   $C^{-v} = \{R(c, c', v) \, : \, c \in C_v^+, c' \in C_v^-, R(c, c', v) \text{ non trivial}\} \cup C_v^0$.

An algorithm for SAT based on the deletion of variables is depicted in Algorithm 1. At the end of the loop, as all the variables have been deleted, one of the two conditions ($C = \emptyset$ or $\emptyset \in C$) must be satisfied. Its efficiency depends of the order of removing variables; however, in general, this algorithm has the problem of producing too many clauses that are large in size (number of literals).

It is also advisable to make some basic operations on C so that a simpler equivalent set is obtained. For example, it is always good to remove subsumed clauses, i.e., if $c, c' \in C$ and $c \subsetneq c'$, then remove c' from C. Unit propagation is another important step. For that, we need the concept of restriction of a set of clauses C to a literal ℓ, which is the set of clauses that is equivalent to C under the condition that ℓ is true. It will be denoted as $U(C, \ell)$ and it is the set of clauses obtained from C by removing any clause containing ℓ and removing $\neg\ell$ from the clauses containing $\neg\ell$. It is important to remark that the true assignments satisfying $U(C, \ell)$ are the same as those satisfying C with ℓ set to *True*.

Algorithm 1 David-Putnam deletion algorithm

```
Require: $C$, a set of clauses.
Ensure: $sat$, a logical value indicating whether $C$ is satisfiable.
 1: procedure DP(C)
 2:     $V \leftarrow$ variables of $C$
 3:     for $v \in V$ do
 4:         $C \leftarrow C^{-v}$
 5:         if $C = \emptyset$ then
 6:             $sat \leftarrow True$
 7:             Break
 8:         end if
 9:         if $\emptyset \in C$ then
10:             $sat \leftarrow False$
11:             Break
12:         end if
13:     end for
14:     return $sat$
15: end procedure
```

Unit propagation consists of transforming C containing a unit clause with a single literal $\{\ell\} \in C$, into $U(C, \ell) \cup \{\{\ell\}\}$. This operation is repeated for each literal ℓ appearing in a unit clause $c = \{\ell\} \in C$.

If C is a set of clauses defined for variables V and V' is a subset of V, then the marginalization of C to V' is the set of clauses $\mathcal{C}(C) \cap \mathcal{L}(V')$, i.e., the set of all the clauses that are defined for variables in V' and are a consequence of the clauses in C. The marginalization to V' will be denoted as $C^{-V \setminus V'}$ making reference to the set of removed variables. The David and Putnam algorithm has two main advantages against other satisfiability algorithms [5,6]. The first one is that the deletion algorithm is really an algorithm to compute the marginal information and has many other utilities. Then, if the loop starting in step 3 of the algorithm is applied for variables $v \in V'$, then the value of C will be equivalent to $C^{-V'}$. This is a consequence of the fact that each step carries out a deletion of a variable $v \in V'$ and that the set of clauses is a particular case of the Shenoy–Shafer axiomatic framework for local computation [2,6].

3. Table Representation of Sets of Clauses

A set of clauses C defined for variables V can be represented by a table with a dimension for each variable $v \in V$. If $v \in V$, we consider the set $\Omega_v = \{\neg v, v\}$ and $\Omega_V = \prod_{v \in V} \Omega_v$, where $\prod$ stands for Cartesian product. An element from Ω_V will be denoted in boldface $\mathbf{v}$. The component v of vector $\mathbf{v}$ will be denoted as $\mathbf{v}_v$. If $V' \subseteq V$, the subvector of V' components will be denoted by $\mathbf{v}^{-V \setminus V'}$, making reference to the removed components.

A table T for variables V will be a mapping $T : \Omega_V \to \{0, 1\}$. The set of variables of table T will be denoted by $V(T)$. A table T needs $2^{|V(T)|}$ bits to be represented.

To simplify the notation, if a table T is defined for variables V and being $V \subseteq V'$, $\mathbf{v}' \in \Omega_{V'}$, we will assume that $T(\mathbf{v}') = T(\mathbf{v}'^{-V' \setminus V})$, i.e., a table can be applied to a larger frame than the actual frame in which it is defined, simply by ignoring the components not in the set of variables for which it is defined.

The set of all tables defined for variables V will be denoted as $\mathcal{T}_V$.

A set of clauses C is represented by a table T if $T(\mathbf{v}) = 1$ if and only if the true assignment given by $t_{\mathbf{v}}(v) = \mathit{True}$ when $\mathbf{v}_v = v$ and $t_{\mathbf{v}}(v) = \mathit{False}$ when $\mathbf{v}_v = \neg v$ satisfies C. This is a semantic representation of the set of clauses C by determining the true assignments satisfying all the clauses, i.e., a truth table.

To simplify the notation, we will consider that vector $\mathbf{v}$ and $t_{\mathbf{v}}$ are equivalent, so that $\mathbf{v}$ can also be called a true assignment, but in fact making reference to $t_{\mathbf{v}}$. Then, in the tables, a true assignment will be a vector $\mathbf{v}$, and this true assignment satisfies table T when $T(\mathbf{v}) = 1$. The set of true assignments satisfying a table T will be $\mathbf{T}(T) = \{\mathbf{v} \in \Omega_{V(T)} \,:\, T(\mathbf{v}) = 1\}$.

The trivial table is the table T_e with $T_e(\mathbf{v}) = 1$ for any $\mathbf{v} \in \Omega_V$ and the contradictory table T_0 is the table with $T_0(\mathbf{v}) = 0$ for any $\mathbf{v} \in \Omega_V$. A trivial table represents the empty set of clauses and a contradictory table; an unsatisfiable set of clauses.

Given C, a table T can be easily computed starting with a trivial table, and then for each $c \in C$, we make $T(\mathbf{v}) = 0$ for each $\mathbf{v}$, such that for each literal $\ell \in c$ associated with variable v, we have that $\mathbf{v}_v \neq \ell$, i.e., $\mathbf{v}_v = v$ if $\ell = \neg v$ and $\mathbf{v}_v = \neg v$ if $\ell = v$.

Given a set of clauses C defined for variables V, the direct table representation is unfeasible if the number of variables of V is not small, given that the table size is $2^{|V|}$, where $|V|$ is the cardinal of V. For this reason, to represent a set of clauses C, first, we will partition the full set of clauses in small sets, $C_1, \ldots, C_k$, each one of them defined for a small set of variables $V_1, \ldots, V_k$. Then, the set C will be represented by the set of tables $T_1, \ldots, T_k$, where T_i is the table representing C_i.

In our experiments, the partition has been computed by the following steps:

1. Carry out unit propagation in C;
2. Group together clauses defined for the same set of variables, i.e., if $V(c) = V(c')$, then $c, c' \in C_i$;
3. Remove sets defined for non-maximal sets of variables, i.e., if C_i is such that there is another set of variables C_j such that $V(C_i) \subset V(C_j)$, then C_j is updated to $C_i \cup C_j$ and C_i is removed.

This is a procedure that we have found reasonable, but it is not the only possible one. We find that steps 1 and 2 are basic, but that there can be other alternative ways of performing step 3. For example, joining into a set, C_i and C_j, if $|C_i \cap C_j|$ is not small and $|C_i \cup C_j|$ is not too large, where large and small can be defined in terms of a couple of parameters n_1, n_2 in each case.

There are three basic operations with tables:

- **Combination.** If T_1 and T_2 are tables, then its combination is the table $T_1 \otimes T_2$ defined for set of variables $V = V(T_1) \cup V(T_2)$ and given by:

$$(T_1 \otimes T_2)(\mathbf{v}) = \min\{T_1(\mathbf{v}), T_2(\mathbf{v})\}.$$

  When considering $T_1(\mathbf{v})$, it is important to notice that this value is $T_1(\mathbf{v}^{V \setminus V(T_2)})$.
- **Variable deletion.** If T is a table and $V' \subseteq V(T)$, then table $T^{-V'}$ is the table defined for variables V' and given by:

$$T^{-V'}(\mathbf{v}') = \max\{T(\mathbf{v}) \,:\, \mathbf{v} \in \Omega_V, \mathbf{v}^{-V'} = \mathbf{v}'\}.$$

  $T^{-V'}$ will also be called the marginalization of T to $V(T) \setminus V'$. If $V' = \{v\}$, then $T^{-V'}$ will also be denoted as T^{-v}.
- **Conditioning.** If T is a table and ℓ is a literal associated with $v \in V(T)$, then the conditioning of T to ℓ is the table $U(T, \ell)$ defined for variables $V' = V(T) \setminus \{v\}$ and given by:

$$U(T, \ell)(\mathbf{v}') = T(\mathbf{v}', \ell),$$

  The conditioning operator can be extended to a partial true assignment: if T is a table, $V' \subseteq V(T)$ and $\mathbf{v}'$ is a true assignment for variables V', then $U(T, \mathbf{v}')$ is the table defined for variables $V'' = V(T) \setminus V'$ and given by:

$$U(T,\mathbf{v}')(\mathbf{v}'') = T(\mathbf{v}'',\mathbf{v}'),$$

where $\mathbf{v} = (\mathbf{v}'',\mathbf{v}')$ is the vector in Ω_V with $\mathbf{v}^{-V'} = \mathbf{v}''$ and $\mathbf{v}^{-V''} = \mathbf{v}'$.
We will assume that conditioning can be applied even if the variable v associated with ℓ is not in $V(T)$. In this case, $U(T,\ell) = T$. Analogously, if V' is not included in $T(V)$ and $\mathbf{v}' \in \Omega_{V'}$, we consider that $U(T,\mathbf{v}') = U(T,\mathbf{v}'^{-V'\setminus V(T)})$.

The following facts are immediate:

- If T_1 and T_2 are two tables associated with C_1 and C_2, respectively, then $T_1 \otimes T_2$ is associated with $C_1 \cup C_2$.
- If table T is associated with C, then $T^{-V'}$ is associated with $C^{-V'}$ and $U(T,\ell)$ is associated with $U(C,\ell)$.

We will now give an example, comparing the computations with clauses and tables.

Example 1. *Assume that we have a set C of clauses given by:*

$$\{p,q,\neg r\},\{p,\neg q,r\},\{\neg p,q,r\},\{q,r,\neg s\},\{q,\neg r,s\},\{\neg q,r,s\}$$

Assume that we want to delete variable q. Then, we have to compute all the resolutions of clauses containing q with clauses containing $\neg q$. After eliminating trivial clauses, the result is given by set of clauses C^{-q}:

$$\{p,r,\neg s\},\{\neg p,r,s\}$$

Using a table representation, we can build a table T_1 for the first three clauses and a table T_2 for the last three ones with the following values:

(p,q,r)	1
$(p,q,\neg r)$	0
$(p,\neg q,r)$	0
$(p,\neg q,\neg r)$	1
$(\neg p,q,r)$	0
$(\neg p,q,\neg r)$	1
$(\neg p,\neg q,r)$	1
$(\neg p,\neg q,\neg r)$	1

,

(q,r,s)	1
$(q,r,\neg s)$	0
$(q,\neg r,s)$	0
$(q,\neg r,\neg s)$	1
$(\neg q,r,s)$	0
$(\neg q,r,\neg s)$	1
$(\neg q,\neg r,s)$	1
$(\neg q,\neg r,\neg s)$	1

The combination $T_1 \otimes T_2$ will be the table defined for variables $\{p,q,r,s\}$ and given by:

(p,q,r,s)	1
$(p,q,r,\neg s)$	0
$(p,q,\neg r,s)$	0
$(p,q,\neg r,\neg s)$	0
$(p,\neg q,r,s)$	0
$(p,\neg q,r,\neg s)$	0
$(p,\neg q,\neg r,s)$	1
$(p,\neg q,\neg r,\neg s)$	1
$(\neg p,q,r,s)$	0
$(\neg p,q,r,\neg s)$	0
$(\neg p,q,\neg r,s)$	0
$(\neg p,q,\neg r,\neg s)$	1
$(\neg p,\neg q,r,s)$	0
$(\neg p,\neg q,r,\neg s)$	1
$(\neg p,\neg q,\neg r,s)$	1
$(\neg p,\neg q,\neg r,\neg s)$	1

If in this combination, variable q is deleted by marginalization, table $(T_1 \otimes T_2)^{-q}$ is obtained:

(p, r, s)	1
$(p, r, \neg s)$	0
$(p, \neg r, s)$	1
$(p, \neg r, \neg s)$	1
$(\neg p, r, s)$	0
$(\neg p, r, \neg s)$	1
$(\neg p, \neg r, s)$	1
$(\neg p, \neg r, \neg s)$	1

which is exactly the table associated with the two clauses in C^{-q}.

The following properties show that tables satisfy the basic Shenoy–Shafer axioms, and therefore, local computation is possible:

1. Combination is commutative and associate: $T_1 \otimes T_2 = T_2 \otimes T_1$, $T_1 \otimes (T_2 \otimes T_3) = (T_1 \otimes T_2) \otimes T_3$;
2. If $V(T) = V$ and V_1, V_2 are two disjoint subsets of V, then $V^{-(V_1 \cup V_2)} = \left(V^{-V_1}\right)^{-V_2}$;
3. If $V(T_1) = V_1$ and $V(T_2) = V_2$, then $(V_1 \otimes V_2)^{-V_2 \setminus V_1} = V_1 \otimes V_2^{-V_2 \setminus V_1}$.

The tables also satisfy the idempotent property: if T is a table and $V' \subseteq V(T)$, then $T \otimes T^{-V'} = T$. As a consequence of these properties, if we consider a set of variables V and $\mathcal{T} = \bigcup_{V' \subseteq V} \mathcal{T}_{V'}$, $\mathcal{T}$ is said to be information algebra [3,8]. It also has a neutral element, T_e, and a null element, T_0.

In information algebra, it is always possible to define a partial order, which in this case is the following: if T_1 and T_2 are tables defined for sets of variables V_1 and V_2, respectively, then we say that $T_1 \preceq T_2$ if and only if for each $\mathbf{v} \in \Omega_{V_1 \cup V_2}$, we have that $T_1(\mathbf{v}) \geq T_2(\mathbf{v})$. The intuitive idea is that T_2 contains more or the same information than T_1 (any true assignment satisfying T_2 will also satisfy T_1).

The following properties of this relation are immediate:

1. If T is a table and $V' \subseteq V$, then $T^{-V'} \preceq T$;
2. If T_1 and T_2 are tables, then $T_1 \preceq (T_1 \otimes T_2)$;
3. If T_1, T_2, T_3 are tables, then $(T_1 \otimes T_2) \preceq T_3$ if and only if $T_1 \preceq T_3$ and $T_2 \preceq T_3$.

Two tables, T_1 and T_2, are said to be equivalent, $T_1 \equiv T_2$, if and only if $T_1 \preceq T_2$ and $T_2 \preceq T_1$. If T_1 and T_2 are defined on V_1 and V_2, respectively, and T_e^1 and T_e^2 are the neutral tables in $\mathcal{T}_{V_2 \setminus V_1}$ and $\mathcal{T}_{V_1 \setminus V_2}$, respectively, we can immediately prove that T_1 and T_2 are equivalent if and only if $T_1 \otimes T_e^1 = T_2 \otimes T_e^2$. The multiplication by the neutral element is necessary for tables to be defined for the same set of variables. If T_1 and T_2 are equivalent and $V(T_1) = V(T_2)$, then they are identical, i.e., $T_1 = T_2$. The quotient set of $\mathcal{T}$ under this equivalence relation is called the domain-free valuation algebra associated with the valuation algebra [3]. In the following, we will consider that we work with equivalent classes of tables, and that a table can be changed into any equivalent one. All the neutral tables T_e defined in different sets of variables are equivalent. Furthermore, the contradictory tables T_0 are equivalent. As a consequence, we will not make reference to the set of variables in which they are defined.

There is a disjunction operation [3,8] which can be defined on the set of tables: if T_1 and T_2 are tables, then its disjunction is the table $T_1 \oplus T_2$ defined for the set of variables $V = V(T_1) \cup V(T_2)$ and given by:

$$(T_1 \oplus T_2)(\mathbf{v}) = \max\{T_1(\mathbf{v}), T_2(\mathbf{v})\}.$$

It is immediately clear that disjunction is commutative and associative. Furthermore, that combination satisfies the distributive property with respect to disjunction and disjunction is also distributive with respect to the combination. In fact, we have the Boolean information algebra from [3], being the complementary to T in the table $T^c = 1 - T$.

We have some interesting properties relating disjunction with the basic table operations.

Proposition 1. *If T is a table and $v \in V(T)$, then $T^{-v} = U(T,v) \oplus U(T,\neg v)$.*

Proof. See Proposition A1 in Appendix A. □

Proposition 2. *If T_1 and T_2 are tables and $v \in V(T_1) \cap V(T_2)$, then $(T_1 \oplus T_2)^{-v} = T_1^{-v} \oplus T_2^{-v}$.*

Proof. See Proposition A2 in Appendix A. □

4. Deletion Algorithm with Tables

We assume that we have some information represented by a set of tables: $H = \{T_1, \ldots, T_k\}$. This set is intended to be a representation of the combination: $\otimes(H) = T_1 \otimes \cdots \otimes T_k$. As the size of the tables increases exponentially with the number of dimensions of the tables and $V(\otimes(H)) = \bigcup_{i=1}^k V(T_i)$, then the representation by a set can use much less space than the representation by a single table (except when the tables are defined for the same set of variables, but this is never the case according to our procedure to build the initial tables). $V(\otimes(H))$ will be denoted as $V(H)$. The total size of the tables in H will be $\sum_{T \in H} 2^{|V(T)|}$. For example, if we have three tables defined for variables $\{p_1, p_2\}, \{p_2, p_3, p_4\}, \{p_4, p_5, p_6\}$, the size of the tables will be $4 + 8 + 8 = 16$, but their combination will be defined for $V(H) = \{p_1, p_2, p_3, p_4, p_5\}$, which corresponds to a table of size 32.

If $V = V(\otimes(H))$, vector $\mathbf{v} \in \Omega_V$ satisfies the set of tables H when $T_i(\mathbf{v}) = 1$ for any $T_i \in H$.

We will say that two sets, H_1, H_2, are equivalent if and only if $\otimes(H_1)$ is equivalent to $\otimes(H_2)$. We also say that $H_1 \preceq H_2$ when $\otimes(H_1) \preceq \otimes(H_2)$.

The following properties are immediate:

- If H_1 and H_2 are equivalent, then $H \cup H_1$ will be equivalent to $H \cup H_2$;
- $H_1 \preceq H_2$ if and only if $\{T\} \preceq H_2$ for any $T \in H_1$;
- If $H' \subseteq H$, then $(H \setminus H') \cup \otimes(H')$ is equivalent to H;
- If $H' \preceq H$, then $H' \cup H$ is equivalent to H.

The set of true assignments of a set H will be the $\mathbf{T}(H) = \bigcap_{T \in H} \mathbf{T}(T)$, where the intersection of two sets defined on different sets of indexes is defined as follows: if $R_1 \subseteq \Omega_{V_1}, R_2 \subseteq \Omega_{V_2}$, then $R_1 \cap R_2 = \{\mathbf{v} \in \Omega_{V_1 \cup V_2} : \mathbf{v}^{-V_2 \setminus V_1} \in R_1, \mathbf{v}^{-V_1 \setminus V_2} \in R_2\}$.

The operations with tables can be translated to set of tables, taking into account that an operation on set H is really carried out on $\otimes(H)$ and that equivalent tables represent the same information. Some operations can be carried out in a simple way:

- The combination can be completed by a simple union:
$$H_1 \otimes H_2 = H_1 \cup H_2.$$
- The disjunction of sets of tables can be computed as follows:
$$H_1 \oplus H_2 = \{T_1 \oplus T_2 : T_1 \in H_1, T_2 \in H_2\}.$$
- The conditioning is the conditioning of its tables:
$$U(H,\ell) = \{U(T,\ell) : T \in H\}.$$

The deletion of variables is the more complex operation for sets of tables. In the following, we describe several methods to carry out this operation.

The marginal problem is as follows: given a set of tables, H, and $V' \subseteq V(H)$, then compute H' such that $\otimes(H')$ is equivalent to $\otimes(H)^{-V'}$. This set will also be denoted as H^{-V}, considering that we are computing an element of the equivalence class.

As tables satisfy the basic Shenoy–Shafer axioms, the marginalization can be computed with basic Algorithm 2, which is very similar to Algorithm 1, but now expressed as a marginalization algorithm and with information represented by tables. Procedure MARGINALIZE0 (H_v) is very simple and it is depicted in Algorithm 3, where the condition of T equivalent to T_0 seems a bit artificial and unnecessary; it was included to show the similarity with other variants of this operation that we will introduce. When implemented, this test is also carried out, and when T is equivalent to T_0, we return the contradictory table which is defined for the empty set of variables and contains a single 0 value.

Algorithm 2 Deletion algorithm

```
Require: $H$, a set of tables.
Require: $V'$, a set of variables to remove.
Ensure: $H'$, a set of tables representing $H^{-V'}$.
 1: procedure DELETION ($H$, $V'$)
 2:     for $v \in V'$ do
 3:         $H_v \leftarrow$ tables $T \in H$ such that $v \in V(T)$
 4:         $(R_1, R_2) \leftarrow$ MARGINALIZE0 ($H_v$,$v$)
 5:         if $R_1$ contains $T_0$ then
 6:             return $\{T_0\}$
 7:         else
 8:             $H \leftarrow (H \setminus H_v) \cup R_1$
 9:         end if
10:     end for
11:     return $H$
12: end procedure
```

This algorithm can be used to solve the satisfiability problem: if for function DELETION(H, V'), $V' = V(H)$, then all the variables are removed and H will only contain tables defined for the empty set of variables which have a single value. Taking into account that trivial $T_e = 1$ tables are not introduced, there are only two possibilities: H contains T_0 and then the problem is unsatisfiable or $H = \emptyset$ and the problem is satisfiable. However, the algorithm can also be used to compute marginal information and to compile the information in initial set H. This compilation is based on the following result.

Algorithm 3 Basic version of marginalize

```
Require: $H_v$, a set of tables containing variable $v$.
Require: $v$, the variable to remove
Ensure: $R_1$, a set of tables representing $H_v^{-v}$.
Ensure: $R_2$, a set of tables containing $v$.
 1: procedure MARGINALIZE0 ($H_v$,$v$)
 2:     $T_1 \leftarrow \otimes(H_v)$
 3:     $T_2 \leftarrow T_1^{-v}$
 4:     if $T_2$ is equivalent to $T_e$ then
 5:         $R_1 \leftarrow \emptyset$
 6:     else
 7:         if $T_2$ is equivalent to $T_0$ then
 8:             $R_1 \leftarrow \{T_0\}$
 9:         else
10:             $R_1 \leftarrow \{T_2\}$
11:         end if
12:     end if
13:     $R_2 \leftarrow \{T_1\}$
14:     return $(R_1, R_2)$
15: end procedure
```

Proposition 3. *In each application of* MARGINALIZE0 (H_v)*, we have that* H_v *is equivalent to* $R_1 \cup R_2$*. Furthermore, if we bring* H' *into the update set* H*, computed in Step 8 of Algorithm 2, then* H *is equivalent to* $H' \cup R_2$*.*

Proof. See Proposition A3 in Appendix A. $\square$

As a consequence of this result, when applying the deletion algorithm, if we call $R_2[v]$ to the R_2 set which is obtained after removing variable v, we have that H is equivalent to $(\bigcup_{v \in V(H)} R_2[v]) \cup H^{-V(H)}$. It is important to remark that $H^{-V(H)}$ is the result of removing all the variables, and then a table from this set is defined for the empty set of variables, which is a number. There are two possibilities: first, if H is satisfiable, then all the tables are 1 (T_e) and can be removed, representing $H^{-V(H)}$ by the empty set. Second, if H is unsatisfiable, $H^{-V(H)}$ will contain T_0, and the full H is equivalent to it: $\{T_0\}$. Let us call this set $R_2[\emptyset]$.

If $H(V) = \{v_1, \ldots, v_k\}$ and the variables are removed in this order: $(v_1, \ldots, v_k)$, then we have that $H^{-\{v_1,\ldots,v_i\}}$ is equivalent to $(\bigcup_{j=i+1}^{k} R_2[v_j]) \cup R_2[\emptyset]$. Therefore, $H^{-\{v_1,\ldots,v_i\}}$ is equivalent to $H^{-\{v_1,\ldots,v_{i+1}\}} \cup R_2[v_{i+1}]$. In this way, we do not only compute the marginal information, but with sets $R_2[v_i]$, we have the necessary information to recover in a backward way the marginal sets: from the marginal in which all the variables are removed, $R_2[\emptyset]$, to the marginal in which no variable is removed, H. This fact can be useful, among other things to obtain the true assignments satisfying all the tables, in case they are satisfiable. The following result provides a procedure to obtain the true assignments satisfying a set of tables H computed from sets of tables $\{R_2[v_i]\}$ obtained when applying a deletion algorithm.

Proposition 4. *Assume a set of tables* H *and that Algorithm 2 is applied removing variables in* $V(H)$ *in order* $(v_1, \ldots, v_k)$*. Assume also that* $R_2[\emptyset]$ *is equivalent to the empty set, i.e., the problem is satisfiable; then, if* $\mathbf{T}_i$ *is the set of true assignments satisfying the set of tables* $H^{-\{v_1,\ldots,v_i\}}$ *and* $\mathbf{T}_0$ *is the true assignments of* H*, then these sets can be computed in reverse order of* $i = 1, \ldots, k$ *in the following way:*

- *Start with* $\mathbf{T}_i = \emptyset$*;*
- *Make* $\mathbf{T}_k$ *equal to the set containing an empty vector* $\mathbf{v}_0 \in \Omega_\emptyset$*;*
- *For each* $\mathbf{v}_{i+1} \in \mathbf{T}_{i+1}$*, compute* T_{i+1}*, which is the only table in* $U(R_2[v_{i+1}], \mathbf{v}_{i+1})$*, which is a table defined only for variable* v_{i+1}*, then this table is never equal to* T_0*, and if* $\mathbf{v}_i^1$ *and* $\mathbf{v}_i^2$ *are the true assignments obtained by extending* $\mathbf{v}_{i+1}$ *to variables* $\{v_{i+1}, \ldots, v_k\}$ *and given by* $\mathbf{v}_i^1 = (v_{i+1}, \mathbf{v}_{i+1}), \mathbf{v}_i^2 = (\neg v_{i+1}, \mathbf{v}_{i+1})$*, i.e., by considering* v_{i+1} *true and false, respectively, then:*
  - *if* $T_{i+1} = T_e$*, add* $\mathbf{v}_i^1$ *and* $\mathbf{v}_i^2$ *to* $\mathbf{T}_i$*;*
  - *if* $T_{i+1}(v_{i+1}) = 1, T_{i+1}(\neg v_{i+1}) = 0$*, add* $\mathbf{v}_i^1$ *to* $\mathbf{T}_i$*;*
  - *if* $T_{i+1}(v_{i+1}) = 0, T_{i+1}(\neg v_{i+1}) = 1$*, add* $\mathbf{v}_i^2$ *to* $\mathbf{T}_i$*.*

Proof. See Proposition A4 in Appendix A. $\square$

This result is the basic for algorithms to compute one solution, all the solutions, or a random solution given a satisfiable set of clauses. It should start with a $\mathbf{T}_k$, which is a set containing an empty vector $\mathbf{v}_0 \in \Omega_\emptyset$. The main difference in these algorithms is when $T_{i+1} = T_e$. When computing one solution, we only pick $\mathbf{v}_i^1$ or $\mathbf{v}_i^2$, when computing all the solutions, both $\mathbf{v}_i^1$ and $\mathbf{v}_i^2$ are selected, and when computing a random solution, there is a random selection of $\mathbf{v}_i^1$ or $\mathbf{v}_i^2$. In the last case, an importance sampling algorithm in the set of solutions is obtained: starting with a weight of 1.0, each time we have a random selection, the weight must be multiplied by 2.0.

In Algorithm 3, we have described the basic marginalization operation, which is the same as the one applied to the general valuation-based systems [2]. However, Boolean tables allow other alternative forms of marginalization. The first one is depicted in Algorithm 4 and determines that it is not necessary to combine all the tables in order to compute the marginal. In fact, only pairwise combinations are necessary.

Algorithm 4 Pairwise combination version of marginalize

```
Require: $H_v$, a set of tables containing variable $v$.
Require: $v$, the variable to remove
Ensure: $R_1$, a set of tables representing $H_v^{-v}$.
Ensure: $R_2$, a set of tables containing $v$.
1: procedure MARGINALIZE1($H_v$,$v$)
2:     $R_1 = \{(T_i \otimes T_j)^{-v} : T_i, T_j \in H_v\}$
3:     $R_2 \leftarrow H_v$
4:     if $R_1$ contains $T_0$ then
5:         $R_1 \leftarrow \{T_0\}$
6:     end if
7:     Remove neutral tables $T_e$ from $R_1$
8:     return $(R_1, R_2)$
9: end procedure
```

The following proposition shows that R_1 is also a set of tables representing H_v^{-v}.

Proposition 5. *If H_v is a set of tables containing variables v, then R_1 computed in Algorithm 4 represents H_v^{-v}.*

Proof. See Proposition A5 in Appendix A. □

Once this is proved, then we can replace MARGINALIZE0 by MARGINALIZE1 in the deletion algorithm and everything works, even the method to compute the solutions given in Proposition 4. The only difference is that now $R_2[v_{i+1}]$ contains, in general, more than one table, and when computing $U(R_2[v_{i+1}], \mathbf{v}_{i+1})$, we have to condition every table in $R_2[v_{i+1}]$, being the result a set of tables depending on variable v_{i+1}. These tables are combined to produce T_{i+1}.

The main difference between MARGINALIZE0 and MARGINALIZE1 is that the former produces an unique table in R_1 and R_2, while the latter produces several tables in both sets, but with smaller size. As the size of a table T is $2^{|V(T)|}$, in general, MARGINALIZE1 is more efficient, but we have to take into account that the number of tables in R_1 is quadratic in relation with the number of tables in H_v and this fact should be taken into account.

However, there is another very important alternative marginalization when we have a variable which is functionally dependent of other variables [20]. This happens very often, especially in problems encoding circuits [21].

If T is a table and $v \in V(T)$, we say that v is functionally determined in T if and only if $U(T,v) \otimes U(T,\neg v)$ is equivalent to T_0. If $V' = V(T) \setminus \{v\}$, this implies that for any $\mathbf{v}' \in \Omega_{V'}$ we have that $(U(T,v) \otimes U(T,\neg v))(\mathbf{v}') = 0$, i.e., either $U(T,v)(\mathbf{v}') = T(\mathbf{v}',v) = U(T,\mathbf{v}')(v) = 0$ or $U(T,\neg v)(\mathbf{v}') = T(\mathbf{v}',\neg v) = U(T,\mathbf{v}')(\neg v) = 0$, i.e., for each $\mathbf{v}'$, there is, at most, one possible value for variable V, v or $\neg v$ for which table T has a value of 1 (at least, one of the values v or $\neg v$ is impossible). This definition generalizes definition given in terms of clauses in [20], which also requires that T^{-v} is equivalent to the neutral element.

In the case that there is a table $T \in H_v$ such that v is functionally determined on T, then marginalization can be done as in Algorithm 5.

This marginalization is much more efficient, as the number of the tables in R_1 is smaller than in MARGINALIZE1. In fact, the number of tables in the problem does not increase: the number of tables in R_1 is always less or equal than the number of tables in H_v. We give an example illustrating the benefits of using this marginalization.

Algorithm 5 Marginalize with functional dependence

```
Require: $H_v$, a set of tables containing variable $v$.
Require: $v$, the variable to remove
Require: $T$ a table in which $v$ is functionally determined
Ensure: $R_1$, a set of tables representing $H_v^{-v}$.
Ensure: $R_2$, a set of tables containing $v$.
1: procedure MARGINALIZE2 ($H_v$,$v$,$T$)
2:     $R_1 = \{(T \otimes T')^{-v} : T' \in H_v\}$
3:     $R_2 \leftarrow \{T\}$
4:     if $R_1$ contains $T_0$ then
5:         $R_1 \leftarrow \{T_0\}$
6:     end if
7:     Remove neutral tables $T_e$ from $R_1$
8:     return $(R_1, R_2)$
9: end procedure
```

Example 2. *Assume that we are deleting variable p and that we have three tables:*

$$T_1(p,q), T_2(p,r,s), T_3(p,u,v).$$

If MARGINALIZE0 *is applied, then we have to combine the three tables producing a table* $(T_1 \otimes T_2 \otimes T_3)(p,q,r,s,u,v)$*, which depends on six variables and has a size of 64. If* MARGINALIZE1 *is applied, then we have to compute* $(T_1 \otimes T_2)(p,q,r,s)$, $(T_1 \otimes T_3)(p,q,u,v)$, $(T_2 \otimes T_3)(p,r,s,u,v)$ *(the combination of a table by itself is not necessary, as this produces the same table which is included in one of these combinations), i.e., we have more tables but of smaller size (16 + 16 + 32). However, if it is known that p is determined by q in table* T_1*, then, for* MARGINALIZE2*, only combinations* $(T_1 \otimes T_2)(p,q,r,s)$, $(T_1 \otimes T_3)(p,q,u,v)$ *are necessary and the result is based on two tables of sizes 16+16, producing an important saving in space and computation. Please note that to finish the marginalization step, variable p should be removed on the computed tables by marginalization, but this step has been omitted in order to keep the notation simpler.*

This marginalization is correct, as R_1 is equivalent to H_v^{-v} and H_v is equivalent to $R_1 \cup R_2$, as the following results shows.

Proposition 6. *If* (R_1, R_2) *are the sets computed in* MARGINALIZE2 *and the initial conditions required by the algorithm are satisfied; then,* H_v^{-v} *is equivalent to* R_1 *and* H_v *is equivalent to* $R_1 \cup R_2$*.*

Proof. See Proposition A6 in Appendix A. □

5. Additional Processing Steps and Marginalization Strategy

In this section, we introduce some additional steps which can be added to the basic deletion algorithm to improve efficiency or to enlarge the family of problems that can be solved.

5.1. Combining Tables

Many times we have in a problem two tables T_1 and T_2, such that $V(T_1) \subseteq V(T_2)$. We can substitute them by its combination $T = T_1 \otimes T_2$ obtaining an equivalent problem. Doing this can have potential advantages: it reduces the size of the problem specification, as the size of T is the same than the size of T_2, but it also increases the chances of having a variable which is functionally dependent of the others. Remember that v is determined in T when $U(T,v) \otimes U(T,\neg v) = T_0$, so if $T \preceq T'$, then v will be also determined in T'. We can consider the procedure COMBINEINCLUDED(H) which compares the sets of variables of each pair of tables $T_1, T_2 \in H$, and substitutes the pair by its combination when one set is included in the other. One important point is the sets to which this procedure applies. In

Algorithm 2, CombineIncluded(H) could be applied to the set H after each deletion of a variable, to the set H_v, or to R_1. Applying it to H could be very costly, as the number of tables can be high and we have to compare all the pairs of tables. A similar effect can be obtained by applying it to set H_v, which are the tables effectively used in each deletion step, but which is of smaller size. It is also important to apply it to set R_1, as it can significantly reduce the number of tables, which can be high when Marginalizalize1 is used to delete a variable.

In the case of H_v, we have already implemented a more aggressive combination method, which combines two tables T_1 and T_2 when $2^{V(T_1 \otimes T_2)} \leq 2^{V(T_1)} + 2^{V(T_2)}$. This always happens when $V(T_1) \subseteq V(T_2)$, but also when $|V(T_1)| = |V(T_2)|$ and $V(T_1) = V(T_2) \setminus \{u\} \cup \{v\}$, i.e., tables are defined for the same variables, except for variables $u \in V(T_2), v \in V(T_1)$. We will call to this procedure GroupTables(H), which will be applied to H_v in the case of Marginalize1 when the number of tables in H is greater or equal is greater or equal than a given threshold, N.

Example 3. *Assume three tables, i.e., T_1 for variables p, q, T_2 for variables q, r, and T_3 for variables p, r, that are given by:*

(p, q)	1
$(p, \neg q)$	0
$(\neg p, q)$	1
$(\neg p, \neg q)$	1

,

(q, r)	0
$(q, \neg r)$	1
$(\neg q, r)$	1
$(\neg q, \neg r)$	1

,

(p, r)	1
$(p, \neg r)$	1
$(\neg p, r)$	0
$(\neg p, \neg r)$	1

If GroupTables (H) *is applied, then tables T_1 and T_2 are combined, as the result has a size of 8 and the combined tables sizes are 4 + 4. Then, table T_3 is also combined, as it is defined for a set of variables included in the variables of the combination of $T_1 \otimes T_2$. The result is that the three tables are replaced by their combination, which is given by the following table:*

(p, q, r)	0
$(p, q, \neg r)$	1
$(p, \neg q, r)$	0
$(p, \neg q, \neg r)$	0
$(\neg p, q, r)$	0
$(\neg p, q, \neg r)$	1
$(\neg p, \neg q, r)$	0
$(\neg p, \neg q, \neg r)$	1

Observe that this table is smaller than the sum of the sizes of the three combined tables. Furthermore, in the combination, we can observe that r is determined by (p, q), for which Marginalize2 *can be applied.*

5.2. Unidimensional Arrays

Unidimensional arrays have a special consideration. If we are going to introduce in H a table T and $|V(T)| = 1$, then there are three possibilities:

- T is equal to T_e, the neutral element. In this case, the table is not introduced in H.
- T is equal to T_0, the contradiction. In this case, the whole set is equivalent to $\{T_0\}$ and all the other tables are removed from H. In fact, in our implementation, the equality to T_0 is checked for any table to be introduced to H, whatever its dimension is.
- In T, there is a value with value 0 and other with value 1. If ℓ is the literal with value 1, this table is equivalent to the unit literal ℓ and we can carry out a unit propagation. This is done by transforming any other table $T' \in H$ into $U(T', \ell)$ and finally introducing table T. In our implementation, instead of introducing the table T, we keep a set of unit literals, and each time another table is introduced in H, the conditioning to the literals in this set is carried out.

Example 4. *Consider the table T' about p, q given by:*

(p, q)	1
$(p, \neg q)$	0
$(\neg p, q)$	1
$(\neg p, \neg q)$	1

If a table, T, with p is introduced with $T(\neg p) = 1, T(p) = 0$, then we can transform table T' into $U(T', \neg p)$, which is again an unidimensional table defined for q with $U(T', \neg p)(q) = 1$, $U(T', \neg p)(\neg q) = 0$. This table may produce further simplifications of tables containing variable q.

5.3. Splitting

The size of the tables in H_v is important when applying the deletion step: the smaller the size, the faster this step can be completed. For that reason, we have implemented a procedure to reduce this size. In order to do this, for each $T \in H_v$, Algorithm 6 is applied, where MINIMIZE (T_1,T_2,T, V) is a procedure that tries to make T_1 as smaller as possible (by marginalization) under the condition that $T = T_1 \otimes T_2$ and that variables in V cannot be removed. The details of this minimizing algorithm can be found in Algorithm 7.

When this split is applied to any table T in H_v, each time T is split into T_1, T_2 with $|V(T_1)| < |V(T)|$, then T is changed to T_1, T_2 in H and to T_1 in H_v. The general procedure making this transformation is called SPLITG(H_v, H).

Algorithm 6 Splitting a table before deleting a variable

```
Require: $T$, a table containing variable $v$.
Require: $v$, the variable to remove
Ensure: $T_1$, a table containing $v$.
Ensure: $T_2$, a table not containing $v$.
1: procedure SPLIT($T$,$v$)
2:     $T_2 \leftarrow T^{-v}$
3:     $T_1 \leftarrow T$
4:     $T_1 \leftarrow$ MINIMIZE($T_1$,$T_2$,$T$, $\{v\}$)
5:     return $(T_1, T_2)$
6: end procedure
```

Algorithm 7 Minimizing the splitting table

```
Require: $T_1, T_2, T$, tables such that $T_1 \otimes T_2 = T$.
Require: $V$, a set of variables that can not be removed.
Ensure: $M$ a table in which $T_1$ is minimized.
1: procedure MINIMIZE($T_1$,$T_2$, $T$, $V$)
2:     if $V(T_1) \setminus V = \emptyset$ then
3:         $M \leftarrow T_1$
4:         return $M$
5:     end if
6:     Let $v$ be an element from $V(T_1) \setminus V$
7:     $T_3 \leftarrow T_1^{-v}$
8:     if $T_3 \otimes T_2 = T$ then
9:         return MINIMIZE($T_3$,$T$,$T_2$, $V \cup \{v\}$)
10:    else
11:        return MINIMIZE($T_1$,$T$,$T_2$, $V \cup \{v\}$)
12:    end if
13: end procedure
```

Example 5. *Assume that in our set of tables, we have a table T with variables p, q, r given by:*

(p, q, r)	1
$(p, q, \neg r)$	0
$(p, \neg q, r)$	0
$(p, \neg q, \neg r)$	0
$(\neg p, q, r)$	1
$(\neg p, q, \neg r)$	0
$(\neg p, \neg q, r)$	1
$(\neg p, \neg q, \neg r)$	1

If we want to remove variable r, then instead of using this table, we can try to split it into two tables, one of them not depending on r. Then we first compute the marginal $T_2 = T^{-r}$, which is given by:

(p, q)	1
$(p, \neg q)$	0
$(\neg p, q)$	1
$(\neg p, \neg q)$	1

Next, we minimize T conditioned to T_2 obtaining in this case the following table T_1:

(q, r)	1
$(q, \neg r)$	0
$(\neg q, r)$	1
$(\neg q, \neg r)$	1

In this way, we obtain the decomposition $T = T_1 \otimes T_2$; we can change T in our set of tables by the two tables T_1, T_2, and then, as T_2 does not depend on r, when deleting this variable, only T_1 has to be considered, which has a lower dimension than the original table T, simplifying in this way the deletion step.

5.4. Minimizing the Dependence Set

If we have the set of tables H_v and there is a table $T \in H_v$ in which v is functionally determined, then the deletion is, in general, quite efficient, but it also depends on the size of T. If there is a table T_m that can be obtained from T by marginalization and v continues being determined in T_m, then the result is also correct if we call MARGINALIZE2 (H_v,v,T_m). The reason is very simple: as T_m is obtained by marginalization of a table in H_v, then $H_v \cup \{T_m\}$ is equivalent to H_v and MARGINALIZE2 ($H_v \cup \{T_m\}$,v,T_m) produces a correct result. The only difference between MARGINALIZE2 ($H_v \cup \{T_m\}$,v,T_m) and MARGINALIZE2 (H_v,v,T_m) is that in the former, $(T_m \otimes T_m)^{-v}$ is included. However, this table is less informative than $(T \otimes T_m)^{-v}$, which is included in MARGINALIZE2 (H_v,v,T_m), and then the two results are equivalent.

The algorithm we have applied to compute a table T_m with smaller size in which there is functional dependence is depicted in Algorithm 8, initially with $V = \{v\}$. In it, we assume that we have a function CHECKDETER(T,v) that determines when v is functionally determined in T.

Algorithm 8 Minimizing the dependence of a variable in a table

```
Require: $T$, a table.
Require: $v$, a variable which is determined in $T$.
Require: $V$, a set of variables which can not be deleted.
Ensure: $T_m$ a marginal table in which $v$ continues being determined.
1: procedure MINDEP($T$,$v$, $V$)
2:     if $V(T) \setminus V = \emptyset$ then
3:         $T_m \leftarrow T$
4:     end if
5:     Let $v$ be an element from $V(T) \setminus V$
6:     $T' \leftarrow T^{-v}$
7:     if CHECKDETER ($T'$,$v$) THEN
8:         $T_m \leftarrow$MINDEP ($T'$,$v$, $V \cup \{v\}$)
9:     ELSE
10:        $T_m \leftarrow$ MINDEP ($T$,$v$, $V \cup \{v\}$)
11:    END IF
12:    RETURN $T_m$
13: END PROCEDURE
```

Example 6. *Assume a table T about variables p, q, r given by:*

(p, q, r)	0
$(p, q, \neg r)$	1
$(p, \neg q, r)$	0
$(p, \neg q, \neg r)$	1
$(\neg p, q, r)$	1
$(\neg p, q, \neg r)$	0
$(\neg p, \neg q, r)$	1
$(\neg p, \neg q, \neg r)$	0

In this table, r is determined by (p, q), but if T^{-q} is computed, the result is:

(p, r)	0
$(p, \neg r)$	1
$(\neg p, r)$	1
$(\neg p, \neg r)$	0

Furthermore, r is determined by p on it. MARGINALIZE2 *is more efficient using this smaller table instead of the original one.*

5.5. Alternative Deletion Procedures

When applying MARGINALIZE0 (H_v,v), table $T = \otimes(H_v)^{-v}$ is computed. In some situations, it is possible that this table is of a small size, but MARGINALIZE1 (H_v,v) or MARGINALIZE2 (H_v,v) are applied. In that case, instead of computing R_1, an alternative method is to compute the maximal sets of the tables from R_1: $M(R_1) = \text{Maximal}\{V(T') : T' \in R_1, V(T')\}$, where Maximal is removed from a family of sets, which are strictly included into another set of the family.

Observe that it is not necessary to actually compute the tables in R_1, but only the sets of variables associated with these tables.

Then, we can compute $R_1' = \{T^{-V(T) \setminus B} : B \in M(R_1)\}$. When comparing R_1 and R_1' we can observe the following facts:

- Each element from R_1 is equal to $T' = (T_i \otimes T_j)^{-v}$, where $T_i, T_j \in H_v$ or a combination of several sets of this type, when COMBINEDINCLUDED has been applied. Then, there will be another table $T'' \in R_1'$ defined for the same set of variables and computed as $T^{-V(T) \setminus V(T'')}$. As T'' is the result of marginalizing after combining all the tables, instead of combining only two tables, we have that $T' \preceq T''$, and in some cases, the tables are not equivalent.

- The whole sets of tables R_1 and R'_1 are equivalent. In fact, both are equivalent to $T = \otimes(H_v)^{-v}$. This is known for R_1. For R'_1, as any element from R'_1 is a marginalization of T, $R'_1 \preceq \{T\}$. On the other hand, as a consequence of the above point, $R_1 \preceq R'_1$, and as R_1 is equivalent to T, we have that R'_1 is also equivalent to $\{T\}$.

Computing R'_1 by computing T and $M(R_1)$ and performing marginalizations has, in general, a higher computational cost, but this may be lowered if T contains few variables; moreover, on the positive side, we have more informative tables in R'_1 and there are more opportunities to find functional dependencies of variables in tables, which can improve the efficiency of posterior deletion steps.

In our implementations this decision is taken by fixing a threshold K, and if $|V(T)| \leq K$, R'_1 is computed. In other cases, R_1 is computed. The versions of MARGINALIZE1 and MARGINALIZE2 COMPUTING R'_1 ARE CALLED MARGINALIZE1B AND MARGINALIZE2B, respectively.

Example 7. *Assume that we are going to delete p and that we have three tables T_1, T_2, T_3, defined for variables $(p,q), (p,r)$, and (p,s), given by:*

(p,q)	1
$(p,\neg q)$	0
$(\neg p,q)$	1
$(\neg p,\neg q)$	1

(p,r)	1
$(p,\neg r)$	0
$(\neg p,r)$	1
$(\neg p,\neg r)$	1

(p,s)	1
$(p,\neg s)$	1
$(\neg p,s)$	0
$(\neg p,\neg s)$	0

,

Assume that MARGINALIZE1 *is applied and we are going to compute $(T_1 \otimes T_2)^{-p}, (T_1 \otimes T_3)^{-p}, (T_2 \otimes T_3)^{-p}$, given by the following tables:*

(q,r)	1
$(q,\neg r)$	1
$(\neg q,r)$	1
$(\neg q,\neg r)$	1

(q,s)	1
$(q,\neg s)$	1
$(\neg q,s)$	0
$(\neg q,\neg s)$	0

(r,s)	1
$(r,\neg s)$	1
$(\neg r,s)$	0
$(\neg r,\neg s)$	0

, (1)

Alternatively, we can compute $(T_1 \otimes T_2 \otimes T_3)^{-\{p,s\}}, (T_1 \otimes T_2 \otimes T_3)^{-\{p,r\}}, (T_1 \otimes T_2 \otimes T_3)^{-\{p,q\}}$, which are defined for the same variables, obtaining:

(q,r)	1
$(q,\neg r)$	1
$(\neg q,r)$	1
$(\neg q,\neg r)$	**0**

(q,s)	1
$(q,\neg s)$	1
$(\neg q,s)$	0
$(\neg q,\neg s)$	0

(r,s)	1
$(r,\neg s)$	1
$(\neg r,s)$	0
$(\neg r,\neg s)$	0

(2)

The product of these three tables is equal to the product of the original tables in Equation (1). However, individually, each one of them can be more informative (has more 0's) than the corresponding one in Equation (1). In this example, the first one has 0 assigned to $(\neg p, \neg q)$ in Equation (2), while this value was 1 in Equation (1). MARGINALIZE1B *will compute the arrays in Equation (2).*

5.6. The Final Global Marginalization Procedure

Now, we put everything together and describe the global marginalization procedure. In Algorithm 9, we describe the algorithm MARGINALIZEG, which gives a set H, and the variable v computes a set H' equivalent to H^{-v} and other set H'', such that H is equivalent to $H' \cup H''$. We assume that DETERMINED $(H_v$,$v)$ is a procedure that checks whether there is a table in $T \in H_v$ in which v is functionally determined. In that case, it returns MINDEP $(T, v, \{v\})$. If this table does not exist, it returns the neutral element T_e.

The first thing is to test whether there is a functional dependence. In that case, MARGINALIZE2 is applied, after minimizing the table with that dependence. If the size of the global combination is lower than a given threshold, then the version B of the marginalization is applied.

In another case, we have to choose between MARGINALIZE0 and MARGINALIZE1. For that, we first compute $M(R_1)$, the set of maximal sets of the sets of variables of the tables of R_1 after applying MARGINALIZE1 and compare the size of the tables defined for these sets with the size of the table obtained with the basic MARGINALIZE0, selecting the method with a smaller final size of the tables. If MARGINALIZE1 is selected, then GROUPTABLES is applied if the number of tables in H_v is greater than a given threshold N.

Algorithm 9 The final global marginalization algorithm

```
Require: $H$, a set of tables.
Require: $v$, the variable to remove
Require: $W$, a Boolean variable indicating the splitting procedure
Require: $N$, a threshold for grouping tables
Require: $K$, a threshold for the alternative deletion procedure
Ensure: $H'$, a set of tables representing $H^{-v}$.
Ensure: $H''$, a set of tables such that $H$ is equivalent to $H' \cup H''$.
 1: procedure MARGINALIZEG($H$,$v$)
 2:     $H_v \leftarrow$ tables $T \in H$ such that $v \in V(T)$
 3:     COMBINEINCLUDED ($H_v$)
 4:     if W then
 5:         SPLIT ($H_v, H$)
 6:     end if
 7:     $T \leftarrow$ DETERMINED ($H_v$,$v$)
 8:     if $T \neq T_e$ then
 9:         if $S(\otimes(H_v)) \leq K$ then
10:             $(R_1, R_2) \leftarrow$ MARGINALIZE2B ($H_v$,$v$,$T$)
11:         else
12:             $(R_1, R_2) \leftarrow$ MARGINALIZE2 ($H_v$,$v$,$T$)
13:         end if
14:     else
15:         $Q \leftarrow M(R_1)$                ▷ Maximal sets of $R_1$ after MARGINALIZE1
16:         if $S(\otimes(H_v)^{-v}) \leq S(Q)$ then
```

Algorithm 9 *Cont.*

```
17:             $(R_1, R_2) \leftarrow$ MARGINALIZE0 ($H_v$,$v$)
18:         else
19:             if $|H_v| \geq N$ then
20:                 GROUPTABLES ($H_v$)
21:             end if
22:             if $S(\otimes(H_v)) \leq K$ then
23:                 $(R_1, R_2) \leftarrow$ MARGINALIZE1B ($H_v$,$v$)
24:             else
25:                 $(R_1, R_2) \leftarrow$ MARGINALIZE1 ($H_v$,$v$)
26:             end if
27:         end if
28:     end if
29:     COMBINEINCLUDED ($R_1$)
30:     $H' \leftarrow H \setminus H_v \cup R_1$
31:     $H'' \leftarrow R_2$
32:     return $(H', H'')$
33: end procedure
```

6. Experiments

Computations with tables have been implemented using `Python` and the `NumPy` library [22]. For the basic operations with tables (combination, marginalization, and conditioning) we have taken as basis the implementation of the same operations in probabilistic calculus in the `pgmpy` library [23]. An important fact about the deletion algorithm is the order in which variables are chosen to be deleted. We have followed the usual heuristic of selecting the variable v with a minimum number of variables in H_v, but with the exception of the case in which v is not functionally determined in any table of H_v, but there is another variable v' that is determined in the table of $H_{v'}$, and where $|V(H_{v'}| - |V(H_v)| \leq 2$. This is done to have a preference for selecting variables in which we can apply MARGINALIZE2, which is the most efficient deletion procedure. We have carried out three experiments.

In the first experiment, we tested the basic deletion algorithms in several SAT examples imported from several repositories of benchmark problems for SAT, mainly from: Hoos and Stützle [24], Junttila [25], Tsuji and Gelder [26], and Burkardt [27]. The main criterion for selecting the cases has been that the size of the arrays does not surpass the maximum array size of 32 that exists in the `NumPy` library. The characteristics of the examples can be found in Table 1. We also provide the maximum cluster size (number of variables) of a join tree built from the connectivity graph under the same ordering than used in our deletion algorithm. We have selected the problems with the restrictions that they are not too simple (a maximum cluster size of at least 15) and that the deletion algorithm can be applied, taking into account that the number of dimensions of a table in `NumPy` is limited to 32, i.e., it is required that for each table T, we have that $|V(T)| \leq 32$.

Table 1. Benchmark problems used in the experiments.

Benchmark	Variables	Clauses	Max Cluster Size	Sol
SATHolV42C133.cnf	42	133	28	SAT
SATPlanV48C261.cnf	48	261	21	SAT
UNSATaimV50C80.cnf	50	80	16	UNSAT
UNSATaimV50C100.cnf	50	100	31	UNSAT
UNSATTmTbV112C245.cnf	112	245	18	UNSAT
SATTmTbV140C301.cnf	140	301	25	SAT
SATV300C1016.cnf	300	1010	28	SAT
aes_32_1_keyfind_1.cnf	300	1016	28	SAT
SATCircuitosV416C1136.cnf	416	1136	27	SAT
UNSATBFCircuitosV421C1000.cnf	421	1000	21	UNSAT
SATBFCircuitosV423C1010.cnf	423	1010	21	SAT
UNSATBFCircuitosV424C1031.cnf	424	1031	23	UNSAT
UNSATBFCircuitosV428C1037.cnf	428	1037	23	UNSAT
UNSATCircuitosV607C1808.cnf	607	1808	35	UNSAT
SATBFCircuitosV837C2169.cnf	837	2169	20	SAT
SATBFCircuitosV837C2169_2.cnf	837	2169	22	SAT
SATBFCircuitosV843C2286.cnf	843	2286	29	SAT
UNSATBFCircuitosV864C2790.cnf	864	2790	74	UNSAT
UNSATBFCircuitosV864C2790_2.cnf	864	2790	66	UNSAT
UNSATBFCircuitosV865C2783.cnf	865	2783	69	UNSAT
UNSATBFCircuitosV865C2783_2.cnf	865	2783	74	UNSAT
UNSATBFCircuitosV865C2784.cnf	865	2784	63	UNSAT

Table 1. *Cont.*

Benchmark	Variables	Clauses	Max Cluster Size	Sol
UNSATBFCircuitosV985C2324.cnf	985	2324	25	UNSAT
UNSATCircuitosV986C2315.cnf	986	2315	24	UNSAT
UNSATBFCircuitosV1040C3668_2.cnf	1040	3668	70	UNSAT
UNSATBFCircuitosV1040C3668.cnf	1040	3668	103	UNSAT
SATBFCircuitosV1207C2940.cnf	1207	2940	41	SAT
UNSATBFCircuitosV1339C3249.cnf	1339	3249	29	UNSAT
UNSATCircuitosV1355C3296.cnf	1355	3296	30	UNSAT
UNSATCircuitosV1359C3321.cnf	1359	3321	30	UNSAT
UNSATV1359C3321.cnf	1359	3321	30	UNSAT
UNSATBFCircuitosV1363C3361.cnf	1363	3361	31	UNSAT
UNSATBFCircuitosV1363C3361_2.cnf	1363	3361	27	UNSAT
UNSATBFCircuitosV1365C3369.cnf	1365	3369	27	UNSAT
UNSATBFCircuitosV1371C3383.cnf	1371	3383	31	UNSAT
UNSATBFCircuitosV1371C3383_2.cnf	1371	3383	26	UNSAT
UNSATBFCircuitosV1371C3401.cnf	1371	3401	30	UNSAT
UNSATBFCircuitosV1373C3391.cnf	1373	3391	33	UNSAT
UNSATBFCircuitosV1379C3417.cnf	1379	3417	26	UNSAT
UNSATBFCircuitosV1379C3417_2.cnf	1379	3417	33	UNSAT
UNSATBFCircuitosV1379C3423.cnf	1379	3423	29	UNSAT
UNSATBFCircuitosV1387C3439.cnf	1387	3439	27	UNSAT
UNSATBFCircuitosV1387C3439_2.cnf	1387	3439	27	UNSAT
UNSATBFCircuitosV1389C3440.cnf	1389	3440	26	UNSAT
UNSATBFCircuitosV1389C3440_2.cnf	1389	3440	25	UNSAT
UNSATCircuitosV1393C3434.cnf	1393	3434	26	UNSAT
UNSATBFCircuitosV1407C3496.cnf	1407	3496	26	UNSAT
UNSATBFCircuitosV1423C3609.cnf	1423	3609	38	UNSAT
UNSATBFCircuitosV1488C3859.cnf	1488	3859	21	UNSAT
SATCircuitosV1501C3575.cnf	1501	3575	26	SAT
SATCircuitosV2013C4795.cnf	2013	4795	29	SAT
UNSATCircuitosV2177C6768.cnf	2177	6768	65	UNSAT
UNSATCircuitosV2180C6778.cnf	2180	6778	70	UNSAT

We have applied the deletion algorithm with the general deletion step of Algorithm 9. In it, we have considered a value of $N = 30$ for grouping tables when pairwise marginalization is applied. We have tested two variants of the algorithm $W = True$ and $W = False$ to test whether splitting is a good idea and a set of values of $K = 5, 10, 15, 20, 25, 30$.

First, we can observe that, in general, the problems are solved fast, with an average of less than 1 min for all the parameters settings.

With a non-parametric Friedman test, the differences between the different combinations of K and W are significant (p-value = 0.001149). The averages of times as a functions of K are depicted in Figure 1 for the case $W = False$ (the case $W = True$ is very similar). We can see that the time increases as a function of K, but being more or less constant for $K = 10, 15, 20$. This increasing pattern does not always occur. For example, in aes_32_1_keyfind_1.cnf, the times with $W = False$ can be seen in Figure 2. In that case, it is possible to see that the optimal K is 15. The extra work of computing the global table is compensated by the presence of more zeros in the resulting tables, which speed up the consecutive deletions.

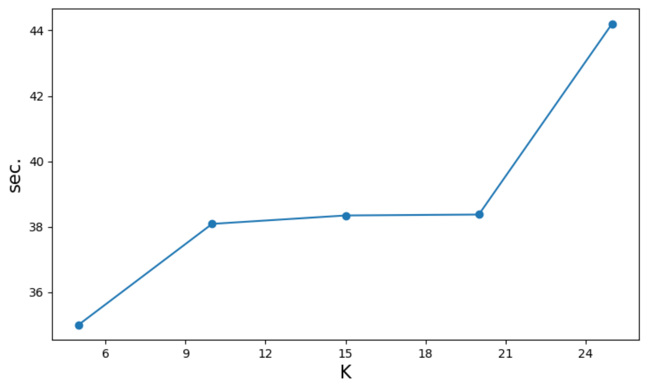

Figure 1. Average time as a function of K.

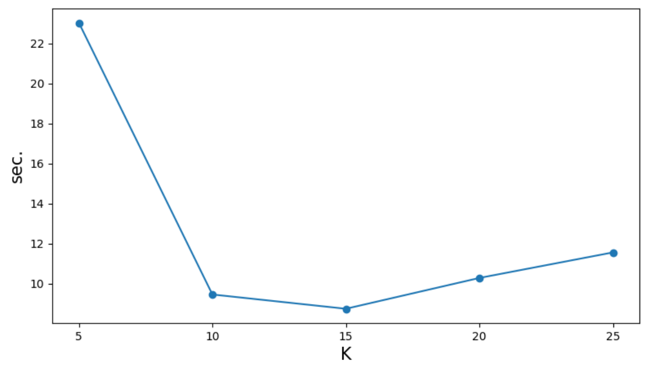

Figure 2. Time as a function of K in aes_32_1_keyfind_1.cnf.

We have carried out a post hoc Conoven–Friedman test (without correction for multiple tests). The resulting p-values for pairwise comparisons can be seen in Table 2. A 'Y' means that $W = True$ and an 'N' means that $W = False$. First, we can observe that there are never significant differences with the use of SPLIT. There are significant differences of $K = 25$ with the cases in which $K = 5, 10$, especially if SPLIT is not applied. In this case, the computation of large tables does not compensate for the presence of more zeros. The comparison of $K = 20$ is only significant with $K = 5$. Again, the significance is higher with no SPLIT. $K = 15$ with SPLIT shows no significant difference with any smaller value of K. There is only one small significance (with $\alpha = 0.05$) if split is not applied.

To compare with previous versions of the DP algorithm, we have also implemented the basic Algorithm 2, as in [5], including a step for unit propagation (each time a unit clause is found, unit propagation is carried out). As this algorithm is less efficient, we have given it a limit of 600 s to solve each problem to avoid very long running times in some of the problems. For example, case aes_32_1_keyfind_1.cnf was not solved even with 2 h of running time. A total of 12 problems could not be solved within this time limit (600 s), and the average time was 168.08 s. This average is much larger than the worst case in our algorithms, even taking into account that the running time was limited to 600 s. A non-parametric Wilcoxon test was not significant. This is due to the fact that this approach was usually faster in the simpler problems.

In the second experiment, we consider four sets of clauses with 504, 708, 912, and 1116 variables and 1840, 2664, 3488, and 4312 clauses, respectively. Our exact algorithms were not able to solve these cases. However, we have computed the number of variables which have been deleted without surpassing a maximum table size of 25, for each combination of K and $W = True$ (the results were the same for $W = False$). We can observe that there is a tendency to delete, in an exact way, more variables when K increases, though this is not always true in each particular problem (Table 3).

Table 2. Results of the post hoc Conoven–Friedman test.

	K05N	**K05Y**	**K10N**	**K10Y**	**K15N**	**K15Y**	**K20N**	**K20Y**	**K25N**	**K25Y**
K05N	1.000000	1.000000	0.319196	0.974352	0.012449	0.403390	0.003590	0.007856	0.006494	0.067368
K05Y	1.000000	1.000000	0.319196	0.974352	0.012449	0.403390	0.003590	0.007856	0.006494	0.067368
K10N	0.319196	0.319196	1.000000	0.303853	0.131254	0.872292	0.054216	0.095066	0.083050	0.403390
K10Y	0.974352	0.974352	0.303853	1.000000	0.011374	0.385562	0.003242	0.007146	0.005897	0.062719
K15N	0.012449	0.012449	0.131254	0.011374	1.000000	0.095066	0.676013	0.872292	0.821945	0.499687
K15Y	0.403390	0.403390	0.872292	0.385562	0.095066	1.000000	0.037087	0.067368	0.058339	0.319196
K20N	0.003590	0.003590	0.054216	0.003242	0.676013	0.037087	1.000000	0.797032	0.847040	0.274662
K20Y	0.007856	0.007856	0.095066	0.007146	0.872292	0.067368	0.797032	1.000000	0.948731	0.403390
K25N	0.006494	0.006494	0.083050	0.005897	0.821945	0.058339	0.847040	0.948731	1.000000	0.368225
K25Y	0.067368	0.067368	0.403390	0.062719	0.499687	0.319196	0.274662	0.403390	0.368225	1.000000

Table 3. Number of deleted variables.

	K05	**K10**	**K15**	**K20**	**K25**
aes_32_2_keyfind_1.cnf	412	421	421	417	430
aes_32_3_keyfind_1.cnf	570	580	580	576	576
aes_32_4_keyfind_1.cnf	726	716	716	734	732
aes_32_5_keyfind_1.cnf	873	864	865	882	880
TOTAL	2581	2581	2582	2609	2618

As a summary of the first two experiments, we find that the use of $K = 5$ with SPLIT is the best option, but we leave open the possibility of using higher values of K, especially in difficult problems.

The third experiment identifies a situation in which marginalization algorithms can be applied. For that, we consider the Bayesian networks used in the UAI 2022 competition (see https://uaicompetition.github.io/uci-2022/ (accessed on 10 May 2023)). We discarded the networks in which all the values in the conditional probability tables are non-zero (five networks) and the networks in which there were non-binary variables (three networks). This makes a total of 96 networks used in the partition function and the marginal probability competitions. The characteristics of these networks can be seen in Table 4, where nv is the number of variables, ne is the number of observed nodes, mcs is the maximum cluster size (in a deletion algorithm with the original probability tables), and $p0$ is the percentage of 0 values in the original conditional probability tables. We can observe that the percentage of 0 values is very high in all the networks. We have to take into account that, being conditional probability tables for binary variables, at least 0.5 of the values are different from 0 (for each 0 value, there is another value equal to 1). Thus, in some networks with 0.455 values that are 0, this implies that only 0.002% of the values are different from 0 and 1. With these networks, we have solved the associated propositional problem. This problem is defined by transforming each conditional probability table T into a logical table T' in such a way that $T'(\mathbf{v}) = 0$ if $T(\mathbf{v}) = 0.0$ and $T'(\mathbf{v}) = 1$, otherwise. If a variable v has been observed, then a unitary logical table is added with value 1 in the observed value and 0 in the unobserved one.

Then, we have applied our basic algorithm with $K = 20$ and SPLIT. All the problems were solved with an average time of 20.54 s (minimum of 0.035 s and maximum of 542.367 s). It is important to remark that most of the problems, 59 out of 96, were solved in less than 1 s, and almost all of them, 82 out of 96, were solved within a time limit of 10 s.

Table 4. Benchmarks used in Experiment 3.

Benchmark	*nv*	*ne*	*mcs*	*p***0**	**Benchmark**	*nv*	*ne*	*mcs*	*p***0**
Promedus_11	461	8	32	0.358	mastermind_04_08_03-0014	1418	48	24	0.491
Promedus_12	534	3	26	0.366	mastermind_04_08_03-0015	1418	48	24	0.491
Promedus_13	894	4	11	0.325	mastermind_04_08_04-0001	2616	36	39	0.494
Promedus_14	414	9	37	0.363	mastermind_04_08_04-0002	2616	36	39	0.494
Promedus_15	385	4	14	0.369	mastermind_04_08_04-0003	2616	36	39	0.494
Promedus_16	715	5	19	0.302	mastermind_04_08_04-0004	2616	36	39	0.494
Promedus_17	916	9	29	0.307	mastermind_04_08_04-0005	2616	36	39	0.494
Promedus_18	374	8	39	0.369	mastermind_05_08_03-0001	1616	27	28	0.490
Promedus_19	624	7	28	0.347	mastermind_05_08_03-0002	1616	27	28	0.490
Promedus_20	546	6	23	0.350	mastermind_05_08_03-0003	1616	27	28	0.490
Promedus_21	473	3	13	0.370	mastermind_05_08_03-0004	1616	27	28	0.490
Promedus_22	400	3	15	0.3339	mastermind_05_08_03-0009	1616	63	35	0.490
Promedus_23	674	9	28	0.320	mastermind_06_08_03-0002	1814	27	33	0.489
Promedus_24	200	4	5	0.294	mastermind_06_08_03-0003	1814	27	33	0.489
Promedus_25	1005	7	27	0.312	mastermind_06_08_03-0005	1814	27	33	0.489
Promedus_26	614	6	4	0.298	mastermind_06_08_03-0009	1814	83	41	0.489
Promedus_27	410	5	22	0.357	mastermind_10_08_03-0008	2606	769	57	0.486
Promedus_28	463	7	19	0.350	mastermind_10_08_03-0009	2606	404	56	0.486
Promedus_29	434	8	5	0.301	or_chain_4.fg	700	9	44	0.348
Promedus_30	306	13	7	0.298	or_chain_12.fg	468	11	42	0.361
Promedus_31	466	2	14	0.370	or_chain_15.fg	656	11	39	0.347
Promedus_32	511	2	13	0.329	or_chain_53.fg	741	13	45	0.348
Promedus_33	378	2	6	0.303	or_chain_61.fg	1028	7	46	0.336
Promedus_34	415	3	22	0.361	or_chain_64.fg	460	9	38	0.358
Promedus_35	467	2	14	0.371	or_chain_90.fg	512	9	42	0.357
Promedus_36	467	2	14	0.371	or_chain_102.fg	860	11	45	0.325
Promedus_37	1039	4	27	0.330	or_chain_106.fg	695	10	41	0.344
Promedus_38	668	5	36	0.353	or_chain_107.fg	631	11	50	0.349
BN_31	1156	120	66	0.448	or_chain_128.fg	648	11	54	0.352
fs-07	1225	1120	36	0.440	or_chain_132.fg	723	13	43	0.344
mastermind_03_08_04-0000	2288	0	31	0.495	or_chain_138.fg	702	11	40	0.350
mastermind_03_08_04-0001	2288	36	31	0.495	or_chain_140.fg	1268	8	43	0.331
mastermind_03_08_04-0002	2288	36	31	0.495	or_chain_149.fg	629	10	38	0.352
mastermind_03_08_04-0003	2288	36	31	0.495	or_chain_150.fg	928	6	38	0.347
mastermind_03_08_04-0007	2288	245	33	0.495	or_chain_153.fg	710	9	35	0.347
mastermind_03_08_04-0008	2288	355	32	0.495	or_chain_155.fg	542	11	42	0.356
mastermind_03_08_04-0010	2288	142	32	0.495	or_chain_161.fg	794	8	46	0.350
mastermind_03_08_04-0012	2288	64	31	0.495	or_chain_188.fg	1061	11	38	0.308
mastermind_03_08_04-0013	2288	64	31	0.495	or_chain_209.fg	859	10	38	0.341
mastermind_03_08_04-0014	2288	64	31	0.495	or_chain_242.fg	613	10	43	0.360

Table 4. *Cont.*

Benchmark	*nv*	*ne*	*mcs*	*p*0	Benchmark	*nv*	*ne*	*mcs*	*p*0
mastermind_03_08_04-0015	2288	64	31	0.495	blockmap_10_01-0009	5650	1078	56	0.499
mastermind_03_08_05-0001	3692	45	41	0.496	blockmap_10_02-0009	6252	1159	56	0.499
mastermind_03_08_05-0003	3692	45	39	0.496	blockmap_10_03-0009	6848	1844	60	0.499
mastermind_03_08_05-0009	3692	785	42	0.496	blockmap_10_03-0010	6848	1610	59	0.499
mastermind_04_08_03-0000	1418	0	24	0.491	blockmap_15_01-0008	16,497	6193	69	0.499
mastermind_04_08_03-0011	1418	48	24	0.491	blockmap_15_02-0008	17,649	5959	81	0.499
mastermind_04_08_03-0012	1418	48	24	0.491	blockmap_15_03-0010	18,787	6273	73	0.499
mastermind_04_08_03-0013	1418	48	24	0.491	blockmap_20_01-0009	39,297	15,222	105	0.499

We have also applied the DP algorithm (Algorithm 2) to the equivalent set of clauses of the different problems. The average time was 200.736 s, but we need to take into account that there was a time limit of 2000 s and that nine problems were not solved within this time limit. The average, after taking into account the full resolution of these nine problems, would have been higher. This shows that our method was able of solving more problems and with less time, especially for difficult cases.

It is important to remark that the result of the deletion algorithms can be used to develop Monte Carlo algorithms to obtain compatible configurations according to Proposition 4, without rejecting any configuration due to a 0 probability. This is important for the development of approximate algorithms and it is not feasible with classical SAT algorithms deciding the satisfiability of the case and providing one satisfying assignment.

7. Conclusions and Future Work

In this paper, we have proposed a new procedure which can be applied to solve the marginalization problem in propositional logic based on the use of Boolean arrays. The experiments show that it is possible to solve, in an exact way, moderate-sized problems (even with thousands of variables and a tree width of more than 100). The method is based on a classical deletion algorithm but with some improvements based on the special characteristics of Boolean arrays. We have provided a full set of tools for working and operating with these arrays, allowing us to apply the Shenoy–Shafer abstract framework [2]. We have studied different methods for carrying out the deletion of a variable. Of special interest is the deletion of a variable when its value is functionally determined from the values of the other variables. Previous experiments with deletion algorithms [4,5] reported experiments in general problems, with up to 25 variables. They have also reported experiments with other problems, called 'chain problems', that were randomly generated in such a way that the tree width was bounded by 5, while we have been able of solving problems with a tree width of 103. We have shown that we have been able to expand the class of problems that can be solved with the deletion algorithm, as the previous versions were unable of solving the more difficult problems in our experiments. Some problems which could be solved in seconds with our approach could not be solved in hours with former basic deletion algorithm. This opens a new set of possibilities for algorithms to solve the marginal problem.

For the future, this framework opens a wide range of possible developments:

- To optimize the deletion strategy by selecting the most appropriate method depending on the characteristics of the tables in H_v.
- To improve the methods for decomposing a large table T as a product of smaller tables, which can be more useful. Of special interest is also to obtain tables of low dimensions, even if T is not decomposed as its product. The extreme case is a table of dimension 1, which always simplifies the problem (if different from the neutral element).

- To combine the clause representation and the table representation, as we have found that the basic clause representation was able of solving faster the simpler problems.
- To organize computations in a join tree with a local computation algorithm [2,6].
- To develop approximate algorithms. For example, the mini-bucket elimination algorithm [28] is a general framework for approximate algorithms based on partitioning the set H_v before carrying out the combination of all the tables. In this case, we can have more opportunities based on the fact that potentials are idempotent and the alternative marginalization procedures we have provided.
- To combine approximate and exact computations. An approximate computation can be the basis to obtain small informative tables, which can be useful to speed up an exact algorithm.
- To develop a backtracking algorithm, taking as a basis the array representation.
- To approximate inference in Bayesian networks is NP-hard, especially when there are extreme probabilities [29]. Approximate algorithms, such as likelihood weighting [30] or penniless propagation [31], could take advantage of a previous and fast propagation of 0–1 values with the procedures proposed in this paper. Experiment 3 has already shown that our algorithms can propagate 0–1 values very fast in hard problems (from the UAI 2022 competition).
- The special role of potentials representing functional dependence could also be studied in the general framework of Information Algebras [32] and applied to other problems which are particular cases, such as constraint satisfaction [33].
- To consider the easy deletion of some variables in an SAT problem as a preprocessing step in SAT problems [34] that could be used as a simplification procedure.

Author Contributions: Conceptualization, E.D.-M. and S.M.; methodology, E.D.-M. and S.M.; software, E.D.-M. and S.M.; validation, E.D.-M. and S.M.; writing—original draft preparation, E.D.-M. and S.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported under project PID2019.106758GB.C31 funded by MCIN/AEI/10.13039/501100011033.

Data Availability Statement: The library with programs together with data to reproduce experiments are available as a Github repository at: https://github.com/serafinmoral/SAT-solver (accessed on 16 May 2023).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Proofs

Proposition A1. *If T is a table and $v \in V(T)$, then $T^{-v} = U(T, v) \oplus U(T, \neg v)$.*

Proof. The result is immediate, since if $W = V(T) \setminus \{v\}$, we have that:

$T^{-v}(\mathbf{w}) = \max\{T(\mathbf{w}, v), T(\mathbf{w}, \neg v)\} = \max\{U(T, v)(\mathbf{w}), U(T, \neg v)(\mathbf{w})\} = (U(T, v) \oplus U(T, \neg v))(\mathbf{w})$. □

Proposition A2. *If T_1 and T_2 are tables and $v \in V(T_1) \cap V(T_2)$, then $(T_1 \oplus T_2)^{-v} = T_1^{-v} \oplus T_2^{-v}$.*

Proof. The proof is immediate taking into account that being $W = (V(T_1) \cup V(T_2)) \setminus \{v\}$, we have:

$$\begin{aligned}
&T_1^{-v} \oplus T_2^{-v}(\mathbf{w}) = \max\{T_1^{-v}(\mathbf{w}), T_2^{-v}(\mathbf{w})\} = \\
&\max\{\max\{T_1(\mathbf{w}, v), T_1(\mathbf{w}, \neg v)\}, \max\{T_2(\mathbf{w}, v), T_2(\mathbf{w}, \neg v)\}\} = \\
&\max\{\max\{T_1(\mathbf{w}, v), T_2(\mathbf{w}, v)\}, \max\{T_1(\mathbf{w}, \neg v), T_2(\mathbf{w}, \neg v)\}\} = \\
&\max\{T_1 \oplus T_2(\mathbf{w}, v), T_1 \oplus T_2(\mathbf{w}, \neg v)\} = (T_1 \oplus T_2)^{-v}(\mathbf{w}). \quad \square
\end{aligned}$$

Proposition A3. *In each application of* MARGINALIZE0(H_v)*, we have that H_v is equivalent to $R_1 \cup R_2$. Furthermore, if we call H' to the update set H computed in Step 8 of Algorithm 2, then H is equivalent to $H' \cup R_2$.*

Proof. R_2 is the combination of all the tables in H_v, and therefore, it is equivalent to it. R_1 is the marginalization of T_1, which is the combination of tables in H_v, and therefore, it is less informative than H_v. The union of a set equivalent to H_v and a set that is less informative than H_v produces a set which is equivalent to H_v.

For the second part, $H' = (H \setminus H_v) \cup R_1$; therefore, $H' \cup R_2 = (((H \setminus H_v) \cup R_2) \cup R_1)$. If we remove from H a set of tables (H_v) and replace them by its combination (R_2), then we obtain a set which is equivalent to the original set H. On the other hand, R_1 only contains a table which is less informative than H, and therefore, if we add it, we obtain a set which is equivalent to H. $\square$

Proposition A4. *Assume a set of tables H and that Algorithm 2 is applied removing variables in $V(H)$ in order $(v_1, \ldots, v_k)$. Assume also that $R_2[\emptyset]$ is equivalent to the empty set, i.e., the problem is satisfiable; then, if $\mathbf{T}_i$ is the set of true assignments satisfying set of tables $H^{-\{v_1,\ldots,v_i\}}$ and $\mathbf{T}_0$ is the true assignments of H, then these sets can be computed in reverse order of $i = 1, \ldots, k$ in the following way:*

- *Start with $\mathbf{T}_i = \emptyset$.*
- *Make $\mathbf{T}_k$ equal to the set containing an empty vector $\mathbf{v}_0 \in \Omega_\emptyset$.*
- *For each $\mathbf{v}_{i+1} \in \mathbf{T}_{i+1}$, compute T_{i+1}, the only table in $U(R_2[v_{i+1}], \mathbf{v}_{i+1})$, which is a table defined only for variable v_{i+1}. Then, this table is never equal to T_0, and if $\mathbf{v}_i^1$ and $\mathbf{v}_i^2$ are the true assignments obtained by extending $\mathbf{v}_{i+1}$ to variables $\{v_{i+1}, \ldots, v_k\}$ and given by $\mathbf{v}_i^1 = (v_{i+1}, \mathbf{v}_{i+1}), \mathbf{v}_i^2 = (\neg v_{i+1}, \mathbf{v}_{i+1})$, i.e., by considering v_{i+1} true and false, respectively, then:*
  - *if $T_{i+1} = T_e$, add $\mathbf{v}_i^1$ and $\mathbf{v}_i^2$ to $\mathbf{T}_i$.*
  - *if $T_{i+1}(v_{i+1}) = 1, T_{i+1}(\neg v_{i+1}) = 0$, add $\mathbf{v}_i^1$ to $\mathbf{T}_i$.*
  - *if $T_{i+1}(v_{i+1}) = 0, T_{i+1}(\neg v_{i+1}) = 1$, add $\mathbf{v}_i^2$ to $\mathbf{T}_i$.*

Proof. We have that $H^{-\{v_1,\ldots,v_{i+1}\}}$ is equivalent to $(H^{-\{v_1,\ldots,v_i\}})^{-v_{i+1}}$, which is equivalent to $U(H^{-\{v_1,\ldots,v_i\}}), v_{i+1}) \oplus U(H^{-\{v_1,\ldots,v_i\}}), \neg v_{i+1})$; then, we have that $\mathbf{v}_{i+1} \in \mathbf{T}_{i+1}$ if and only if

$\mathbf{v}_{i+1} \in (\mathbf{T}(U(H^{-\{v_1,\ldots,v_i\}}), v_{i+1})) \cup \mathbf{T}(U(H^{-\{v_1,\ldots,v_i\}}), \neg v_{i+1})))$.

Therefore, we have that:

$\mathbf{T}_{i+1} = \{\mathbf{v}_i^{-v_{i+1}} : \mathbf{v}_i \in \mathbf{T}_i\}$.

Now, we take into account that $H^{-\{v_1,\ldots,v_i\}}$ is equivalent to $H^{-\{v_1,\ldots,v_{i+1}\}} \cup R_2[v_{i+1}]$, and then $\mathbf{v}_i \in \mathbf{T}(H^{-\{v_1,\ldots,v_{i+1}\}} \cup R_2[v_{i+1}])$ if and only if this assignment satisfies both $H^{-\{v_1,\ldots,v_{i+1}\}}$ and $R_2[v_{i+1}]$.

To satisfy $H^{-\{v_1,\ldots,v_{i+1}\}}$, the condition is that there is $\mathbf{v}_{i+1} \in \mathbf{T}_{i+1}$ such that $\mathbf{v}_i^{-v_{i+1}} = \mathbf{v}_{i+1}$, i.e., that $\mathbf{v}_i = \mathbf{v}_i^1$ or $\mathbf{v}_i = \mathbf{v}_i^2$ for $\mathbf{v}_{i+1} \in \mathbf{T}_{i+1}$.

To satisfy, $R_2[v_{i+1}]$, we have that if T is the only table in $R_2[v_{i+1}]$, we have that $T(\mathbf{v}_i^1) = 1$ when $U(T, \mathbf{v}_{i+1})(v_{i+1}) = 1$, i.e., $T_{i+1}(v_{i+1}) = 1$ and $T(\mathbf{v}_i^2) = 1$ when $U(T, \mathbf{v}_{i+1})(\neg v_{i+1}) = 1$, i.e., $T_{i+1}(\neg v_{i+1}) = 1$. Furthermore, these are the conditions that are checked to introduce $\mathbf{v}_i^1$ and/or $\mathbf{v}_i^2$ in $\mathbf{T}_i$.

T_i can never be equal to T_0, since for any $\mathbf{v}_{i+1} \in \mathbf{T}_{i+1}$, we must have that either $\mathbf{v}_i^1$ or $\mathbf{v}_i^2$ is in $\mathbf{T}_i$. $\square$

Proposition A5. *If H_v is a set of tables containing variables v, then R_1 computed in Algorithm 4 represents H_v^{-v}.*

Proof. We have to prove that R_1 represents $\otimes(H_v)^{-v}$.

Each table $(T_i \otimes T_j)^{-v} \in R_1$ is such that $T_i \otimes T_j \preceq \otimes(H_v)$, and therefore, $(T_i \otimes T_j)^{-v} \preceq (\otimes(H_v))^{-v}$. Then, $R_1 \preceq \{(\otimes(H_v))^{-v}\}$.

On the other hand:

$$(\otimes(H_v))^{-v} = U(\otimes(H_v), v) \oplus U(\otimes(H_v), \neg v)$$

Thus, if $\mathbf{v}' \in \Omega_{V(H_v)\setminus\{v\}}$, we have that if $\otimes(H_v))^{-v}(\mathbf{v}') = 0$, then for $\ell = v$ and $l = \neg v$, $U(\otimes(H_v), \ell)) = 0$ and $\otimes(U(H_v, \ell))(\mathbf{v}') = 0$. Since:

$$\otimes(U(H_v, \ell))(\mathbf{v}') = \min_{T_i \in H_v} U(T_i, \ell)(\mathbf{v}'),$$

we have that for $\ell = v$, there is $T_i \in H_v$ with $U(T_i, v)(\mathbf{v}') = 0$, and for $\ell = \neg v$, there is $T_j \in H_v$ with $U(T_j, \neg v)(\mathbf{v}') = 0$. Since $U(T_i, v)(\mathbf{v}') \geq (U(T_i, v) \otimes U(T_j, v))(\mathbf{v}')$ and $U(T_j, v)(\mathbf{v}') \geq (U(T_i, v) \otimes U(T_j, v))(\mathbf{v}')$, then $U(T_i \otimes T_j, v) = 0$ and $U(T_i \otimes T_j, \neg v) = 0$.

Since $(T_i \otimes T_j)^{-v} = U(T_i \otimes T_j|v) \oplus U(T_i \otimes T_j|\neg v)$, we also have $(T_i \otimes T_j)^{-v}(\mathbf{v}') = 0$, and, taking into account that $(T_i \otimes T_j)^{-v} \in R_1$, we have that $\otimes(R_1)(\mathbf{v}') = 0$.

We have proved that if $(\otimes(H_v))^{-v}(\mathbf{v}') = 0$, then we have that $\otimes(R_1)(\mathbf{v}') = 0$, and as a consequence, $(\otimes(H_v))^{-v} \preceq \otimes(R_1)$, and we have the other inequality and the equivalence. □

Proposition A6. *If (R_1, R_2) are the sets computed in* MARGINALIZE2 *and the initial conditions required by the algorithm are satisfied, then H_v^{-v} is equivalent to R_1 and H_v is equivalent to $R_1 \cup R_2$.*

Proof. To prove that H_v^{-v} is equivalent to R_1, we only has to prove that for any $T_i, T_j \in H_v$, then $(T_i \otimes T_j)^{-v} \preceq (T \otimes T_i)^{-v} \otimes (T \otimes T_i)^{-v}$. We have:

$$\begin{aligned}
&(T \otimes T_i)^{-v} \otimes (T \otimes T_i)^{-v} = \\
&(U(T \otimes T_i, v) \oplus U(T \otimes T_i, \neg v)) \otimes (U(T \otimes T_j, v) \oplus U(T \otimes T_j, \neg v)) = \\
&(U(T \otimes T_i, v) \otimes U(T \otimes T_j, v)) \oplus (U(T \otimes T_i, \neg v) \otimes U(T \otimes T_j, \neg v)),
\end{aligned}$$

where the last inequality comes from the fact that $(U(T \otimes T_i, v) \otimes (T \otimes T_j, \neg v))$ is more informative than $(U(T, v) \otimes (T, \neg v))$, which is equivalent to T_0, and then $(U(T \otimes T_i, v) \otimes (T \otimes T_j, \neg v))$ is also equivalent to T_0 and can be removed from a disjunction.

Thus, if R_1' is the set of tables computed with MARGINALIZE1, then R_1' is less informative than R_1. On the other hand, since $R_1 \subseteq R_1'$, then R_1 is less informative than R_1' and the two sets are equivalent.

Now, we have to prove that $R_1 \cup R_2$ is equivalent to H_v. $R_1 \cup R_2$ is less informative than H_v as R_1 and R_2 are contained in the sets R_1', R_2', computed with MARGINALIZE1.

Consider $V' = V(H_u) \setminus \{v\}$. If $\otimes(H_v)(\mathbf{v}', \ell) = 0$, where $\ell = v$ or $\ell = \neg v$, we can have the following situations:

- If $\otimes(H_v)(\mathbf{v}', \neg\ell)$ is also 0, then $\otimes(H_v)^{-v}(\mathbf{v}') = 0$ and then $\otimes(R_1)(\mathbf{v}') = \otimes(R_1)(\mathbf{v}', \ell) = 0$.
- If $\otimes(H_v)(\mathbf{v}', \neg\ell) = 1$, we have that $T(\mathbf{v}', \neg\ell) = 1$ as T is an element from H_v, and given that T determines v, we must have $T(\mathbf{v}', \ell) = 0$, and as T is the only element from R_2, obviously, $\otimes(R_2)(\mathbf{v}', \ell) = 0$.

As a consequence if $\otimes(H_v)(\mathbf{v}', \ell) = 0$, we always have $\otimes(R_1 \cup R_2)(\mathbf{v}', \ell) = 0$, and $R_1 \cup R_2$ is also more informative than H_v, which are, finally, shown to be equivalent. □

References

1. Davis, M.; Putnam, H. A Computing Procedure for Quantification Theory. *J. ACM* **1960**, *7*, 201–215. [CrossRef]
2. Shafer, G.; Shenoy, P. *Local Computation in Hypertrees*; Working Paper N. 201; School of Business, University of Kansas: Lawrence, KS, USA, 1988.
3. Kohlas, J. *Information Algebras: Generic Structures for Inference*; Springer Science & Business Media: London, UK 2012.

4. Dechter, R.; Rish, I. Directional resolution: The Davis-Putnam procedure, revisited. In *Principles of Knowledge Representation and Reasoning*; Morgan Kaufmann: San Francisco, CA, USA, 1994; pp. 134–145.
5. Risht, I.; Dechter, R. Resolution versus Search: Two Strategies for SAT. *J. Autom. Reason.* **2000**, *24*, 225–275. [CrossRef]
6. Kohlas, J.; Haenni, R.; Moral, S. Propositional information systems. *J. Log. Comput.* **1999**, *9*, 651–681. [CrossRef]
7. Lauritzen, S.L.; Spiegelhalter, D.J. Local computations with probabilities on graphical structures and their application to expert systems. *J. R. Stat. Soc. Ser. B* **1988**, *50*, 157–194. [CrossRef]
8. Hernández, L.D.; Moral, S. Inference with idempotent valuations. In Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, Providence, RI, USA, 1–3 August 1997; pp. 229–237.
9. Biere, A.; Heule, M.; van Maaren, H.; Walsh, T. *Handbook of Satisfiability*, 2nd ed.; Part I and II; IOS Press: Amsterdam, The Netherlands, 2021.
10. Cook, S.A. The complexity of theorem-proving procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing, Shaker Heights, OH, USA, 3–5 May 1971; pp. 151–158.
11. Biere, A. Bounded Model Checking. In *Handbook of Satisfiability*, 2nd ed.; Part II; IOS Press: Amsterdam, The Netherlands, 2009; pp. 739–764.
12. Rintanen, J. Planning and SAT. In *Handbook of Satisfiability*, 2nd ed.; Part II; IOS Press: Amsterdam, The Netherlands, 2021; pp. 765–789.
13. Björk, M. Successful SAT encoding techniques. *J. Satisf. Boolean Model. Comput.* **2011**, *7*, 189–201. [CrossRef]
14. Davis, M.; Logemann, G.; Loveland, D. A Machine Program for Theorem-proving. *Commun. ACM* **1962**, *5*, 394–397. [CrossRef]
15. Pipatsrisawat, K.; Darwiche, A. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.* **2011**, *175*, 512–525. [CrossRef]
16. Simon, L. Reasoning with propositional logic: From sat solvers to knowledge compilation. In *A Guided Tour of Artificial Intelligence Research: Volume II: AI Algorithms*; Springer Nature: Cham, Switzerland, 2020; pp. 115–152.
17. Knuth, D.E. *The Art of Computer Programming. Volume 4, Fascicle 6. Satisfiability*; Addison-Wesley: Boston, MA, USA, 2015.
18. Gomes, C.P.; Sabharwal, A.; Selman, B. Model counting. In *Handbook of Satisfiability*; IOS Press: Amsterdam, The Netherlands, 2021; pp. 993–1014.
19. Gogate, V.; Dechter, R. Approximate counting by sampling the backtrack-free search space. In Proceedings of the AAAI, Vancouver, BC, Canada, 22–26 July 2007; pp. 198–203.
20. Eén, N.; Biere, A. Effective preprocessing in SAT through variable and clause elimination. In Proceedings of the International Conference on Theory and Applications of Satisfiability Testing, Scotland, UK, 19–23 June 2005; Springer: Berlin/Heidelberg, Germany, 2005; pp. 61–75.
21. Tseitin, G.S. On the complexity of derivation in propositional calculus. In *Automation of Reasoning*; Springer: Berlin/Heidelberg, Germany, 1983; pp. 466–483.
22. Harris, C.R.; Millman, K.J.; van der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J.; et al. Array programming with NumPy. *Nature* **2020**, *585*, 357–362. [CrossRef] [PubMed]
23. Ankan, A.; Panda, A. pgmpy: Probabilistic graphical models using python. In Proceedings of the 14th Python in Science Conference (SCIPY 2015), Austin, TX, USA, 6–12 July 2015; pp. 6–11.
24. Hoos, H.H.; Stützle, T. SATLIB: An online resource for research on SAT. *Sat* **2000**, *2000*, 283–292.
25. Junttila, T. Tools for Constrained Boolean Circuits. Available online: https://users.ics.aalto.fi/tjunttil/circuits/ (accessed on 22 May 2022).
26. Tsuji, Y.; Gelder, A.V. Instances Selected for the Second DIMACS Challenge for Circuit Fault Analysis. Available online: https://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/DIMACS/BF/descr.html (accessed on 22 May 2022).
27. Burkardt, J. CNF Files. Available online: https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html (accessed on 22 May 2022).
28. Dechter, R.; Rish, I. Mini-buckets: A general scheme for bounded inference. *J. ACM* **2003**, *50*, 107–153. [CrossRef]
29. Roth, D. On the hardness of approximate reasoning. *Artif. Intell.* **1996**, *82*, 273–302. [CrossRef]
30. Fung, R.; Chang, K.C. Weighing and integrating evidence for stochastic simulation in Bayesian networks. In *Machine Intelligence and Pattern Recognition*; Elsevier: Amsterdam, The Netherlands, 1990; Volume 10, pp. 209–219.
31. Cano, A.; Moral, S.; Salmerón, A. Penniless propagation in join trees. *Int. J. Intell. Syst.* **2000**, *15*, 1027–1059. [CrossRef]
32. Kohlas, J. Algebras of information. A new and extended axiomatic foundation. *arXiv* **2017**, arXiv:1701.02658.
33. Dechter, R. *Constraint Processing*; Morgan Kaufmann: San Francisco, CA, USA, 2003.
34. Biere, A.; Järvisalo, M.; Kiesl, B. Preprocessing in SAT Solving. In *Handbook of Satisfiability*; IOS Press: Amsterdam, The Netherlands, 2021; Volume 336, pp. 391–435.