Boosting Definability Bipartition Computation Using SAT Witnesses

Jean-Marie Lagniez, Pierre Marquis

Boosting Definability Bipartition Computation using SAT Witnesses

Jean-Marie Lagniez[1] [0000−0002−6557−4115]
and Pierre Marquis[1,2] [0000−0002−7979−6608]

[1] Univ. Artois, CNRS, CRIL, Lens, France
[2] Institut Universitaire de France
{lagniez,marquis}@cril.fr

Abstract. Bipartitioning the set of variables $Var(\Sigma)$ of a propositional formula Σ w.r.t. definability consists in pointing out a bipartition $\langle I, O\rangle$ of $Var(\Sigma)$ such that Σ defines the variables of O (outputs) in terms of the variables in I (inputs), i.e., for every $o \in O$, there exists a formula Φ_o over I such that $o \Leftrightarrow \Phi_o$ is a logical consequence of Σ. The existence of Φ_o given o, I, and Σ is a coNP-complete problem, and as such, it can be addressed in practice using a SAT solver. From a computational perspective, definability bipartitioning has been shown as a valuable preprocessing technique for model counting, a key task for a number of AI problems involving probabilities. To maximize the benefits offered by such a preprocessing, one is interested in deriving *subset-minimal bipartitions* in terms of input variables, i.e., definability bipartitions $\langle I, O\rangle$ such that for every $i \in I$, $\langle I \setminus \{i\}, O \cup \{i\}\rangle$ is not a definability bipartition. We show how the computation of subset-minimal bipartitions can be boosted by leveraging not only the decisions furnished by SAT solvers (as done in previous approaches), but also the SAT witnesses (models and cores) justifying those decisions.

Keywords: Automated reasoning including satisfiability checking and its extensions, definability, propositional logic

1 Introduction

In this paper, we are interested in identifying *definability relations* between variables occurring in a given propositional formula Σ. When I is a subset of the variables of Σ and o is a variable of Σ, Σ is said to define o in terms of I if and only if (1) there exists a formula Φ_o over I such that $o \Leftrightarrow \Phi_o$ is a logical consequence of Σ. o can be considered as an *output* variable provided that the variables in I are viewed as *input* variables, since Σ defines o in terms of I if and only if (2) under every assignment γ_I of variables from I that is consistent with Σ, either o or $\neg o$ is implied by Σ (i.e., the truth value of variable o is fixed). The two characterizations (1) and (2), namely explicit definability and implicit definability, are known to be equivalent in classical logic [2].

More precisely, in the following, our goal is to compute *definability bipartitions*: we want to split the set $Var(\Sigma)$ of variables occurring in Σ into disjoint

subsets I and O such that Σ defines every variable of O in terms of the variables in I. The resulting pair $\langle I, O\rangle$ is referred to as a *definability bipartition*.

Deriving such bipartitions has been shown as a valuable preprocessing technique for model counting, a key task for a number of AI problems involving probabilities (see e.g., [6,4,7,14,5]). Indeed, whenever a variable o has been identified as an output variable from O, it can be *forgotten* from Σ without modifying the number of models. The forgetting of o from Σ [11,9] is the quantified formula $\exists o \cdot \Sigma$, i.e., Σ into which o is existentially quantified. This quantified formula is equivalent to the standard formula given by $(\Sigma|\neg o) \vee (\Sigma|o)$. Here, $\Sigma|\neg o$ (resp. $\Sigma|o$) is the formula Σ where every occurrence of variable o has been replaced by the Boolean constant $\bot$ (falsum) (resp. the Boolean constant $\top$ (verum)). Thus, when $\langle I, O\rangle$ is a definability bipartition of $Var(\Sigma)$, the number of models of Σ is precisely the number of models of Σ once projected onto I.

This observation led us to design and evaluate a preprocessing technique dedicated to model counting, which consists in deriving first a definability bipartition $\langle I, O\rangle$ of $Var(\Sigma)$ and then in eliminating in Σ the implicit existential quantifications over variables from O [6,7]. Accordingly, the corresponding preprocessing algorithm `B` + `E` consists of a pipeline of two algorithms `B` (for deriving a bipartition) and `E` for eliminating the output variables. Many experiments with various model counters considered upstream have shown that huge computational benefits can be achieved by taking advantage of `B` + `E` (see [7] for details). In order to avoid trivial bipartitions (e.g., $\langle Var(\Sigma), \emptyset\rangle$) to be considered and maximize the leverage of the approach, the focus was on deriving *subset-minimal bipartitions* in terms of input variables, i.e., definability bipartitions $\langle I, O\rangle$ such that for every $i \in I$, $\langle I \setminus \{i\}, O \cup \{i\}\rangle$ is not a definability bipartition.[3]

`B` mainly is a greedy algorithm, equipped with a definability oracle based on a SAT solver. The definability oracle used in `B` relies only on the *decision* returned by the SAT solver, i.e., whether the `CNF` formula considered as input is consistent or not. However, modern SAT solvers furnish more information that can be exploited for further treatments when used in an incremental way. Especially, when the input formula is consistent, a truth assignment forming a *model* of the formula is computed; when it is inconsistent, a subset of clauses that is conjunctively inconsistent (alias a *core*) can be extracted as well.

In the following, we show how the computation of subset-minimal bipartitions can be boosted by taking into account not only the decisions furnished by SAT solvers (as done in `B`), but also the SAT witnesses (models and cores) justifying those decisions. We present an improved bipartition algorithm, named `B+`, which takes advantage of extra-information reported by the SAT solver. We prove the correctness of the algorithm and present an empirical evaluation of it based on a large number of benchmarks (`CNF` instances) from various families. In order to figure out the benefits offered by the use of SAT witnesses of each type, we measure the number of instances for which `B+` (using models, cores, both of them, or none of them) succeeds in computing a subset-minimal bipartition in

[3] In practice, computing smallest bipartitions in terms of input variables, i.e., bipartitions $\langle I, O\rangle$ such that $|I|$ is minimal, is in general too demanding for being useful.

a given amount of time. The experiments made show that taking advantage of both models and cores is useful in practice and that B+ outperforms B (which boils down to the variant of B+ where neither models nor cores are exploited). For space reasons, proofs are not provided in the paper, but they are available online at http://www.cril.univ-artois.fr/~marquis/BDBCSW.pdf.

2 Preliminaries

Let $\mathcal{L}$ be the (classical) propositional language defined inductively from a countable set $\mathcal{P}$ of propositional variables, the usual connectives ($\neg$, $\vee$, $\wedge$, $\leftrightarrow$, etc.) and including the Boolean constants $\top$ (verum) and $\bot$ (falsum). A *literal* ℓ is a variable $\ell = x$ from $\mathcal{P}$ or a negated one $\ell = \neg x$. When ℓ is a literal, $var(\ell)$ denotes the variable from $\mathcal{P}$ upon which ℓ is built. An *interpretation* ω is a mapping from $\mathcal{P}$ to $\{0, 1\}$, represented as a set of literals. Formulae Σ are interpreted in the classical way. If $\Sigma(\omega) = 1$, then ω is called a *model* of Σ and Σ is *consistent*. In the case when Σ has no model, Σ is said to be *inconsistent*. $\models$ denotes logical entailment. For any formula Σ from $\mathcal{L}$, $Var(\Sigma)$ is the set of variables from $\mathcal{P}$ occurring in Σ. A *term* is a conjunction of literals or $\top$, and a *clause* is a disjunction of literals or $\bot$. A CNF formula Σ is a conjunction of clauses (also viewed as a set of clauses when convenient). Let X be any subset of $\mathcal{P}$. A *canonical term* γ_X over a subset X of $\mathcal{P}$ is a consistent term into which every variable from X appears (either as a positive literal or as a negative one, i.e., as a negated variable).

Let us now recall the concept of definability in propositional logic (the following definition is about implicit definability, i.e., it does not refer to any definition of y in Σ):

Definition 1 (definability). *Let $\Sigma \in \mathcal{L}$, $X \subseteq \mathcal{P}$ and $y \in \mathcal{P}$. Σ defines y in terms of X if and only if for every canonical term γ_X over X, we have $\gamma_X \wedge \Sigma \models y$ or $\gamma_X \wedge \Sigma \models \neg y$.*

Example 1. Let Σ be the CNF formula consisting of the following clauses:

$$\neg a \vee b \qquad \neg a \vee c \qquad a \vee \neg b \vee \neg c \quad \neg e \vee c \vee d \qquad \neg c \vee e$$
$$\neg d \vee e \qquad b \vee \neg c \vee d \qquad d \vee \neg a$$

a and e are defined in Σ in terms of $X = \{b, c, d\}$. For instance, the canonical term $\gamma_X = b \wedge c \wedge d$ over $\{b, c, d\}$ is such that $\Sigma \wedge \gamma_X \models a \wedge e$.

Clearly enough, Definition 1 shows that the concept of definability trivializes when Σ is inconsistent, in the sense that Σ defines each of its variables in terms of $\emptyset$ when Σ is inconsistent. Thus, in the rest of the paper, we suppose that Σ is consistent.

In the following, we state that a subset Y of variables from $\mathcal{P}$ is defined in terms of X in Σ (denoted by $X \sqsubseteq_\Sigma Y$) when every variable $y \in Y$ is defined in terms of X in Σ. It is known that deciding whether Σ defines y in terms of X is "only" coNP-complete [10]. Indeed, we can take advantage of the following result

(Padoa's theorem [13]), restricted to propositional logic and recalled in [10], to decide whether Σ defines y in terms of X. This theorem gives an entailment-based characterization of definability:

Theorem 1. *For any $\Sigma \in \mathcal{L}$ and any $X \subseteq \mathcal{P}$, let Σ'_X be the formula obtained by replacing in Σ in a uniform way every propositional symbol z from $Var(\Sigma) \setminus X$ by a new propositional symbol z'. Let $y \in \mathcal{P}$. If $y \notin X$, then Σ (implicitly) defines y in terms of X if and only if $\Sigma \wedge \Sigma'_X \wedge y \wedge \neg y'$ is inconsistent.*[4]

In [6,7], the authors took advantage of Theorem 1 in order to design a greedy bipartition algorithm, called `B`. Given a `CNF` formula Σ, `B` makes intensive use of Theorem 1 to compute a subset-minimal bipartition $\langle I, O\rangle$ of $Var(\Sigma)$, where a definability bipartition $\langle I, O\rangle$ of $Var(\Sigma)$ is a subset-minimal bipartition of $Var(\Sigma)$ if $\nexists x \in I$ such that $I \setminus \{x\} \sqsubseteq_\Sigma O \cup \{x\}$. Algorithm 1 presents a version of `B` where some technicalities (the ordering under which the variables are processed and the possibility to limit the number of learned clauses) are abstracted away. At line 1, `backbone`(Σ) computes the backbone of Σ (i.e., the set of all literals implied by Σ), and initializes O with the corresponding variables (indeed, if a literal ℓ belongs to the backbone of Σ, then $var(\ell)$ is defined in Σ in terms of $\emptyset$). Still at line 1, I is initialized to the empty set. Then, variables that are not yet identified as inputs or outputs are considered iteratively (lines 2 – 4). At line 3, `defined`? takes advantage of Theorem 1 for determining whether x is defined in Σ in terms of $Var(\Sigma) \setminus (O \cup \{x\})$, i.e., all the variables but x and those in the current set of output variables. `defined`? uses a `SAT` solver `solve` based on CDCL architecture [12] for achieving the (in)consistency test required by Padoa's method. Note that `solve` allows SAT solving "under assumptions" [3], i.e., the input `CNF` formula consists of two parts, F a `CNF` formula and $\mathcal{A}$, a conjunction of unit clauses. Thanks to the use of assumptions, clauses that are learnt at each call to `solve` are kept for the subsequent calls to `solve` within `B`.

`defined`?$(x, \Sigma, Var(\Sigma) \setminus (O \cup \{x\}))$ returns true precisely when `solve` indicates that the `CNF` formula $F \wedge \mathcal{A}$ is inconsistent, where

$$F = Padoa(\Sigma) = \Sigma \wedge \Sigma'_\emptyset \wedge \bigwedge_{z \in Var(\Sigma)} ((\neg s_z \vee \neg z \vee z') \wedge (\neg s_z \vee z \vee \neg z'))$$

and $\mathcal{A} = (\bigwedge_{s_z | z \in Var(\Sigma) \setminus (O \cup \{x\})} s_z) \wedge x \wedge \neg x'$. Variables s_z are fresh variables, used as selectors: whenever s_z is set to 1, z and z' must take the same truth value in $Padoa(\Sigma)$. Depending on the result returned by `defined`?, x is added either to O (line 3) or to I (line 4). Finally, the bipartition that has been computed is returned at line 5.

Example 2 (Example 1 cont'ed). Let us consider the `CNF` formula given in Example 1, we have:

[4] Obviously enough, in the remaining case when $y \in X$, Σ defines y in terms of X.

Algorithm 1: `B`

input : a `CNF` formula Σ
output : $\langle I, O\rangle$ a subset-minimal definability bipartition of $Var(\Sigma)$

```
1 ⟨Σ, O⟩← backbone(Σ); I←∅;
2 foreach x ∈ Var(Σ) \ (I ∪ O) do
3     if defined?(x, Σ, Var(Σ) \ (O ∪ {x})) then O←O ∪ {x};
4     else I←I ∪ {x};
5 return ⟨I, O⟩
```

$$
\begin{array}{lllll}
Padoa(\Sigma) = & \neg a \vee b & \neg a \vee c & a \vee \neg b \vee \neg c & \neg e \vee c \vee d \\
& \neg c \vee e & \neg d \vee e & d \vee \neg a & b \vee \neg c \vee d \\
& \neg a' \vee b' & \neg a' \vee c' & a' \vee \neg b' \vee \neg c' & \neg e' \vee c' \vee d' \\
& \neg c' \vee e' & \neg d' \vee e' & d' \vee \neg a' & b' \vee \neg c' \vee d' \\
& s_a \vee a \vee \neg a' & s_a \vee \neg a \vee a' & s_b \vee b \vee \neg b' & s_b \vee \neg b \vee b' \\
& s_c \vee c \vee \neg c' & s_c \vee \neg c \vee c' & s_d \vee d \vee \neg d' & s_d \vee \neg d \vee d' \\
& s_e \vee e \vee \neg e' & s_e \vee \neg e \vee e' &&
\end{array}
$$

To check whether e is defined in Σ in terms of $X = \{a, b, c\}$, it is enough to test the consistency of $Padoa(\Sigma)$ under the assumption $\mathcal{A} = \{s_a, s_b, s_c, e, \neg e'\}$. Since $\{\neg a, b, \neg c, d, e, \neg a', b', \neg c', \neg d', \neg e', s_a, s_b, s_c, \neg s_d, \neg s_e\}$ is a model of $Padoa(\Sigma)$ under the given assumption $\mathcal{A}$, we can conclude that e is not defined in Σ in terms of X. On the other hand, e is defined in Σ in terms of $X' = \{b, c, d\}$, since $Padoa(\Sigma)$ under the assumption $\mathcal{A}' = \{s_b, s_c, s_d, e, \neg e'\}$ is inconsistent.

3 Exploiting SAT Oracle Witnesses

Interestingly, modern SAT solvers "under assumptions" $\mathcal{A}$ may provide an output that does not consist only of the decision made about the consistency of its input, but may also contain a justification of the decision, alias a SAT witness. Thus, a triple $(s, \mathcal{V}, \mathcal{C})$ can be reported by the SAT solver when run on F and $\mathcal{A}$. In such a triple, s (the decision) is a Boolean value set to true when $F \wedge \mathcal{A}$ is consistent, $\mathcal{V}$ is a model of $F \wedge \mathcal{A}$ when s is true (the `SAT` witness), and $\mathcal{C} \subseteq \mathcal{A}$ is an inconsistent core of $F \wedge \mathcal{A}$, i.e., $\mathcal{C}$ is such that $F \wedge \mathcal{C}$ is inconsistent, when s is false (the `UNSAT` witness).

In order to understand how the witnesses offered by the SAT solver can be exploited in the context of a definability bipartition algorithm, it is necessary to enter into the details about the way `B` and `defined?` precisely work. At each iteration of the greedy algorithm `B`, the focus is on a variable x that has not been considered so far in previous iterations. When encountered, x is *undecided*, i.e., it has not been classified yet as input or as output. For every x, $Var(\Sigma)$ can be split into four pairwise-disjoint sets of variables: $\{x\}$, the set I_x of variables already encountered when x is processed, classified as input variables, and put in I, the set O_x of variables already encountered when x is processed, classified as output variables, and put in O, and finally the remaining set U_x of variables

that will be considered next and are still undecided when x is processed. Each of the variables $u \in U_x$ will be classified either as input (noted $u \in U_x^I$) or as output (noted $u \in U_x^O$) in a subsequent iteration, thus when all the variables will be processed, we will have either $u \in I$ or $u \in O$. When $I_x \cup U_x \sqsubseteq_\Sigma \{x\}$, since $U_x = U_x^I \cup U_x^O$, $U_x^I \subseteq I$, $U_x^O \subseteq O$, and $I \sqsubseteq_\Sigma O$, we can conclude that $I \sqsubseteq_\Sigma \{x\}$, as expected. Thus, no backtracking is necessary: the classification of each variable x achieved by the greedy algorithm when x is processed never has to be questioned. This ensures the correctness of `B` for computing a definability bipartition of $Var(\Sigma)$ (see Proposition 3 in [7]).

At each step, `defined?` is called to determine whether or not Σ defines x in terms of $I_x \cup U_x$. Finally, deciding whether or not Σ defines x in terms of $I_x \cup U_x$ amounts to calling `solve` on $F = Padoa(\Sigma) = \Sigma \wedge \Sigma'_\emptyset \wedge \bigwedge_{z \in Var(\Sigma)}((\neg s_z \vee \neg z \vee z') \wedge (\neg s_z \vee z \vee \neg z'))$ and $\mathcal{A} = \{x, \neg x'\} \cup \{s_x \mid x \in I_x \cup U_x\}$.

In the definability bipartition algorithm `B` presented in [6,7], only the decision value s of the triple returned by `solve` has been considered. We now explain how the two other components of the triple, i.e., the two types of SAT witnesses, can be exploited to improve the performance of the algorithm. Basically, at each iteration, when `defined?` is called to decide whether or not Σ defines x in terms of $I_x \cup U_x$, the goal is to take advantage of the SAT witnesses furnished by `solve` when providing a decision about x to reduce the computational efforts to be made by the definability bipartition algorithm in subsequent iterations. The two types of witnesses will be leveraged in two different ways.

When the decision returned by `solve` is false, Σ defines x in terms of $I_x \cup U_x$ and an inconsistent core $\mathcal{C}$ will be exhibited. This core makes precise a subset of assumptions of $X_\mathcal{C} \subseteq I_x \cup U_x$ such that Σ defines x in terms of $X_\mathcal{C}$. The idea is to keep track of this definability relation by forming a clause (based on selectors) $(\bigvee_{v \in X_\mathcal{C}} \neg s_v) \vee s_x)$ that reflects that Σ defines x in terms of $X_\mathcal{C}$. This clause can be freely added to $Padoa(\Sigma)$ in subsequent iterations: adding it does not change the definability relationships that may exist among variables in Σ but enables to reducing the search space visited by `solve` in the next iterations. As a matter of example, suppose that $\mathcal{C}$ indicates that Σ defines x in terms of $X_\mathcal{C} \subseteq I_x \cup U_x$. For each variable y processed after x by the definability bipartition algorithm, if $X_\mathcal{C} \subseteq I_y \cup U_y$, then the part of the search space where x and x' do not take the same truth value does not have to be explored by `solve`. The clause $(\bigvee_{v \in X_\mathcal{C}} \neg s_v) \vee s_x$ that is recorded prevents such an exploration.

When the decision returned by `solve` is true, Σ does not define x in terms of $I_x \cup U_x$. A model $\mathcal{V}$ of $F \wedge \mathcal{A}$ is exhibited and by construction, the restriction of $\mathcal{V}$ to $Var(\Sigma)$ is a model of Σ. Let y be any variable from U_x. If the canonical term $\gamma^{flip(\mathcal{V},y)}_{I_x \cup \{x\} \cup U_x}$ that coincides with $\mathcal{V}$ on $I_x \cup \{x\}$ and on every variable from U_x but y is consistent with Σ, then Σ does not define y in terms of $I_x \cup \{x\} \cup U_x$. Since $I_x \cup \{x\} \cup U_x \subseteq I_y \cup U_y$ when y is processed after x and x has been put into I, this implies that Σ does not define y in terms of $I_y \cup U_y$. Therefore, y can be put in I as well (and removed from the current set of undecided variables) as soon as x is processed. That way, there is no need to process y later on (one iteration of the definability bipartition algorithm is saved). In order to avoid an expensive

SAT call to determine whether $\gamma^{flip(\mathcal{V},y)}_{I_x \cup \{x\} \cup U_x}$ is consistent with Σ, an incomplete local search approach is used instead. The neighborhood of $\mathcal{V}$ constrained by $\gamma^{flip(\mathcal{V},y)}_{I_x \cup \{x\} \cup U_x}$ is explored in search for a partial assignment of the variables of O_x that extends $\gamma^{flip(\mathcal{V},y)}_{I_x \cup \{x\} \cup U_x}$ into a model of Σ.

In the following, before presenting our improved definability bipartition algorithm `B+`, we first point out a couple of formal results that will be useful to establish the correctness of `B+`.

3.1 Exploiting `UNSAT` Witnesses

First, let us consider the case when the decision s returned by `solve` when x is processed is false. In this case, x is defined in terms of X. Hence, $x \leftrightarrow x'$ is a logical consequence of $\Sigma \wedge \Sigma'_X$. As a consequence, assigning in $Padoa(\Sigma)$ the corresponding selector s_x to true or false does not matter when the selectors associated to the variables $v \in X$ have been set to true. Formally:

Proposition 1. *If Σ defines $x \in Var(\Sigma)$ in terms of $X \subseteq Var(\Sigma) \setminus \{x\}$, then $Padoa(\Sigma) \wedge \bigwedge_{v \in X} s_v$ does not depend on s_x, i.e., it can be rewritten into an equivalent formula into which s_x does not occur.*

As a consequence, *definability recording clauses* that keep track of previously identified definability relationships $X \sqsubseteq_\Sigma \{x\}$ can be freely added to $Padoa(\Sigma)$. Formally, $(\bigvee_{v \in X} \neg s_v) \vee s_x$ is a definability recording clause of Σ if $X \sqsubseteq_\Sigma \{x\}$ holds. The presence of such clauses does not modify the definability relationships between variables of Σ that can be found using $Padoa(\Sigma)$. This is made precise by the following proposition:

Proposition 2. *If R is a set of definability recording clauses of Σ, then for any set $X \subseteq Var(\Sigma)$ of variables and variable $x \in Var(\Sigma)$, we have that $Padoa(\Sigma) \wedge R \wedge \bigwedge_{v \in X} s_v \wedge x \wedge \neg x'$ is inconsistent if and only if $Padoa(\Sigma) \wedge \bigwedge_{v \in X} s_v \wedge x \wedge \neg x'$ is inconsistent.*

A last observation is that the inconsistent core $\mathcal{C}$ returned by `solve` can be exploited to derive a definability recording clause that is, in general, logically stronger than $(\bigvee_{v \in X} \neg s_v) \vee s_x$:

Proposition 3. *Let $(s, \mathcal{V}, \mathcal{C}) \leftarrow$ `solve`$(Padoa(\Sigma), \{x, \neg x'\} \cup \{s_v \mid v \in X\})$. If s is false, then Σ defines x in terms of $S = \{v \mid s_v \in \mathcal{C}\}$.*

The definability recording clause $(\bigvee_{v \in S} \neg s_v) \vee s_x$ found using the extracted core can then be added to $Padoa(\Sigma)$ for subsequent computations, as justified by Proposition 2.

Example 3 (Example 2 cont'ed). Suppose that the variables are considered in the following order by the definability bipartition algorithm: e, a, b, c, d. Thus, the first definability test that occurs aims to decide whether Σ defines e in

terms of $\{a, b, c, d\}$. In this case, the assumption under consideration is $\mathcal{A} = \{s_a, s_b, s_c, s_d, e, \neg e'\}$. $(false, \emptyset, \{s_b, s_c, s_d, e, \neg e'\})$ could be an outcome returned by the SAT solver. This reflects the fact that $\{b, c, d\} \sqsubseteq_\Sigma \{e\}$ (there is no need to consider a as an input variable for defining e in Σ when b, c, d are already considered as input variables). Consequently, the clause added into F is $\neg s_b \vee \neg s_c \vee \neg s_d \vee s_e$. In the next iteration, when we will check whether Σ defines a in terms of $\{b, c, d\}$, the assumption $\mathcal{A} = \{s_b, s_c, s_d, a, \neg a'\}$ will enforce s_e to be assigned to true without any significant computational effort, thanks to the added clause. Consequently, the two clauses $\neg e \vee e'$ and $e \vee \neg e'$ will be activated, so that `solve` will not explore the part of the search space where e and e' take distinct truth values.

3.2 Exploiting SAT Witnesses

Now let us consider the case when s is true, so that the variable x tested at the current iteration of the definability bipartition algorithm must be added to I. As we will see, in this case, it is possible to take advantage of the model $\mathcal{V}$ returned by `solve` and to "dig around" it in order to classify as inputs variables y that are still undecided.

Proposition 4. *Let $X \subseteq Var(\Sigma)$ and $x \in Var(\Sigma)$. If there exists a canonical term γ_X over X such that $\gamma_X \wedge x$ is consistent with Σ and $\gamma_X \wedge \neg x$ is consistent with Σ, then Σ does not define x in terms of X.*

Since the restriction to $Var(\Sigma)$ of the SAT witness returned by `solve` is a model of Σ, Proposition 4 gives a sufficient condition based on $\mathcal{V}$ that ensures that a variable $y \in U_x$ can be put into I. Indeed, if the canonical term $\gamma^{flip(\mathcal{V},y)}_{I_x \cup \{x\} \cup U_x}$ that coincides with $\mathcal{V}$ on $I_x \cup \{x\}$ and on every variable from U_x but y is consistent with Σ, then Σ does not define y in terms of $I_x \cup \{x\} \cup U_x$. Hence, as explained previously, y can be put in I and removed from the current set of undecided variables.

Because deciding whether $\gamma^{flip(\mathcal{V},y)}_{I_x \cup \{x\} \cup U_x}$ is consistent with Σ is computationally expensive in general (it requires to call a SAT solver if a complete algorithm is expected), we turn to a much cheaper, though incomplete, greedy local search to do the job. One looks for a canonical term γ_{O_x} over O_x such that the interpretation that satisfies $\gamma^{flip(\mathcal{V},y)}_{I_x \cup \{x\} \cup U_x}$ and γ_{O_x} satisfies Σ. Since every variable of $Var(\Sigma)$ is assigned either in $\gamma^{flip(\mathcal{V},y)}_{I_x \cup \{x\} \cup U_x}$ or in γ_{O_x}, the latter model checking test can be achieved in linear time.

Thus, once `solve` has shown that x must be put in the set I of input variables and has returned a model $\mathcal{V}$ justifying this decision, the variables $y \in U_x$ are considered successively. For each y, starting from $\mathcal{V}$ where the truth value of y has been flipped, we iteratively flip the truth value of variables of O_x. Such flipping operations are made while they lead to an interpretation that decreases the number of falsified clauses in Σ. If at the end of the process, the number of falsified clauses is zero, then the current interpretation is a model of Σ and

Algorithm 2: `greedyLS`

input : a `CNF` formula Σ, (I, U, O) a partition of $Var(\Sigma)$ s.t. Σ defines O in terms of $I \cup U$ and $\mathcal{V}$ a model of Σ over $Var(\Sigma)$.

output : $I_U \subseteq U$ such that Σ does not define any variable $y \in I_U$ in terms of $(I \cup U) \setminus \{y\}$.

1 $I_U \leftarrow \emptyset$;
2 **foreach** $y \in U$ **do**
3 $\quad \mathcal{V}' \leftarrow flip(\mathcal{V}, y)$;
4 $\quad$ **while** $\exists o \in O$ *and* `#false(`Σ`,` $\mathcal{V}'$`)` $>$ `#false(`Σ`,` $flip(\mathcal{V}', o)$`)` **do**
5 $\quad\quad \mathcal{V}' \leftarrow flip(\mathcal{V}', o)$;
6 $\quad$ **if** `#false(`Σ`,` $\mathcal{V}'$`)` $= 0$ **then** $I_U \leftarrow I_U \cup \{y\}$;
7 **return** I_U

since it satisfies $\gamma^{flip(\mathcal{V},y)}_{I_x \cup \{x\} \cup U_x}$ by construction, y can be definitely put in the set I of input variables that will be returned by the definability bipartition algorithm, so it is removed from the current set of undecided variables. In the remaining case when the number of falsified clauses is not null, no conclusion can be drawn. y is kept in the current set of undecided variables and it will be put in I or in O later on (at last, during the iteration when y will be processed).

Algorithm 2 implements the greedy local search process `greedyLS`. It takes as input the `CNF` formula Σ, a partition (I, U, O) of $Var(\Sigma)$ s.t. Σ defines O in terms of $I \cup U$, and a model $\mathcal{V}$ of Σ over $Var(\Sigma)$. The algorithm returns a subset of variables I_U of U such that Σ does not define any variable y of I_U in terms of $(I \cup U) \setminus \{y\}$. It starts by initializing I_U to the empty set (line 1). For each variable y in U, the algorithm tests whether y can be moved into I_U (lines 2 – 6). To do so, at line 3 the interpretation $\mathcal{V}'$ obtained by flipping the truth value of y in $\mathcal{V}$ is considered. Then, while it is possible to decrease the number of falsified clauses of Σ by flipping the truth value of some output variable $o \in O$, the truth value of o in $\mathcal{V}'$ is flipped (lines 4 – 5). If at line 6 the number of falsified clauses is zero, then y fulfills the expected requirement (the resulting interpretation $\mathcal{V}'$ is a model of Σ) and y can be added safely to the set I_U of variables (thus, its status changes from "undecided" to "input"). Finally, I_U is returned at line 7.

A last, yet useful observation, is that the `SAT` witness $\mathcal{V}$ returned by `solve` run on its input $F \wedge \mathcal{A}$ when variable x is processed can be exploited to derive not only one model of Σ but in general *two* models of Σ that can be used to classify as inputs variables that are still undecided when x is processed. As explained before, the restriction of $\mathcal{V}$ to $Var(\Sigma)$ is one of them, but there is a second interpretation that can be exploited, namely the interpretation over $Var(\Sigma)$ obtained from the restriction of $\mathcal{V}$ to $\{v' \mid v \in Var(\Sigma)\}$ by "renaming back" every variable v' into v.

Example 4 (Example 3 cont'ed). Suppose now that the variables have been considered in the following order by the definability bipartition algorithm: e, b, a, c, d. And that e has already been processed and classified as an output variable.

Algorithm 3: B+

input : a CNF formula Σ
output : $\langle I, O\rangle$ a subset-minimal definability bipartition of $Var(\Sigma)$

```
1  ⟨Σ, O, 𝕄⟩← backbone(Σ);
2  I←∅;
3  foreach 𝓜 ∈ 𝕄 do
4      I←I ∪ greedyLS(Σ, (I, Var(Σ) \ (I ∪ O), O), 𝓜);
5  Ψ←Padoa(Σ);
6  U←Var(Σ) \ (I ∪ O);
7  while U ≠ ∅ do
8      Pick a variable x in U and remove it from U;
9      (s, 𝒱, 𝒞)←solve(Ψ, {s_v | v ∈ I ∪ U} ∪ {x, ¬x'});
10     if s is false then
11         O←O ∪ {x};
12         Ψ←Ψ ∧ (s_x ∨ ⋁_{s_v ∈ 𝒞} ¬v)
13     else
14         I←I ∪ {x};
15         I←I ∪ greedyLS(Σ, (I, U, O), {ℓ ∈ 𝒱 | var(ℓ) ∈ Var(Σ)});
16         U←U \ I;
17         I←I ∪ greedyLS(Σ, (I, U, O), {ℓ | ℓ' ∈ 𝒱 and var(ℓ') ∈ Var(Σ'_∅)});
18         U←U \ I;
19 return ⟨I, O⟩
```

Then the next step is to determine whether Σ defines b in terms of $\{a, c, d\}$. In this case, the assumption under consideration is $\mathcal{A} = \{s_a, s_c, s_d, b, \neg b'\}$ and a possible outcome of `solve` is ($true$, $\mathcal{V} = \{\neg a,\ b,\ \neg c,\ d,\ e,\ \neg a',\ \neg b',\ \neg c',\ d',\ e',\ s_a,\ \neg s_b,\ s_c,\ s_d,\ s_e\}$, $\emptyset$). Hence, b can be put into the set of input variables. Before considering the next iteration of the definability bipartition algorithm, it is possible to take advantage of two models of Σ to determine whether some variables that are currently undecided can be classified as inputs. The two models are $\{\neg a, b, \neg c, d, e\}$ (obtained by restricting $\mathcal{V}$ to $Var(\Sigma)$) and $\{\neg a, \neg b, \neg c, d, e\}$ (obtained from the restriction of $\mathcal{V}$ to $\{v' \mid v \in Var(\Sigma)\}$ by "renaming back" every variable v' into v). Digging around $\{\neg a, b, \neg c, d, e\}$, we can check that the interpretation $\{\neg a, b, \neg c, \neg d, \neg e\}$ is a model of Σ. As a consequence, variable d which is still undecided can be put into the set of input variables. Digging around $\{\neg a, \neg b, \neg c, d, e\}$, we can check that the interpretation $\{\neg a, \neg b, c, d, e\}$ also is a model of Σ. Therefore, the undecided variable c can be put as well into the set of input variables. Hence, only variable a remains undecided after two iterations provided that the order e, b, a, c, d has been used.

3.3 Improving B by considering SAT Oracle Witnesses

Algorithm 3 gives the pseudo-code of B+, our implementation of the improved version of B that exploits the witnesses returned by the SAT oracle `solve`.

B+ starts by computing the backbone of Σ using the algorithm proposed in [8]. Starting from a model ω of Σ and considering each literal ℓ satisfied by ω in an iterative way, one tests the consistency of $\Sigma \wedge \neg\ell$. If $\Sigma \wedge \neg\ell$ is consistent, then a model of Σ different from ω is exhibited and we can conclude that neither ℓ nor its negation belongs to the backbone of Σ. Otherwise, ℓ belongs to the backbone of Σ. Contrary to what happens in B, where the models generated during the computation of the backbone are not used, they are exploited in B+. At line 1, Σ is simplified by its backbone, O is assigned to the set of variables belonging to the backbone and $\mathbb{M}$ is a set of models of Σ found as a by-product of the computation of the backbone. I is set to the empty set at line 2. Then, for each model $\mathcal{M}$ in $\mathbb{M}$, the greedy algorithm `greedyLS` is called in order to spot input variables (lines 3 – 4). More precisely, additional input variables are gathered into I in an iterative way by calling `greedyLS` on Σ and the given model $\mathcal{M}$. O is not modified during those iterations over $\mathbb{M}$, but as soon as input variables are detected, they are added into I (line 4), hence I usually changes (and as a consequence $Var(\Sigma) \setminus (I \cup O)$ changes as well) during the iterations.

Then, Padoa's theorem is leveraged. First, at line 5, Ψ is initialized with the `CNF` formula $Padoa(\Sigma)$. At line 6, the set of undecided variables U is set to the variables that have not been classified so far as inputs or outputs. Then, while some undecided variables remain, one variable is selected and a call to the `SAT` solver `solve` is performed (lines 7 – 18). More precisely, at line 8, a variable x belonging to U is selected and it is removed from U. `solve` is called at line 9 with the formula Ψ, and its output is stored in the triple $(s, \mathcal{V}, \mathcal{C})$. The set of assumption variables used for this call contains x, x' and the propositional variables that correspond to selectors s_v making equivalent the pairs of variables v, v' in $I \cup U$.

Depending on the value of s, two cases have to be considered (lines 10 – 18). If s is false, which means Ψ is inconsistent regarding the given assumptions, then x is added into the set of output variables (line 11) and a definability recording clause, as presented in Section 3.1, is added to Ψ (line 12). If s is true, which means Ψ is consistent under the considered assumptions, then the input set of variables I is updated with x (line 14). From the model $\mathcal{V}$ found by `solve`, two models of Σ can be extracted in general, as explained previously. Consequently, it is possible to call `greedyLS` twice in order to try and collect additional input variables (lines 15 – 18). The first call is made at line 15, where the function `greedyLS` is called with the `CNF` formula Σ, O the set of already identified output variables, U the set of undecided variables and the restriction of $\mathcal{V}$ to the variables of Σ. Then, the set of undecided variables U is updated to take into account the variables that have just been identified as inputs (line 16). The second call to `greedyLS` differs from the first one only as to the model of Σ used. For the second call, the restriction of $\mathcal{V}$ to $\{v' \mid v \in Var(\Sigma)\}$ obtained by "renaming back" its literals is considered. Again, at line 18, U is updated to take account for the update of I at line 17. Finally, the computed bipartition is returned at line 19.

The following proposition ensures that the bipartition computed by Algorithm 3, when considering Σ as an input, is a subset-minimal definability bipartition of Σ.

Proposition 5. *Algorithm 3 is correct and it terminates after a number of calls to a SAT oracle that does not exceed* $2n + 1$ *if* n *is the number of variables in* $Var(\Sigma)$.

4 Experimental Evaluation

Our objective was to evaluate empirically the benefits offered by the use of SAT witnesses of each type within B+. In our experiments, we have considered 1942 CNF instances from the *Compile!* project.[5] Those instances are gathered into 8 datasets, as follows: BN (Bayes nets) (1116), BMC (Bounded Model Checking) (18), Circuit (41), Configuration (35), Handmade (58), Planning (557), Random (104), Qif (7) (Quantitative Information Flow analysis - security) and Scheduling (6). We have also considered 1200 instances from the model counting and the projected model counting tracks of the last model counting competitions (see https://mccompetition.org (2020-2022)). The SAT solver (solve) used was Glucose [1]. Our experiments have been conducted on Intel Xeon E5-2643 (3.30 GHz) processors with 32 GiB RAM on Linux CentOS. A time-out of 100 seconds and a memory-out of 7.6 GiB have been considered for each instance.

For each instance, we measured the time needed by B+ to derive a subset-minimal definability bipartition. For each run, the opportunity of exploiting cores (core) or models (model) has been activated or not, rendering possible to compare four variants of B+ depending on the choices made for the two parameters. The version where cores and models are not used is noted (init) in the following. It merely boils down to B and is considered as a baseline approach in the comparison.

Table 1 presents the number of instances for which B+ terminated in due time and returned a subset-minimal definability bipartition. From this table, it is clear that exploiting SAT witnesses really helps in practice to reduce the time needed to compute a subset-minimal bipartition. Furthermore, whatever the benchmark category considered, the version of B+ equipped with both SAT and UNSAT witnesses solved systematically at least as many instances as B (init), and for several datasets, significantly more instances. A similar conclusion can be drawn for the versions of B+ equipped with either SAT or UNSAT witnesses. No significant degradation of performance in terms of the number of instances solved in due time can be observed (compared to B (init), only one instance from the Planning dataset is lost by B+ equipped with SAT witnesses). Thus, B+ equipped with both SAT and UNSAT solved 72 more instances than the baseline approach (2859 vs. 2787), 21 more instances than the version of B+ that only uses SAT witnesses (2859 vs. 2838), and 51 more instances than the version of B+ that only uses UNSAT witnesses (2859 vs. 2808). From this table, we also observe

[5] See http://www.cril.univ-artois.fr/kc/benchmarks.html for details.

Category	Method init	model	core	core+model
Competition	930	966	937	**973**
BN	1078	**1081**	1079	**1081**
Handmade	38	39	42	**43**
Circuit	**36**	**36**	**36**	**36**
Planning	548	547	**557**	**557**
Random	95	**104**	95	**104**
BMC	**18**	**18**	**18**	**18**
Configuration	**35**	**35**	**35**	**35**
Qif	**7**	**7**	**7**	**7**
Scheduling	2	**5**	2	**5**
Total	2787	2838	2808	**2859**

Table 1: The table shows the number of instances solved by different versions of B+ within a time limit of 100 seconds and a memory limit of 7680 MB.

that the best improvement is obtained when considering the two types of SAT witnesses at the same time, which means that the benefits offered by each type of witness are complementary.

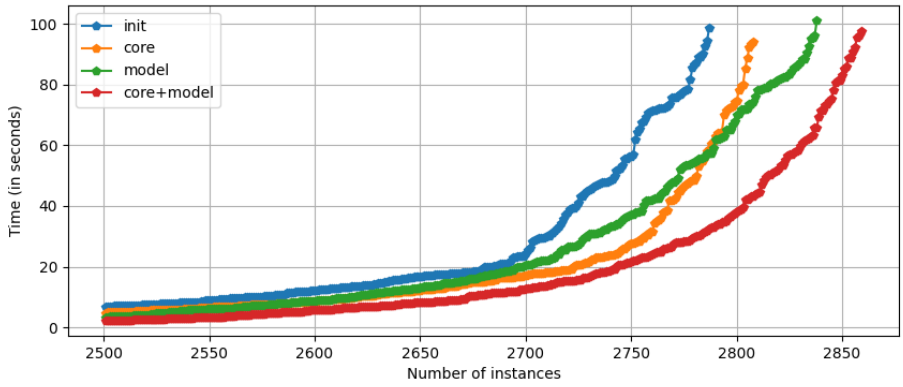
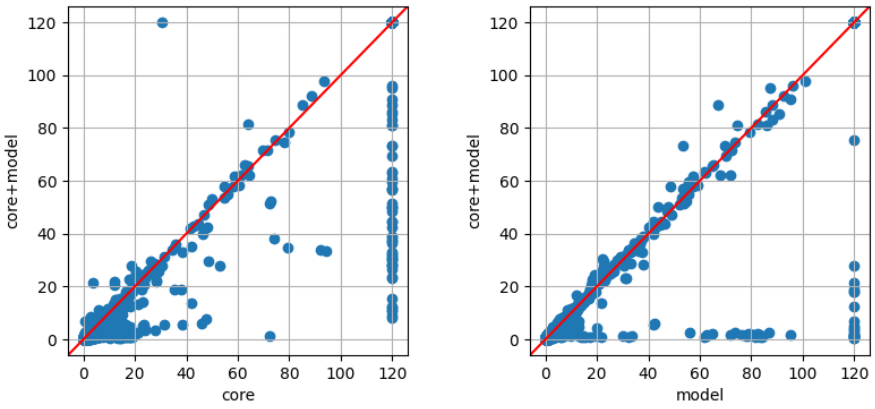


Fig. 1: Cactus plot used to compare different versions of B+. The number of instances solved is provided on the x-axis and the time needed to solve them on the y-axis.

The cactus plot in Figure 1 compares the run times of the four different versions of B+. It shows that whatever the time limit between 10 and 100 seconds, the `init` configuration solves systematically less instances than the other configurations (this behavior still occurs when the time limit is set to a value less than 10 seconds, we do not report this part of the cactus here because the figure becomes hard to be read). Figure 1 also shows that when instances become harder, the performance gap between B+ and B increases with the time bound,



(a) `core+model` vs. `core` (b) `core+model` vs. `model`

Fig. 2: Comparing the run times of different versions of `B+`.

which demonstrates that using SAT witnesses is all the more efficient when the instance under consideration appears as difficult.

Figure 2 shows a comparison between between `B+` equipped with `core+model` and the versions of `B+` where only one of `core` or `model` is used. Each dot represents an instance. The time (in seconds) needed to solve it using the version of `B+` corresponding to the x-axis (resp. y-axis) is given by the x-coordinate (resp. y-coordinate) of the dot. The experimental results, reported in Figures 2a and 2b, clearly show that the version of `B+` exploiting `SAT` and `UNSAT` witnesses is generally faster than the versions of `B+` where only one of the two types of witness is leveraged.

5 Conclusion

We have shown how to boost the computation of subset-minimal definability bipartitions through the leverage of SAT witnesses (models and cores) justifying the decisions made by the SAT solver used to solve successive instances of the definability problem. The experiments made show that taking advantage of both models and cores is useful in practice. Our new algorithm `B+` for computing subset-minimal definability bipartitions clearly outperforms the previous algorithm, `B`, developed so far for the same purpose.

Acknowledgements

The authors would like to thank the anonymous reviewers for their comments and insights. This work has benefited from the support of the PING/ACK project (ANR-18-CE40-0011) of the French National Research Agency (ANR).

References

1. Audemard, G., Lagniez, J., Simon, L.: Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In: Järvisalo, M., Gelder, A.V. (eds.) Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7962, pp. 309–317. Springer (2013). https://doi.org/10.1007/978-3-642-39071-5_23, https://doi.org/10.1007/978-3-642-39071-5_23
2. Beth, E.: On Padoa's method in the theory of definition. Indagationes mathematicae **15**, 330–339 (1953)
3. Eén, N., Sörensson, N.: An extensible sat-solver. In: Giunchiglia, E., Tacchella, A. (eds.) Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers. Lecture Notes in Computer Science, vol. 2919, pp. 502–518. Springer (2003). https://doi.org/10.1007/978-3-540-24605-3_37, https://doi.org/10.1007/978-3-540-24605-3_37
4. Ivrii, A., Malik, S., Meel, K.S., Vardi, M.Y.: On computing minimal independent support and its applications to sampling and counting. Constraints An Int. J. **21**(1), 41–58 (2016). https://doi.org/10.1007/s10601-015-9204-z, https://doi.org/10.1007/s10601-015-9204-z
5. Kiesel, R., Totis, P., Kimmig, A.: Efficient knowledge compilation beyond weighted model counting. Theory Pract. Log. Program. **22**(4), 505–522 (2022). https://doi.org/10.1017/S147106842200014X, https://doi.org/10.1017/S147106842200014X
6. Lagniez, J., Lonca, E., Marquis, P.: Improving model counting by leveraging definability. In: Kambhampati, S. (ed.) Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. pp. 751–757. IJCAI/AAAI Press (2016), http://www.ijcai.org/Abstract/16/112
7. Lagniez, J., Lonca, E., Marquis, P.: Definability for model counting. Artif. Intell. **281**, 103229 (2020). https://doi.org/10.1016/j.artint.2019.103229, https://doi.org/10.1016/j.artint.2019.103229
8. Lagniez, J., Marquis, P.: On preprocessing techniques and their impact on propositional model counting. J. Autom. Reason. **58**(4), 413–481 (2017). https://doi.org/10.1007/s10817-016-9370-8, https://doi.org/10.1007/s10817-016-9370-8
9. Lang, J., Liberatore, P., Marquis, P.: Propositional independence: Formula-variable independence and forgetting. Journal of Artificial Intelligence Research **18**, 391–443 (2003)
10. Lang, J., Marquis, P.: On propositional definability. Artif. Intell. **172**(8-9), 991–1017 (2008). https://doi.org/10.1016/j.artint.2007.12.003, https://doi.org/10.1016/j.artint.2007.12.003
11. Lin, F., Reiter, R.: Forget it! In: Proc. of AAAI Fall Symposium on Relevance. pp. 154–159 (1994)
12. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 133–182. IOS Press (2021). https://doi.org/10.3233/FAIA200987, https://doi.org/10.3233/FAIA200987

13. Padoa, A.: Essai d'une théorie algébrique des nombres entiers, précédé d'une introduction logique à une théorie déductive quelconque. In: Bibliothèque du Congrès International de Philosophie, pp. 309–365. Paris (1903)
14. Soos, M., Meel, K.S.: Arjun: An efficient independent support computation technique and its applications to counting and sampling. In: Mitra, T., Young, E.F.Y., Xiong, J. (eds.) Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2022, San Diego, California, USA, 30 October 2022 - 3 November 2022. p. 71. ACM (2022). https://doi.org/10.1145/3508352.3549406, https://doi.org/10.1145/3508352.3549406