Identifying Soft Cores in Propositional Formulae

Gilles Audemard, Jean-Marie Lagniez, Marie Miceli, Olivier Roussel

Identifying Soft Cores in Propositional Formulæ

Gilles Audemard[a], Jean-Marie Lagniez[b], Marie Miceli[c] and Olivier Roussel[d]
CRIL, Univ Artois & CNRS, Lens, France
{*audemard, lagniez, miceli, roussel*}*@cril.fr*

Keywords: SAT, Explanation, #SAT, Max#SAT.

Abstract: In view of the emergence of explainable AI, many new concepts intend to explain why systems exhibit certain behaviors while other behaviors are excluded. When dealing with constraints, explanations can take the form of subsets having few solutions, while being sufficiently small for ensuring that they are intelligible enough. To make it formal, we present a new notion, called *soft core*, characterizing both small and highly constrained parts of GCNF instances, whether satisfiable or not. *Soft cores* can be used in unsatisfiable instances as an alternative to MUSes (*Minimal Unsatisfiable Subformulæ*) or in satisfiable ones as an alternative to MESes (*Minimal Equivalent Subformulæ*). We also provide an encoding to translate *soft cores* instances into MAX#SAT instances. Finally, we propose a new method to solve MAX#SAT instances and we use it to extract *soft cores*.

1 INTRODUCTION

Nowadays, SAT is well known and used for solving many different problems such as planning, verification, cryptography or mathematical conjectures (Biere et al., 1999), (Heule et al., 2016). Solvers become increasingly efficient and new applications emerge frequently. When a problem is modeled as a SAT instance, it is often observed that its set of solutions may differ from the one that was expected. It may be due to errors in modeling, or result from the fact that the problem is too constrained. In both cases, it is worth providing the user with some form of explanation clearing up why the expected solutions are impossible.

In the particular case when the set of constraints is unsatisfiable, a MUS (*Minimally Unsatisfiable Subset*) can be extracted and reported to the user as an explanation of the discrepancy between the solutions obtained and those being expected. A MUS is a subset of constraints which is unsatisfiable but whose each proper subset of it is satisfiable (Bruni and Sassano, 2001). It can be seen as a minimal (for inclusion) part of the formula which is unsatisfiable. As instances may contain an exponential number of MUSes compared to their number of elements, one element from each MUS has to be modified or removed to make the instance satisfiable. In some practical applications like hardware and software verification (Liffiton and Sakallah, 2008), finding all MUSes is valuable to get the best diagnostic in order to repair errors. However, when it comes to explaining to human users, returning an exponential number of MUSes does not make sense. As in extreme cases its size edges toward the size of the formula, returning a single MUS may also overwhelm the user. Accordingly, the search for a smallest MUS (SMUS) has been the central topic of several papers (Ignatiev et al., 2015), (Mneimneh et al., 2005). Obviously, generating a smallest MUS of a CNF formula is more computationally difficult than extracting any MUS, as these problems are in $\mathrm{FP}^{\mathrm{NP}}$ and $\mathrm{FP}^{\Sigma_2^p}$ respectively (Ignatiev et al., 2015). However, a SMUS may still be not succinct enough or may pinpoint a part of the formula that would be irrelevant when taking into account the entire problem.

When the set of constraints at hand is satisfiable, the notion of MUS trivializes and the notion of *Minimal Equivalent Subformula* (MES) (Belov et al., 2014) can be more suited. Given a set of constraints $\mathcal{C}$, a MES is a minimal (for inclusion) subset of constraints $\mathcal{C}' \subseteq \mathcal{C}$ which is equivalent to $\mathcal{C}$. Even if computing a MES might gather interesting information, it does not help in detecting which part of the formula is hard to solve. Moreover, like MUSes, a

[a] https://orcid.org/0000-0003-2604-9657
[b] https://orcid.org/0000-0002-6557-4115
[c] https://orcid.org/0000-0002-2591-6491
[d] https://orcid.org/0000-0002-9394-3897

MES and even a smallest MES (SMES) may not be small enough to be handled by users.

To deal with these issues, we introduce in the following a generalization of the notion of MUS to both satisfiable and unsatisfiable formulæ. A *soft core* is a subset of constraints which is both small and highly constrained, meaning it has a limited number of models. Identifying a *soft core* is clearly a bi-criteria optimization problem, where both the size of the *soft core* and its number of models matter. Depending on the context, it may be more important to reduce the size of the *soft core*, even if we obtain more models. Alternatively, in other cases, reducing the number of models will be the main objective, even if this gives larger *soft cores*.

Soft cores can be proven useful in different application scenarios. They pinpoint the most constrained parts of the formula, and determinate which constraints prevent expected solutions from being possible. For example, they can be used in scheduling problems to find which part needs to be relaxed, or in the analysis of mathematical conjectures (Heule et al., 2016). Indeed, for such case, once the parameters of a conjecture (e.g. a number of colors) are instantiated, the problem can often be encoded by constraints. Except for small values of the parameters, the resulting formula is hard to be solved. Besides, if the formula is unsatisfiable, a MUS is expected to be quite large, since conjectures generally use a minimum number of hypotheses. Thus, *soft cores* may provide information and can become the start of mathematical proof if the models admitted have specific characteristics or are not numerous.

Soft cores can also be used in debugging tools, as in (Dodaro et al., 2018), whose purpose is to help the user to model problems. Generally, problems are not modeled directly into SAT, but first in higher level modeling languages and then transformed into SAT instances. When encoded as CNF formulæ, problems loose a lot of structural information, and a simple natural constraint (for example, *"we want at least participate in three sessions among the five available"*, as $s_1 + s_2 + s_3 + s_4 + s_5 \geqslant 3$) might become a large set of clauses, whose decision variables are not distinguishable from auxiliary ones, and which, if not gathered into groups, can be lost among other clauses. Then, using a Group CNF (GCNF) formula instead of a CNF formula can be preferred to keep structural information, and to treat together clauses issued from a former higher-level constraint.

We propose using *soft cores* to determine, when modeling a problem into SAT, which parts of the formula would be hard to explore, in order to either change modeling or use the information when solving the generated instance.

The paper is organized as follows. First, we formally define *soft cores*, and show that the decision problem related to *soft cores* is an $\mathrm{NP}^{\mathrm{PP}}$-hard problem by printing out a reduction from E-MajSAT (Pipatsrisawat and Darwiche, 2009). Afterwards, we demonstrate how to turn instances of the *soft cores* problem into instances of MAX#SAT, an $\mathrm{NP}^{\mathrm{PP}}$-complete problem (Fremont et al., 2017). Finally, from a practical side, we present an exact MAX#SAT solver. We show how to leverage it for computing *soft cores* and we compare it to the approximate MAX#SAT solver `Maxcount`, (Fremont et al., 2017), the only available tool for solving MAX#SAT instances up to now.

2 PRELIMINARIES

Boolean Logic. We consider standard Boolean logic. Let $\mathcal{L}_P$ be a language of formulæ over an alphabet P of Boolean variables also called *atoms*, denoted by $a, b, c, \ldots$ The symbols $\wedge$, $\vee$, $\neg$, $\Rightarrow$ and $\Leftrightarrow$ represent the standard conjunctive, disjunctive, negation, material implication and equivalence connectives, respectively. *Propositional formulæ* are built in the usual way from variables, connectives and parentheses. They are denoted by greek letters as α, β, Γ, Δ, ...We denote by $Var(\Gamma)$ the set of variables appearing in a formula Γ. For convenience we sometimes write $\Gamma(X)$ to represent that Γ is assumed to be defined on variables X. A *literal* is a variable or its negation. A *term* is a conjunction of literals $(\ell_1 \wedge \ldots \wedge \ell_s)$ while a *clause* is a disjunction of literals $(\ell_1 \vee \ldots \vee \ell_s)$. A *unit* clause is formed of one literal.

Interpretation and Model. An *interpretation* (or an *assignment*) ω to P is a mapping from P to $\{true, false\}$. ω is *complete* when all variables are assigned, otherwise it is *partial*. A variable not assigned is said to be *free*. The set of all interpretations is denoted by Ω. An interpretation ω is a *model* of a formula $\Gamma \in \mathcal{L}_P$ if and only if it makes it true in the usual truth functional way. On the contrary, an interpretation is a *counter-model* if it does not satisfy the formula. The set of models admitted by Γ is denoted $Mod(\Gamma)$, with $Mod(\Gamma) = \{\omega \in \Omega \mid \omega \text{ is a model of } \Gamma\}$. $\models$ and $\equiv$ denote respectively logical entailment and logical equivalence. Let Γ and Δ be two distinct propositional formulæ, $\Gamma \models \Delta$ if and only if $Mod(\Gamma) \subseteq Mod(\Delta)$

and $\Gamma \equiv \Delta$ if and only if $Mod(\Gamma) = Mod(\Delta)$.

CNF and DNF. A propositional formula is in *Conjunctive Normal Form* (CNF) (resp. in *Disjunctive Normal Form* (DNF)) when it is written as a conjunction of clauses (resp. a disjunction of terms). Alternatively, CNF (resp. DNF) can be represented by their set of clauses (resp. set of terms). The size of the CNF Ψ (resp. DNF), denoted by $|\Psi|$, is its number of clauses (resp. terms). The conditioning of a CNF formula Ψ by a consistent term γ is the CNF formula $\Psi|_\gamma$, obtained from Ψ by first removing each clause containing a literal $\ell \in \gamma$ and then removing all occurrences of $\neg\ell$ from the remaining clauses. When a CNF formula contains a unit clause $\{\ell\}$, Ψ and $\Psi|_\ell$ are *equisatisfiable*. The unit propagation of clause $\{\ell\}$ is the conditioning of Ψ on ℓ. A CNF formula with no unit clause is said to be *closed under propagation*. The *Boolean Constraint Propagation* (BCP), is an algorithm that, given a CNF formula, returns an equivalent CNF closed under unit propagation.

Related problems. Many problems revolve around solutions (or lack thereof) of a given CNF formula Ψ. The decision problem determining whether a model of Ψ exists is the *Boolean Satisfiability Problem* (SAT) (Biere et al., 2009). A more general problem is the counting problem #SAT (Thurley, 2006), (Lagniez and Marquis, 2017) which returns the number of models of Ψ over its own set of variables, denoted by $\| \Psi \|$. When $Var(\Psi)$ is a proper subset from the initial alphabet P, i.e. there are free variables that are omitted in Ψ, we denote its model count over P by $\| \Psi \|_P$. When $\Psi \in \mathcal{L}_P$ and $X \subseteq P$, $\exists X.\Psi$ is a quantified Boolean formula denoting (up to logical equivalence) the most general consequence of Ψ which is independent from the variables of X (Lang et al., 2003). Observe that $Var(\exists X.\Psi) \subseteq Var(\Psi) \setminus X$. The problem $\#\exists SAT$ (Aziz et al., 2015) is to determine the number of models of a quantified CNF formula $\exists X.\Psi$.

Transforming a propositional formula into a CNF form. Tseitin encoding scheme is a linear-time query-equivalent encoding scheme to translate any propositional formula Γ into a CNF formula Ψ (Tseitin, 1983). To do this, auxiliary variables are added in order to represent subformulæ. Since each additional variable is defined from the input variables, both formulæ have the same number of models (Lagniez et al., 2020). More precisely, the models of the resulting CNF encoding are extensions of the models of the input formula, but no model is created nor removed. It is also possible to use a more compact encoding that does not consider equivalence but only implication. This encoding, called Plaisted&Greenbaum encoding (Plaisted and Greenbaum, 1986), does not preserve the number of models but is equivalent modulo forgetting on the auxiliary variables.

Group CNF. Given a CNF formula, clauses can be semantically linked and thus gathered into groups $\mathcal{G} = \{G_1, G_2, \ldots, G_m\}$ (Liffiton and Sakallah, 2008). Concretely, each group is associated with an identifier which is assigned to the clauses that compose it. In some problems, it is also relevant to gather integrity constraints into a dedicated group denoted by D. A group CNF (GCNF) formula $\Phi = D \cup \mathcal{G}$ with $\mathcal{G} = \{G_1, G_2, \ldots, G_m\}$ can be interpreted as $D \wedge G_1 \wedge \ldots \wedge G_m$. Thus, all notions introduced so far on CNF naturally extend on GCNF by considering $D \wedge G_1 \wedge \ldots \wedge G_m$. Considering GCNF instead of CNF becomes really crucial when the solution of the modeled problem must satisfy some integrity constraints and when we have to select some groups, which taken together with D respect some interesting properties (Nadel, 2010), (Liffiton and Sakallah, 2008), (Belov et al., 2014).

3 PROBLEM STATEMENT AND COMPUTATIONAL COMPLEXITY

In this section, we first introduce the new notion of *soft core*, whose objective is to determine a small set of groups that constrains the most a GCNF formula. Then, we will show that computing a *soft core* whose size is predetermined is an NP^{PP}-hard problem by proposing a reduction from E-MAJSAT (Pipatsrisawat and Darwiche, 2009), which is the prototypical NP^{PP}-complete problem (Littman et al., 1998).

3.1 Definition of a *soft core*

Informally, a *soft core* is a subset of a propositional formula, that is small both in size and in the number of models admitted. More formally, it is defined as an element of the Pareto frontier of the problem whose purpose is to find a subset of the formula with two objective functions to be minimized: (a) its size and (b) its number of models. This is detailed in Definition 1.

Definition 1. Given a GCNF formula $\Phi = D \cup \mathcal{G}$, $\mathcal{G}'$ is a *soft core* of Φ if and only if:

1. $\mathcal{G}' \subseteq \mathcal{G}$;

2. $\forall \mathcal{G}'' \subseteq \mathcal{G}$, with $\mathcal{G}' \neq \mathcal{G}''$ and $|\mathcal{G}''| \leq |\mathcal{G}'|$, $\| D \cup \mathcal{G}'' \|_{Var(\Phi)} \geq \| D \cup \mathcal{G}' \|_{Var(\Phi)}$.

To simplify, we restrict the bi-criteria problem to a single criterion optimization problem by fixing the size of the subset. This size k is chosen by the user. Then, a *k-soft core* is a subset of the formula of size k with a minimum number of models. We note that, in general, a *k-soft core* is not a *soft core* because k may not appear on the Pareto frontier of the bi-criteria optimization problem.

Definition 2. Given a GCNF formula $\Phi = D \cup \mathcal{G}$ and an integer k with $k \leq |\mathcal{G}|$, $\mathcal{G}'$ is a *k-soft core* if and only if:

1. $\mathcal{G}' \subset \mathcal{G}$ and $|\mathcal{G}'| = k$;
2. $\forall \mathcal{G}'' \subseteq \mathcal{G}$ and $\mathcal{G}'' \neq \mathcal{G}'$, with $|\mathcal{G}''| = k$, $\| D \cup \mathcal{G}'' \|_{Var(\Phi)} \geqslant \| D \cup \mathcal{G}' \|_{Var(\Phi)}$.

We also define the decision problem associated to the optimization problem.

Definition 3. Given a GCNF formula $\Phi = D \cup \mathcal{G}$, $k \in \mathbb{N}$ and $m \in \mathbb{N}$, a subset $\mathcal{G}'$ of $\mathcal{G}$ is said to be a $\langle k,m \rangle$-*soft core* if $|\mathcal{G}'| \leq k$ and $\| D \cup \mathcal{G}' \|_{Var(\Phi)} \leq m$.

As already mentioned in the introduction, it is easy to demonstrate that *soft cores* generalize both SMES and SMUS notions. Indeed, given a CNF formula Ψ, the problem of minimizing k for $\langle k,0 \rangle$-*soft core* can be seen as a generalization of SMUS when treating unsatisfiable formulæ and minimizing k for $\langle k, \| \Psi \| \rangle$-*soft core* can be seen as a generalization of SMES when treating satisfiable formulæ.

In the following section, we analyze the computational complexity of the decision version of the *k-soft core* problem.

3.2 Reduction from E-MAJSAT to $\langle k,m \rangle$-*soft core*

We prove that $\langle k,m \rangle$-*soft core* is NP^{PP}-hard by considering a reduction in polynomial time from the NP^{PP}-complete problem E-MAJSAT (Littman et al., 1998). This problem is defined as follows. Let Ψ be a propositional formula in CNF defined over $X \cup Y$, where X and Y are two disjoint sets of propositional variables. Does there exist an assignment ω over X such that the majority of assignments over Y satisfies $\Psi|w$? In other words, E-MAJSAT determines if an assignment ω over X such as $\| \Psi(\omega, Y) \|_{Var(\Psi)} > \frac{1}{2} \times 2^{|Y|}$ exists.

Proposition 1. Finding a $\langle k,m \rangle$-*soft core* is NP^{PP}-hard.

Proof. Let us consider a CNF formula Ψ defined over the set of propositional variables $X \cup Y$, with $X = \{x_1, x_2, \ldots, x_n\}$. Without loss of generality we suppose that Ψ is not a tautology. Now, let us associate with Ψ the GCNF formula $\Phi = D \cup \mathcal{G}$, with:

$\mathcal{G}$. $\mathcal{G}$ simulates the choices of variables $x_i \in X$ thanks to $2 \times n$ propositional variables $C = \{c_{x_1}, c_{\neg x_1}, \ldots, c_{x_n}, c_{\neg x_n}\}$. Each pair $\{c_{x_i}, c_{\neg x_i}\}$, that we note c_ℓ, represents the choice of a literal ℓ in the chosen assignment of X. If x_i is true, then c_{x_i} and therefore c_ℓ are also true. Otherwise, if x_i is false, then $c_{\neg x_i}$ is true and c_ℓ is false.

$$\mathcal{G} = \{G_\ell \text{ s.t. } Var(\ell) \in X \text{ and } G_\ell = \{c_\ell\}\} \quad (1)$$

We denote by $\mathcal{G}'$ the set of groups whose unitary clause c_ℓ (either c_{x_i} or $c_{\neg x_i}$) are fixed to true. In order to select a well constructed interpretation ω of X, such as it does not contain both c_{x_i} and $c_{\neg x_i}$, we want to fix $k = n$, ensuring that exactly n choices are made. To do this, we have to add constraints to D.

D. Nothing ensures that x_i and its complementary $\neg x_i$ will not be chosen together. To detect this situation, we add a first constraint to D:

$$s \Leftrightarrow \bigvee_{x \in X} (c_x \Leftrightarrow c_{\neg x}) \quad (2)$$

Equation 2 introduces a new propositional variable s which is logically defined by C. Then, s indicates if ω is consistent (meaning well constructed): if at least one pair x_i and $\neg x_i$ is set to true, then s is true, otherwise s is false. We use this new variable to make impossible the selection of an inconsistent ω by adding to D the following constraint:

$$s \vee ((c_x \Rightarrow x) \wedge (c_{\neg x} \Rightarrow \neg x) \wedge \neg\Psi) \quad (3)$$

Then, given the selected interpretation ω, Equation 3 either states the number of models of $\neg\Psi$ conditioned by ω if ω is well constructed or if not, states $2^{|Var(\Psi)|}$ models. In the E-MAJSAT problem, we search for an assignment of X such that we get at least $\frac{1}{2} \times 2^{|Y|}$ models, whereas when computing a $\langle k,m \rangle$-*soft core*, we search for k groups such that we have at most m models. Thus, we use the fact that minimizing the number of models of Ψ is maximizing the number of counter models of Ψ and we consider the negation of Ψ. We obtain the following GCNF:

$$\Phi = \mathcal{G} \wedge \left(s \Leftrightarrow \bigvee_{x \in X} (c_x \Leftrightarrow c_{\neg x}) \right) \wedge (s \vee ((c_x \Rightarrow x) \wedge (c_{\neg x} \Rightarrow \neg x) \wedge \neg\Psi)) \quad (4)$$

For the sake of simplicity, we chose to not translate Equations 2 and 3 into CNF formulæ. However, and as pointed out in Section 2, this translation into a CNF formula with the same number of models can be done in polynomial time using Tseitin encoding.

Now, let us demonstrate that by fixing $m = (2^n - 1) \times 2^{|Var(\Psi)|} + \frac{1}{2} \times 2^{|Y|} - 1$, we intend to prove that there is an assignment w over X such that the majority of assignments over Y satisfies $\Phi|w$, if and only if there exists a $\mathcal{G}' \subseteq \mathcal{G}$ s.t. $|\mathcal{G}'| \leq k$ and $\| D \cup \mathcal{G}' \|_{Var(\Phi)} \leq m$.

First, let us remark that after selecting n groups, n variables of C are units and the remaining n variables of C are free. From now on, two cases have to be considered: (a) the selected groups are inconsistent in a sense that there exists a literal ℓ of X such that G_ℓ and $G_{\neg\ell}$ have been selected and (b) the selected groups are consistent in a sense that there does not exist such a literal ℓ. The remaining case, which consists in the situation where there exists a literal ℓ of X such that neither G_ℓ or $G_{\neg\ell}$ has been selected, is a consequence of case (a). Indeed, since $|\mathcal{G}| = 2 \times |X|$ and only one group is associated with each literal of X, if there exists a literal ℓ of X such that neither G_ℓ or $G_{\neg\ell}$ have been selected, then there exists ℓ' of X such that $G_{\ell'}$ and $G_{\neg\ell'}$ has been selected (Dirichlet's drawer principle).

a. Then, let us show that whatever the selected groups $\mathcal{G}'$ which fall in case (a), we have $\| D \cup \mathcal{G}' \|_{Var(\Phi)} = 2^n \times 2^{|Var(\Psi)|}$. If there exists a literal ℓ of X such that G_ℓ and $G_{\neg\ell}$ have been selected, then s is necessary true by Equation 2. By both replacing s by $\top$ in Equation 3 and simplifying Equation 2 we get $D = \top$. Consequently, the number of models of $D \cup \mathcal{G}'$ is given by the number of free variables in $D \cup \mathcal{G}'$ over $Var(\Phi)$, which are the variables of Ψ as well as half of the variables of C, thus $n + |Var(\Psi)|$ free variables. Then, whatever the n selected groups, we always have in case (a):

$$\begin{aligned} m_1 = & \| D \cup \mathcal{G}' \|_{Var(\Phi)} = 2^n \times 2^{|Var(\Psi)|} \\ & = (2^n - 1) \times 2^{|Var(\Psi)|} + 2^{|Var(\Psi)|} \end{aligned} \quad (5)$$

Since we supposed that $\Psi \not\equiv \top$, then $\| \Psi \|_{Var(\Psi)} < 2^{|Var(\Psi)|}$ and we have $m < m_1$. Consequently, we can not find out a subset $\mathcal{G}' \subseteq \mathcal{G}$ that is a $\langle k, m \rangle$-*soft core* of Φ for $k = n$ and $m = (2^n - 1) \times 2^{|Var(\Psi)|} + \frac{1}{2} \times 2^{|Y|} - 1$ if we are in case (a).

b. Let us consider the second case (b). By Dirichlet's drawer principle, $\mathcal{G}'$ will only consider one group for each literal, $\mathcal{G}' = \{G_{\ell_1}, G_{\ell_2}, \ldots, G_{\ell_n}\}$, such that $Var(\ell_i) = x_i$ and $x_i \in X$. Then, $D \cup \mathcal{G}'$ is equal to $D \cup \{c_{\ell_1}, c_{\ell_2}, \ldots, c_{\ell_n}\}$, which is equivalent to $\Gamma = D \wedge c_{\ell_1} \wedge c_{\ell_2} \wedge \ldots \wedge c_{\ell_n}$. We can show that the only situation where s is not true by Equation 2 is when $c_{\neg\ell_1}, c_{\neg\ell_2}, \ldots, c_{\neg\ell_n}$ are all set to false. Indeed, for all remaining $2^n - 1$ cases, there exists ℓ_i such that c_{ℓ_i} and $c_{\neg\ell_i}$ are true, which makes $s = \top$. For each w_c assignment of $c_{\neg\ell_1}$, $c_{\neg\ell_2}$, ..., $c_{\neg\ell_n}$, in these $2^n - 1$ cases we have $\Gamma|_{w_c} = \top$. Thus, all variables from Ψ are free and the number of models is $2^{|Var(\Psi)|}$. When we consider the interpretation w'_c that makes all $c_{\neg\ell_1}, c_{\neg\ell_2}, \ldots, c_{\neg\ell_n}$ set to false, s would also be set to false by Equation 2. Thus, we get:

$$\begin{aligned} \Gamma|_{w'_c} \equiv & \; D \wedge c_{\ell_1} \wedge c_{\ell_2} \wedge \ldots \wedge c_{\ell_n} \\ & \wedge \neg c_{\neg\ell_1} \wedge \neg c_{\neg\ell_2} \wedge \ldots \wedge \neg c_{\neg\ell_n} \\ \equiv & \; \neg s \wedge \ell_1 \wedge \ell_2 \wedge \ldots \wedge \ell_n \wedge \neg\Psi \\ & \wedge c_{\ell_1} \wedge c_{\ell_2} \wedge \ldots \wedge c_{\ell_n} \\ & \wedge \neg c_{\neg\ell_1} \wedge \neg c_{\neg\ell_2} \wedge \ldots \wedge \neg c_{\neg\ell_n} \end{aligned} \quad (6)$$

Since all the variables except those of Y are units, then $\| \Gamma|_{w'_c} \|_{Var(()\Phi)} = \| (\neg\Psi)|_{\ell_1,\ell_2,\ldots,\ell_n} \|_{Var(\Psi)}$. Consequently, in case (b) we have:

$$\begin{aligned} m_2 = & \| D \cup \mathcal{G}' \|_{Var(\Phi)} \\ & = (2^n - 1) \times 2^{|Var(\Psi)|} + \| (\neg\Psi)|_{\ell_1,\ell_2,\ldots,\ell_n} \|_{Var(\Psi)} \end{aligned} \quad (7)$$

Finally, $\mathcal{G}'$ is a $\langle k, m \rangle$-*soft core* of Φ if and only if $\mathcal{G}'$ falls in the case (b) and $m_2 \leq (2^n - 1) \times 2^{|Var(\Psi)|} + \frac{1}{2} \times 2^{|Y|} - 1$ which implies that $\| (\neg\Psi)|_{\ell_1,\ell_2,\ldots,\ell_n} \|_{Var(\Psi)} \leq \frac{1}{2} \times 2^{|Y|} - 1$. Since all variables of X are assigned, the last assertion is true only when $(\neg\Psi)|_{\ell_1,\ell_2,\ldots,\ell_n}$ has a minority of models over Y, which is the case when $\Psi|_{\ell_1,\ell_2,\ldots,\ell_n}$ has a majority of models over Y. We have proven that there is an assignment w over X such that the majority of assignments over Y satisfies $\Psi|_w$ if and only if there exists a $\langle k, m \rangle$-*soft core* of Φ. □

Proposition 1 shows that it is theoretically possible to leverage an E-MAJSAT solver in order to compute a *k-soft core*. However, in practice, the transformation of an instance of *k-soft core* into an instance of E-MAJSAT is not straightforward and more importantly (to the best of our knowledge) no E-MAJSAT solver is available. In the next section, we propose a more convenient transformation from an instance of the *soft core* problem to an instance of MAX#SAT, another NP^{PP}-complete problem. MAX#SAT is to determine an assignment of some variables that max-

imizes the number of models of a given CNF formula. The possibility of using an available tool to approximate MAX#SAT (https://github.com/dfremont/maxcount) is also an argument for using such translation.

4 EXTRACTING A *k-soft core*

A naive way to extract a *k-soft core* from a GCNF formula would be to enumerate all possible combinations of k groups, extract the selection from the formula and call a model counter to compute its number of models over the whole alphabet. Obviously, in practice this is feasible only on very small instances. Then, we propose an alternative procedure, which also guarantees completion, that encodes the problem as an instance of the problem MAX#SAT.

4.1 Translation to MAX#SAT

MAX#SAT is a recent optimization problem, defined as an extension of the #SAT problem and useful in different applications, such as planning and probabilistic inference (Fremont et al., 2017). Let Ψ be a CNF formula defined on three distinct sets of variables, X, Y, and Z, respectively called maximization, counting and existentially quantified variables. The MAX#SAT problem is to find the truth assignment ω_X over the variables X that maximizes $\| \exists Z.\Psi(\omega_X, Y, Z) \|_Y$. In other words, ω_X is the assignment that maximizes the number of assignments to Y such that $\Psi(\omega_X, Y, Z)$ is satisfiable. Thus, MAX#SAT can succinctly be summarized as $max_X \#Y \exists Z.\Psi(X,Y,Z)$. Its decision version is NP^{PP}-complete (Fremont et al., 2017).

To the best of our knowledge, the only implementation of MAX#SAT is `Maxcount` (Fremont et al., 2017), an approximate solver that takes upon entry a CNF formula and a (X,Y,Z)-partition of its variables, Z being possibly empty. `Maxcount` returns the best truth assignment found over X, alongside its approximate projected model count. As it is, it can not be used to extract a *k-soft core*. Then, we propose an encoding to transform linearly any *k-soft core* instance into a MAX#SAT one.

Addition of selectors. Let $\Phi = D \cup \{G_1, G_2, \ldots, G_m\}$ be a GCNF formula. First, to be able to extract groups from Φ, a new variable x_i called a *selector* is added to every clause from the same group $G_i \in \Phi$. For each $i \in \{1 \ldots m\}$, we obtain the augmented group $G_i^* = \bigwedge_{j=1}^{|G_i|} (\alpha_j \vee \neg x_i)$. Thus, selectors have the same behavior than identifiers in GCNF: we can interpret groups as sets of clauses gathered together via selectors and the augmented formula as a CNF. Let Y be $Var(\Phi)$ and $X = \{x_1, \ldots, x_m\}$ the set of selector variables. Then, we note Ψ the CNF formula obtained, with $\Psi(X,Y) = D \wedge \bigwedge_{i=1}^{m} G_i^*$. When a selector x_i is fixed to *false*, the set of clauses associated is satisfied and thus, said to be *deactivated*. Otherwise, the selector is removed and the set of clauses is *activated*. Conditioning $\Psi(X,Y)$ on any truth assignment ω_X over X removes all groups whose selectors have been set to *false* and extracts the remaining.

Negation of the formula. Let Θ be $(\bigwedge_{\omega_X(x_i)=1} G_i)$, i.e., the subset of groups selected by the assignment to X. Then, $\Psi(X,Y)|_{\omega_X} = D \wedge \Theta$. To make Θ a *soft core*, we want to find ω_X that minimizes $\| D \wedge \Theta \|_Y$. As minimizing the number of models of a propositional formula corresponds to maximizing its number of counter-models, we can use MAX#SAT to compute a *soft core* from a CNF formula, provided it has been negated first. By De Morgan's law, negating Ψ results in the DNF formula $\Psi^*(X,Y) = \neg D \vee \bigvee_{i=1}^{m} \neg G_i^*$. Thus, solving $\max_X \#Y(\neg\Psi^*(X,Y))$ extracts a *soft core*. However, `Maxcount` asks for a CNF input.

Transformation into a CNF. As pointed out in Section 2, a DNF formula can be linearly transformed into a CNF formula by adding auxiliary variables. To keep the correct number of models, all the auxiliary variables that are not logically defined by $Var(\Psi(X,Y))$ are put into Z, which we recall is the set of variables to be existentially quantified. To be more precise, this applies when the Plaisted&Greenbaum scheme is used. As Tseitin scheme ensures that the number of models is kept, auxiliary variables can either be in Y or Z. Whatever the encoding selected, let us call $\hat{\Psi}(X,Y,Z)$ the CNF formula that encodes $\Psi^*(X,Y)$.

Activation of k selectors. Solving $\max_X \#Y \exists Z.\hat{\Psi}(X,Y,Z)$ results into an assignment ω_X over X that selects the set of groups minimizing $\| \Psi(\omega_X, Y) \|_Y$, which without further constraint, would correspond to all groups of Φ. To guarantee that exactly k groups are selected, we add over the selectors X the cardinality constraint $\sum_{i=1}^{m} x_i = k$, translated into a CNF formula Γ (Asín et al., 2011) defined over X and Z', with $X \cap Z' = \emptyset$, Z' being the set of auxiliary variables mandatory to generate the selected encoding. The resulting formula is $\Psi^k(X,Y,Z'') = \hat{\Psi}(X,Y,Z) \wedge \Gamma(X,Z')$, with $Z'' = Z \cup Z'$. Whenever ω_X falsifies $\Gamma(X,Z')$,

$\| \exists Z''.\Psi^k(\omega_X, Y, Z'') \|_Y$ would be equal to zero. Therefore, ω_X can not be the solution to maximizing the number of models and solving $max_X \# Y \exists Z''.\Psi^k(X,Y,Z'')$ has to return a *k-soft core*.

To sum up, given a GCNF formula $\Phi = D \wedge \bigwedge_{i=1}^{m} G_i$, we compute a *k-soft core* by considering the following MAX#SAT formulation, τ being the transformation chosen to get a CNF from a DNF and χ the CNF encoding of the cardinality constraint :

$$max_X \# Y \exists Z''.(\tau(\neg(\bigwedge_{i=1}^{m}\bigwedge_{j=1}^{|G_i|}(\alpha_j \vee \neg x_i))) \wedge \chi(\sum_{i=1}^{m} x_i = k))$$

As already mentioned, there exists only one software able to handle the MAX#SAT problem, and it returns only an approximation. In the next section, we propose a new and exact approach to tackle the MAX#SAT problem.

4.2 Algorithm for Computing MAX#SAT

Algorithm 1 provides the pseudo-code of Function `max#SAT` that solves exactly MAX#SAT and which takes upon entry a CNF Ψ and a (X,Y,Z)-partition of its variables, respectively the counting, optimization and existentially quantified variables. It returns a term t, corresponding to an assignment of some variables of X, and the projected number of models of $\exists Z.\Psi(t,Y,Z)$ over Y.

By construction, t is such that all complete interpretations w of X extending t respect $\| \exists Z.\Psi(t,Y,Z) \|_Y = \| \exists Z.\Psi(w,Y,Z) \|_Y$. Based on the model counter `d4` (Lagniez and Marquis, 2017), `max#SAT` is a top-down tree-search algorithm which is, in our case, decomposed into two parts: (a) as long as the current formula contains variables from X, we branch on such variables. We keep the assignment that maximizes the number of projected models, $\| \exists Z.\Psi(t,Y,Z) \|_Y$, which is computed in the second part (b) of the tree as soon as there is no more variables from X to select and by considering in priority variables from Y.

We also take advantage of the dynamic decomposition and cache implementation of `d4`. Let Ψ be the current formula. Ψ can be partitioned into disjoint subformulæ $\{\Psi_1, \ldots, \Psi_d\}$, when for each i, $j \in \{1, \ldots, d\}$ with $i \neq j$, Ψ_i and Ψ_j do not share any variable (i.e. $Var(\Psi_i) \cap Var(\Psi_j) = \emptyset$). Then, each subformula Ψ_i is treated separately and their solutions aggregated afterwards. Furthermore, we use a cache to avoid computing once more an already encountered subformula. Each time a new value of $\langle t, \| \exists Z.\Psi(t,Y,Z) \|_Y \rangle$ is computed, it is stored in a map. If a previously computed formula is found again, then the cache would return the combination saved.

First, at line 1, one tests whether the formula Ψ is satisfiable. If not, whatever the interpretation considered on X is, the number of models corresponding would be equal to zero and `max#SAT` returns $\langle \emptyset, 0 \rangle$. `BCP` simplifies Ψ at line 2 and returns an equivalent formula Ψ' alongside the unit literals which were propagated. If Ψ' has already been cached, `max#SAT` returns `cache[`Ψ'`]` (line 3). Afterwards, we construct *ret* (line 4), the temporary result that will be returned at line 21 and which initially contains an empty term with a neutral number of models.

`connectedComponent` partitions Ψ' into a set of disjoint connected components at line 8. If there are more than one component, then at line 8, the current solution is equal to the aggregation of the solutions of all its sub-components. As they do not share any variables, $\Psi' \equiv \Psi_1 \wedge \ldots \wedge \Psi_n$. Then, $\exists Z.\Psi' \equiv \exists Z.(\Psi_1 \wedge \ldots \wedge \Psi_n) \equiv \exists Z.(\Psi_1) \wedge \ldots \wedge \exists Z.(\Psi_n)$. Therefore, the number of projected models $\| \exists Z.\Psi' \|_Y$ is equal to $\| \exists Z.\Psi_1 \|_Y \times \ldots \times \| \exists Z.\Psi_n \|_Y$, and the assignments over X are concatenated.

If Ψ' can not be partitioned into more than one component ($j = 1$), a variable from X (or if X is empty, Y) is selected (lines 11-13). If Y is also empty, then the only remaining variables are existentially quantified and the current branch can be stopped, as we already know that the current assignment has an extension on Z that satisfies Ψ'. Otherwise, regardless of whether the decision variable is from X, we compute via two recursive calls to `max#SAT` the solutions for either conditioning Ψ' by v or $\neg v$ (lines 15 and 16). If v is a maximization variable (line 17), then the current number of models is equal to the one given by the conditioning that resulted the higher number of models. If v is a counting variable (line 18), then the current number of models is equal to the sum of the two solutions, as in normal model counters. In line 19, we replace *ret* by the computed result we just stored in `cache`$[\Psi']$.

Finally, at line 20, we update the current result stored in *ret* by extending its term with the unit literals of X that have been computed by BCP, and by multiplying its number of models with the free variables of Ψ' belonging to Y.

Input: Ψ: a CNF formula,
(X,Y,Z): a partition of $Var(\Psi)$.
Output: $ret = \langle t, c \rangle$ s.t. any interpretation of X that extends t is a solution for MAX#SAT, and c the number of models obtained.

if Ψ *is unsat* **then return** $\langle \emptyset, 0 \rangle$;
(Ψ', units) $\leftarrow$ BCP(Ψ)
if `cache`$[\Psi'] \neq \emptyset$ **then return** `cache`$[\Psi']$;
$ret \leftarrow \langle \emptyset, 1 \rangle$;
$\{\Psi_1, \ldots, \Psi_j\} \leftarrow$ `connectedComponent`(Ψ')
if $j > 1$ **then**
  for $\Psi_i \in \{\Psi_1, \ldots, \Psi_j\}$ **do**
    ret $\leftarrow$ ret $\times$ `max#SAT`(Ψ_i, X, Y, Z)
  end
else if $j = 1$ **then**
  v $\leftarrow$ *undef*
  if $Var(\Psi') \cap X \neq \emptyset$ **then**
    v $\leftarrow$ `selectVar`$(Var(\Psi') \cap X)$
  else if $var(\Psi') \cap Y \neq \emptyset$ **then**
    v $\leftarrow$ `selectVar`$(Var(\Psi') \cap Y)$
  end
  if $v \neq$ *undef* **then**
    $\langle t_1, v_1 \rangle \leftarrow$ `max#SAT`$(\{\Psi' \wedge v\}, X, Y, Z)$
    $\langle t_2, v_2 \rangle \leftarrow$ `max#SAT`$(\{\Psi' \wedge \neg v\}, X, Y, Z)$
    if $v \in X$ **then**
      `cache`$[\Psi'] \leftarrow (v_1 > v_2)\ ?\ \langle t_1, v_1 \rangle : \langle t_2, v_2 \rangle$
    else
      `cache`$[\Psi'] \leftarrow \langle \emptyset, v_1 + v_2 \rangle$
    end
    $ret \leftarrow$ `cache`$[\Psi']$
  end
end
return $ret \times \langle\ \{\ell \in \text{units} \mid Var(\ell) \in X\},\ 2^{|(Var(\Psi) \setminus (var(\Psi') \cup Var(units))) \cap Y|}\ \rangle$

Algorithm 1: Function `max#SAT`.

5 EXPERIMENTAL RESULTS

We experimentally evaluated our MAX#SAT encoding of *k-soft core* instances on both the state-of-art approximate solver `Maxcount` and our complete solver based on the algorithm `max#SAT`, to compare their performance. To the best of our knowledge, as there is no similar notion of *soft cores* in the literature, we can not rely on existing benchmarks to experiment on. Thus, as performance was not the primary concern in this paper, we only considered small satisfiable instances crafted by a random 3-CNF formulæ generator, with ten different variables and a clause-to-variables ratio of 4.2. Afterwards, we translated all problems into MAX#SAT instances using the encoding presented in the previous section, with one clause per group. We considered three variants, depending on the transformation selected to get back a CNF. If the Plaisted&Greenbaum scheme was used, then all auxiliary variables were existentially quantified. Otherwise, with the Tseitin scheme, auxiliary variables were added on one side to the counting set, and on the other to the existentially quantified one, which we denote respectively by *Tseitin(Y) and Tseitin(Z)*.

For each instance, we measured the time in (seconds) required by `max#SAT` and `Maxcount` to terminate, as well as the number of models found. While `max#SAT` is exact, `Maxcount` is an approximate model counter. Thus, the estimated number of models is important to evaluate correctly the *soft cores* selected by `Maxcount`. Furthermore, `Maxcount` takes upon entry the number of copies n of the formula to use in

Table 1: Computing *soft cores* from random 3-CNF formulæ.

P&G			Tseitin (Y)			Tseitin (Z)		
max#SAT	Maxcount		max#SAT	Maxcount		max#SAT	Maxcount	
time(s)	#mod	time(s)	time(s)	#mod	time(s)	time(s)	#mod	time(s)
102	560	**92**	**135**	timeout		107	536	**77**
91	560	**84**	**147**	timeout		101	560	**86**
91	576	99	**136**	timeout		111	560	**85**
99	547	**81**	**151**	timeout		95	544	**80**
85	560	89	**130**	timeout		105	564	**89**
105	536	**99**	**148**	timeout		100	544	**84**
95	576	99	**137**	timeout		125	576	**105**
107	576	**102**	**153**	timeout		102	576	**92**
94	560	**92**	**137**	timeout		115	560	**85**
107	576	**102**	**142**	timeout		97	532	**90**

the self-decomposition. We have tested each instance with a n set to 0, 3 and 5. When n is equal to zero, the assignment to X is given at random with no constraints, and during our experiments, it always returned a solution that violated the cardinality constraint. However, setting n to a much higher integer ($n = 5$) did not result into a better estimated count, and increased significantly the runtime. To be fair, we only report `Maxcount` with $n = 3$, the best parameter from all tested.

All the experiments have been conducted on a cluster of Intel XEON X5550 (2,66 Ghz) bi-core processors with 32 GiB RAM. Each solver was run with a time-out of three hours and a memory limit of 32 GiB of per input instance.

Table 1 reports the results for each considered variant. As each instance contains 42 clauses and thus, 42 selectors, we set k equal to 5 in order to restrict the number of possible combinations, $\binom{42}{5}$ being already equal to 859,668. Yet, the naive method proposed in the introduction of Section 4 did not terminate within the specified time. As the structure stays the same, i.e., instances are all composed of 42 ternary clauses, all formulæ after translation were up to 462 variables, with respectively 1625 and 1667 clauses for the Plaisted&Greenbaum or Tseitin transformations. As `max#SAT` is complete, it returns the assignment on X that maximizes the number of models of the negated formula. We did not report the number of models computed, as it is always the same, which is here equal to 640 models.

The experiment shows that `Maxcount` only terminates when auxiliary variables are put into Z, and is slightly faster if the encoding variables are equivalent to the subformulæ they represent, as in the Tseitin scheme. The results given with `max#SAT` are a mirror image of `Maxcount`: besides giving a solution when counting the auxiliary variables, selecting the Plaisted&Greenbaum scheme is faster than using the Tseitin transformation. In most cases, `Maxcount` is faster than `max#SAT` but it also always returns an estimated count that is lower than the optimum solution. Obviously, this experiment is only an outline of what could be solved by `Maxcount` and `max#SAT`, the primary objective being to show that on small instances, `max#SAT` is rather competitive with `Maxcount`. However, on larger instances, `Maxcount` may scale up better than MAX#SAT, as it is an approximate solver.

6 CONCLUSION AND PERSPECTIVES

In order to explain results from CNF formulæ, we introduced a new notion called *soft core*, which is a sufficiently small and highly constrained part of a formula. *Soft cores* can be used to identify which constraints should be relaxed in order to obtain other solutions, to select the most relevant constraints in the formula or to help when modeling. Identifying *soft cores* is a bi-criteria optimization problem where both the size and the number of models of the subformula have to be minimized. In this article, we focused on the restricted problem *k-soft core* where the size of the *soft core* is given by the user, thus becoming a single objective function problem. We showed the NP^{PP}-hardness of the decision version of the *k-soft core* problem by considering a reduction from E-MAJSAT, and proposed an encoding to transform *k-soft core* instances into MAX#SAT ones, as well as a first exact MAX#SAT solver. At last, an experimental evaluation on randomly small generated CNF formulæ has been realized.

As expected, even the restricted version *k-soft core* is a difficult problem. At the present time, as long as

the cardinality constraint is considered globally, only small instances can be envisioned. Indeed, the major part of the tree search corresponds to assignments that falsify the cardinality constraint. Thus, pruning them would improve the performance. Furthermore, the cardinality constraint, in addition to increase the size of the formula, may prevent it to be partitioned into disjoint components, which would also fasten the runtime. Thus, to be more effective when searching for *k-soft cores*, a first improvement is to handle directly the cardinality constraint. A second perspective is to stop considering the problem as a MAX#SAT instance, but to use a dedicated solver whose purpose would be to minimize directly the number of models, and use the input formula without transforming it. Finally, we could also compute approximately *soft cores* by considering local search: we first pick k groups from the formula, and we switch elements one by one until no switch decreases the number of models admitted.

REFERENCES

Asín, R., Nieuwenhuis, R., Oliveras, A., and Rodríguez-Carbonell, E. (2011). Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221.

Aziz, R. A., Chu, G., Muise, C. J., and Stuckey, P. J. (2015). #∃SAT: Projected model counting. In *Proceedings of SAT'05*, volume 9340 of *Lecture Notes in Computer Science*, pages 121–137.

Belov, A., Janota, M., Lynce, I., and Marques-Silva, J. (2014). Algorithms for computing minimal equivalent subformulas. *Artif. Intell.*, 216:309–326.

Biere, A., Cimatti, A., Clarke, E. M., and Zhu, Y. (1999). Symbolic model checking without bdds. In *Proceedings of TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207.

Biere, A., Heule, M., van Maaren, H., and Walsh, T., editors (2009). *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Bruni, R. and Sassano, A. (2001). Restoring satisfiability or maintaining unsatisfiability by finding small unsatisfiable subformulae. *Electron. Notes Discret. Math.*, 9:162–173.

Dodaro, C., Gasteiger, P., Reale, K., Ricca, F., and Schekotihin, K. (2018). Debugging non-ground ASP programs: Technique and graphical tools. *CoRR*.

Fremont, D. J., Rabe, M. N., and Seshia, S. A. (2017). Maximum model counting. In *Proceedings of AAAI'17*, pages 3885–3892.

Heule, M. J. H., Kullmann, O., and Marek, V. W. (2016). Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *Proceedings of SAT'16*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245.

Ignatiev, A., Previti, A., Liffiton, M. H., and Marques-Silva, J. (2015). Smallest MUS extraction with minimal hitting set dualization. In *Proceedings of CP'15*, volume 9255 of *Lecture Notes in Computer Science*, pages 173–182.

Lagniez, J., Lonca, E., and Marquis, P. (2020). Definability for model counting. *Artif. Intell.*, page 103229.

Lagniez, J. and Marquis, P. (2017). An improved Decision-DNNF compiler. In *Proceedings of IJCAI'17*, pages 667–673.

Lang, J., Lin, F., and Marquis, P. (2003). Causal theories of action: A computational core. In *Proceedings of IJCAI'03*, pages 1073–1078.

Liffiton, M. H. and Sakallah, K. A. (2008). Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reason.*, 40(1):1–33.

Littman, M. L., Goldsmith, J., and Mundhenk, M. (1998). The computational complexity of probabilistic planning. *J. Artif. Intell. Res.*, 9:1–36.

Mneimneh, M. N., Lynce, I., Andraus, Z. S., Silva, J. P. M., and Sakallah, K. A. (2005). A branch-and-bound algorithm for extracting smallest minimal unsatisfiable formulas. In *Proceedings of SAT'05*, volume 3569 of *Lecture Notes in Computer Science*, pages 467–474.

Nadel, A. (2010). Boosting minimal unsatisfiable core extraction. In *Proceedings of FMCAD'10*, pages 221–229.

Pipatsrisawat, K. and Darwiche, A. (2009). A new d-dnnf-based bound computation algorithm for functional E-MAJSAT. In *Proceedings of IJCAI'09*, pages 590–595.

Plaisted, D. A. and Greenbaum, S. (1986). A structure-preserving clause form translation. *J. Symb. Comput.*, 2(3):293–304.

Thurley, M. (2006). Sharpsat - counting models with advanced component caching and implicit BCP. In *Proceedings of SAT'06*, volume 4121 of *Lecture Notes in Computer Science*, pages 424–429.

Tseitin, G. S. (1983). On the complexity of derivation in propositional calculus. In *Automation of Reasoning*, pages 466–483.