Artificial Intelligence 216 (2014) 309–326

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Algorithms for computing minimal equivalent subformulas ☆

Anton Belov [a,*], Mikoláš Janota [b], Inês Lynce [b], Joao Marques-Silva [a,b]

[a] CASL, University College Dublin, Ireland
[b] IST/INESC-ID, Technical University of Lisbon, Portugal

ARTICLE INFO

Article history:
Received 18 October 2013
Received in revised form 3 July 2014
Accepted 26 July 2014
Available online 30 July 2014

Keywords:
Boolean satisfiability
Redundancy
Irredundant subformula
Minimal unsatisfiability

ABSTRACT

Knowledge representation and reasoning using propositional logic is an important component of AI systems. A propositional formula in Conjunctive Normal Form (CNF) may contain redundant clauses — clauses whose removal from the formula does not affect the set of its models. Identification of redundant clauses is important because redundancy often leads to unnecessary computation, wasted storage, and may obscure the structure of the problem. A formula obtained by the removal of all redundant clauses from a given CNF formula $\mathcal{F}$ is called a Minimal Equivalent Subformula (MES) of $\mathcal{F}$. This paper proposes a number of efficient algorithms and optimization techniques for the computation of MESes. Previous work on MES computation proposes a simple algorithm based on iterative application of the definition of a redundant clause, similar to the well-known deletion-based approach for the computation of Minimal Unsatisfiable Subformulas (MUSes). This paper observes that, in fact, most of the existing algorithms for the computation of MUSes can be adapted to the computation of MESes. However, some of the optimization techniques that are crucial for the performance of the state-of-the-art MUS extractors cannot be applied in the context of MES computation, and thus the resulting algorithms are often not efficient in practice. To address the problem of efficient computation of MESes, the paper develops a new class of algorithms that are based on the iterative analysis of subsets of clauses, and a lightweight pruning technique based on the computation of backbones. The experimental results, obtained on representative problem instances, confirm the effectiveness of the proposed methods. The experimental results also reveal that many CNF instances obtained from the practical applications of SAT exhibit a large degree of redundancy.

1. Introduction

Knowledge representation and reasoning based on propositional logic is an important component of AI systems [1]. Conjunctive Normal Form (CNF) plays a special role in both representation and reasoning. Indeed, CNF has become a standard for the input of modern Boolean satisfiability (SAT) solvers and other reasoning engines due to its generality and its amenability to automated processing. This article is concerned with redundancy in propositional formulas in CNF. Specifically, this article addresses the problem of computing an *irredundant* subformula of a given CNF formula — that is, a subformula $\mathcal{E}$ of $\mathcal{F}$ that has exactly the same set of models as $\mathcal{F}$ and is minimal in the sense that removal of any clause from $\mathcal{E}$ violates

☆ This work is partially supported by SFI PI grant BEACON (09/IN.1/I2618), FCT grants ATTEST (CMU-PT/ELE/0009/2009) and POLARIS (PTDC/EIA-CCO/123051/2010), and multiannual PIDDAC program funds (PEst-OE/EEI/LA0021/2013).

* Corresponding author.
E-mail addresses: anton.belov@ucd.ie (A. Belov), mikolas.janota@gmail.com (M. Janota), ines@sat.inesc-id.pt (I. Lynce), jpms@ucd.ie (J. Marques-Silva).

http://dx.doi.org/10.1016/j.artint.2014.07.011
0004-3702/

this property. Such subformula $\mathcal{E}$ will be referred to as a *Minimal Equivalent Subformula (MES)* of $\mathcal{F}$. Intuitively, an MES of $\mathcal{F}$ constitutes the result of removal of all *redundant clauses* from $\mathcal{F}$ — these are the clauses whose removal does not change the set of models of $\mathcal{F}$.

The complexity characterization of the MES computation problem, as well as a number of additional problems related to identification of redundant clauses, has been investigated extensively by Liberatore in [2]. Previous work on algorithms for removing redundant clauses from CNF formulas proposes a direct approach [3] (and, also, [2]), which iteratively checks the definition of redundant clause and removes the clauses that are found to be redundant. While the direct approach is similar to the well-known deletion-based approach for the computation of Minimal Unsatisfiable Subformulas (MUSes), most of the techniques developed for the extraction of MUSes (e.g. see [4–6]) have not been extended to the computation of MESes.

One specific application of the MES computation problem, and the extended version called the *group-MES* computation problem (also discussed in this article), is the removal of some or all redundant clauses from CNF formulas. Although in some settings redundancy in a formula might be desirable, in many others it is not. One example of desirable redundancy is that introduced by CNF encodings of various CSP constraints, where redundant clauses help to maintain higher levels of consistency (see, for example, [7] for an overview of work on this topic). Another important example of desirable redundancy is the modern Conflict-Driven Clause Learning (CDCL) SAT solvers. In this setting, all clauses learned during SAT solving are redundant [8,9], and are often essential for solving practical instances of SAT. However, redundancy can also be undesirable. For example, in knowledge bases formula redundancy leads to the use of unnecessary storage and computational resources [2]. Another example is the undesirable redundant clauses in the CNF representation of belief states in a conformant planner [10,11]. In the context of probabilistic reasoning systems, concise representation of conditional independence information can be computed by removing redundant clauses from certain propositional encodings [12].

More generally, given the wide range of applications of SAT, one can pose the following question: does a given problem domain encoder introduce an undesirable, or an unintended, redundancy, and if so, which clauses are redundant, and how significant is the percentage of these redundant clauses? The algorithms and techniques for computation of MESes and group-MESes presented in this article are intended to assist in answering this question.

Removal of redundancies can also find application in solving problems from different complexity classes, for example in Quantified Boolean Formulas (QBF) [13]. It has been shown that simplifications of the propositional part of a QBF often lead to a performance improvement in the QBF solver [14,15]. Besides propositional logic formulas, the problem of the identification of redundant constraints is relevant in other domains. Concrete examples include Constraint Satisfaction Problems (CSP) [16–18], Satisfiability Modulo Theories (SMT) [19], and Ontologies [20].

The article makes the following contributions. First, it shows that many of the existing MUS extraction algorithms can be extended to the computation of MESes. Since efficient MUS extraction uses a number of key techniques for reducing the total number of SAT solver calls — namely, clause set refinement [21–23] and model rotation [23,24] — this article analyzes these techniques in the context of MES extraction. The second contribution of the article is, then, to show that model rotation can be integrated, and, in fact, improved, in MES extraction. However, clause set refinement *cannot* be used in MES algorithms derived from the existing MUS algorithms. Third, the article proposes a reduction from MES computation problem to group-MUS computation problem [25,22]; this reduction enables the use of *both* model rotation and clause set refinement for MES extraction. Fourth, given that the approach of reduction to group-MUS can result in hard instances of SAT, the article proposes an incremental reduction of MES to group-MUS extraction, that involves the separate analysis of subsets of clauses. Though the proposed algorithms already significantly outperform the direct approach, the article also proposes a very effective and light-weight incomplete technique for pruning redundant clauses based on the computation of backbones of propositional formulas. Finally, the article extends the techniques to a group-MES computation problem, and develops solutions for checking that computed MESes are correct. These solutions find application in settings where independent certification is required.

Experimental results, obtained on representative satisfiable instances from past SAT competitions, show that the new algorithms for MES computation achieve significant performance gains over the basic algorithms, and allow targeting redundancy removal for reasonably sized formulas. In addition, the experimental results show that *many* CNF formulas, from a wide range of application domains, contain a significant percentage of redundant clauses, in some cases exceeding 90% of the original clauses.

This article is an extended version of the publication that appeared in the proceedings of 18th International Conference on Principles and Practice of Constraint Programming (CP 2012) [26]. With respect to the original publication, this article introduces the novel backbone-based pruning technique (Section 4), and presents the experimental evaluation results that confirm its effectiveness. Furthermore, the article formalizes the group-MES computation problem, and presents the generalizations of the new techniques for the computation of MESes to the group-MES setting (Section 6).

2. Preliminaries

Standard definitions for propositional logic are assumed. Propositional formulas are defined over a set of propositional variables $X = \{x_1, \ldots, x_n\}$. A CNF formula $\mathcal{F}$ is a conjunction of disjunctions of *literals* (*clauses*), where a literal is a variable or its complement. Unless otherwise stated, a CNF formula will be referred to as a formula. Formulas are represented by letters with calligraphic fonts, e.g. $\mathcal{F}$, $\mathcal{E}$, $\mathcal{S}$, $\mathcal{W}$, etc. When necessary, subscripts are used. Clauses are represented by c or c_i, $i = 1, \ldots, m$. A CNF formula can also be viewed as a multiset of non-tautologous clauses where a clause is a (multi)set of

literals. The two representations are used interchangeably, and are clear from the context. A *truth assignment* μ is a mapping from X to $\{0,1\}$, $\mu : X \to \{0,1\}$. For a truth assignment μ we write $\mathcal{F}[\mu]$ to denote the substitution of each variable x in the domain of μ with the constant $\mu(x)$; additionally, trivial simplifications are applied ($\mathcal{F} \vee 0 = \mathcal{F}$, $\mathcal{F} \vee 1 = 1$, etc.). Note that whenever μ assigns a value to all variables occurring in a formula $\mathcal{F}$, the formula $\mathcal{F}[\mu]$ is equal to either 1 or 0. A truth assignment μ *satisfies* (resp. *falsifies*) a clause c if $c[\mu] = 1$ (resp. $c[\mu] = 0$). A truth assignment μ is a *model* of F if it satisfies all clauses in $\mathcal{F}$, i.e. $\mathcal{F}[\mu] = 1$. Two formulas $\mathcal{F}_1$ and $\mathcal{F}_2$ are *equivalent*, $\mathcal{F}_1 \equiv \mathcal{F}_2$, if they have the same set of models. Note that models are always defined on a fixed set of variables X, which ensures that equivalence is well-defined even for cases when some variables are present only in one of the formulas. The negation of a clause c, denoted by $\neg c$ represents a conjunction of unit clauses, one for each literal in c. It will also be necessary to negate CNF formulas, e.g. $\neg\mathcal{F}$. The following CNF encoding for $\neg\mathcal{F}$ is used [27]. An auxiliary variable u_i is associated with each $c_i \in \mathcal{F}$, defining a set of variables U. For each $l_{i,j} \in c_i$, create binary clauses $(\neg l_{i,j} \vee \neg u_i)$. Finally, create a clause $(\bigvee_{u_i \in U} u_i)$. This CNF encoding is represented as CNF$(\cdot)$. For example, CNF$(\neg c)$ and CNF$(\neg\mathcal{F})$ denote, respectively, the CNF encoding of $\neg c$ and $\neg\mathcal{F}$ described above; both CNF$(\neg c)$ and CNF$(\neg\mathcal{F})$ can be used in set context to denote sets of clauses. Calls to a SAT solver are represented with SAT$(\mathcal{F})$ and unless specified otherwise, the return value is true if and only if the given formula $\mathcal{F}$ is satisfiable.

2.1. MUSes and MESes

The following definition of Minimal Unsatisfiable Subformulas (MUSes) is used [28].

Definition 1 *(MUS).* $\mathcal{M} \subseteq \mathcal{F}$ is a *Minimal Unsatisfiable Subformula* (MUS) of $\mathcal{F}$ iff $\mathcal{M}$ is unsatisfiable and $\forall_{\mathcal{S} \subsetneq \mathcal{M}}\, \mathcal{S}$ is satisfiable.

MUS extraction algorithms can be broadly characterized as *deletion-based* [29,30] or *insertion-based*. Moreover, insertion-based algorithms can be characterized as *linear search* [31], *dichotomic search* [32], or QuickXplain [33]. Recent work proposed a theoretically-optimal, and also efficient in practice, progression-based algorithm [34]. The most practically efficient MUS extraction algorithms — the *hybrid* [23] and the progression-based — employ a number of pruning techniques that allow to significantly reduce the number of SAT solver calls. Such techniques include *clause set trimming* (during preprocessing), *clause set refinement* and *model rotation* [21–24]. Clause set refinement [21,23] exploits the SAT solver *false* (or *unsatisfiable*) outcomes to reduce the set of clauses that need to be analyzed. This consists of removing clauses that are *not* included in the MUS being constructed, e.g. by restricting the target set of clauses to the unsatisfiable subset computed by the SAT solver. In contrast, model rotation exploits the SAT solver *true* (or *satisfiable*) outcomes to also reduce the set of clauses that need to be analyzed. In this case, models are used to identify clauses that *must* be included in the MUS being constructed. Recent experimental data [23] indicates that these two techniques are *essential* for MUS extraction on large application problem instances.

Motivated by several applications, MUSes and related concepts have been extended to CNF formulas where clauses are partitioned into disjoint sets called *groups* [25,22].

Definition 2 *(Group-oriented MUS).* Given an explicitly partitioned unsatisfiable CNF formula $\mathcal{F} = \mathcal{D} \cup \mathcal{G}_1 \cup \ldots \cup \mathcal{G}_n$, a *group oriented MUS* (or, *group-MUS*) of $\mathcal{F}$ is a subset $\mathcal{F}' = \mathcal{D} \cup \mathcal{G}_{i_1} \cup \ldots \cup \mathcal{G}_{i_k}$ of $\mathcal{F}$ (with $1 \le i_j \le n$ for $1 \le j \le k \le n$) such that $\mathcal{F}'$ is unsatisfiable and, for every $1 \le j \le k$, $\mathcal{F}' \setminus \mathcal{G}_{i_j}$ is satisfiable.

The group $\mathcal{D}$ in the above definition is called a *don't care* group, and the explicitly partitioned CNF formulas as above are referred to as *group-CNF* formulas.

MUSes are a special case (for unsatisfiable formulas) of irredundant subformulas [2]. The following definitions will be used throughout.

Definition 3 *(Redundant/irredundant clause/formula).* A clause $c \in \mathcal{F}$ is said to be *redundant* in $\mathcal{F}$ if $\mathcal{F} \setminus \{c\} \models c$ or, equivalently, $\mathcal{F} \setminus \{c\} \cup \text{CNF}(\neg c) \models \bot$. Otherwise, c is said to be *irredundant* in $\mathcal{F}$. A formula $\mathcal{F}$ is redundant if it has at least one redundant clause; otherwise it is irredundant.

Irredundant subformulas of (redundant) formulas are referred to as *irredundant equivalent subsets* [2] and as *irredundant cores* [35]. In this paper, irredundant subformulas are referred to as *Minimal Equivalent Subformulas* (MESes), by analogy with MUSes.

Definition 4 *(MES).* $\mathcal{E} \subseteq \mathcal{F}$ is a *Minimal Equivalent Subformula* (MES) of $\mathcal{F}$ iff $\mathcal{E} \equiv \mathcal{F}$ and $\forall_{\mathcal{Q} \subsetneq \mathcal{E}}\, \mathcal{Q} \not\equiv \mathcal{F}$.

Clearly, an MES of any formula $\mathcal{F}$ is irredundant. Moreover, deciding whether a CNF formula is an MES is D^P-complete [2]. In the case of group-CNF formulas, the concept of *group-oriented MES (group-MES)* can be defined analogously to Definition 2 — Section 6 contains the necessary details.

2.2. Existing techniques

This subsection overviews several techniques referred to throughout the article, namely, unsatisfiable cores, model rotation, and clause set refinement. These are described in detail in the references [36,37,23,24,38,21,22].

Some of the presented algorithms avail of *unsatisfiable cores* [36,37]. For an unsatisfiable formula $\mathcal{F}$ an unsatisfiable core is a formula $\mathcal{C}$ that is unsatisfiable and such that $\mathcal{C} \subseteq \mathcal{F}$. To obtain an unsatisfiable core, one can use either a *proof-tracing* SAT solver, such as `picosat` [39], or an *assumption-based* SAT solver, cf. [40,41]. In theory, an unsatisfiable core does not guarantee any minimality — it can for instance be equal to the whole formula. In practice, however, unsatisfiable cores are often significantly smaller than the input formula.

In MUS computation a clause c is necessary for unsatisfiability of $\mathcal{F}$ if removing c from $\mathcal{F}$ yields satisfiability. When this is detected by a SAT solver, the solver provides us with a model μ of $\mathcal{F} \setminus \{c\}$. The technique of *model rotation* tries to obtain a model μ' of $\mathcal{F} \setminus \{c'\}$ for some different clause c' from μ by flipping the value of variables that appear in c. Once such μ' and c' are found, the clause c' is also marked as necessary for unsatisfiability [23,24,38]. This technique in practice allows to significantly reduce the number of calls to SAT solver and results in multiple orders of magnitude reduction in run-times of hybrid MUS extraction algorithms on industrial problem instances (cf. [38]).

In MUS computation a SAT solver is commonly invoked. Often, the unsatisfiable core of an unsatisfiable call is useful. For instance, given a formula $\mathcal{F}$, one can inspect each clause c at a time and test whether removing it from $\mathcal{F}$ is satisfiable or not. If $\mathcal{F} \setminus \{c\}$ satisfiable, the clause is put back into $\mathcal{F}$. If it is unsatisfiable, the clause c is discarded. Moreover, if the underlying SAT solver provides an unsatisfiable core $\mathcal{C} \subseteq (\mathcal{F} \setminus \{c\})$, the algorithm continues with $\mathcal{C}$ as it is an unsatisfiable subset of $\mathcal{F}$ (which is guaranteed to not contain c). This technique is called the *clause set refinement* [21–23]. Note that clause set refinement and model rotation complement one another.

2.3. Related work

MUSes find a wide range of practical applications, and have been extensively studied (see [4–6] for recent overviews, and [33,32,42,43] for connections with CSP). The problem of computing minimal (or irredundant) representations of CNF formulas (and related problems) has been the subject of extensive research [44,45,3,2,46,47,35], with applications in the compression of Horn knowledge bases [45], and minimal representation of hypergraphs [44]. Complexity characterizations of redundancy problems in logic can be found in [2]. An algorithm for computing an MES based on the direct application of the definition of clause redundancy is studied in [3]. More recently, properties of MESes are studied in [35].

Approximate solutions for redundancy removal based on unit propagation are proposed in [48,49]. A number of applications (of restricted forms) of redundancy removal have been proposed in recent years [50,10,11,48,2,12], including contingent planning [10,11], conditional independence [12], and CNF formula preprocessing [50] The importance of redundant clauses in CDCL SAT solvers is addressed in [8,9].

The removal of redundancy in propositional formulas finds important applications in combinational and sequential circuit design [51,52], including both computing minimum size DNF representations [53–56], or minimal DNF representations [51, 52]. Nevertheless, the decision problem of exact minimization of propositional formulas is located in the second level of the polynomial hierarchy [57], even for the case of DNF representations [58]. As a result, the computation of a minimal DNF representation finds application in the design of most modern computing systems. Recent work in propositional formula minimization has addressed the removal of variables and connections between variables [59,60].

Redundancy problems have been studied in many other settings, e.g. [16–20], including constraint networks [16–18], satisfiability modulo theories [19], and ontologies [20].

Backbones find a number of applications in other analyses, e.g. product configuration [61–63] or problem encoding [64].

3. MES extraction algorithms

This section develops several new approaches for computing one MES of a CNF formula $\mathcal{F}$. The first solution consists of adapting *any* MUS extraction algorithm based on identification of so-called transition clauses, for MES extraction. Afterwards, key techniques used in MUS extraction are studied. Model rotation [23,24] is applied to MES extraction, and it is argued that clause set refinement [21–23] cannot be directly applied to MES extraction algorithms resulting from adapting existing MUS extraction algorithms. Next, a reduction from MES to group-MUS formulation [25,22] is developed, which enables the use of both model rotation and clause set refinement. Although the reduction of MES to group-MUS extraction enables the integration of key techniques, it is also the case that the resulting instances of SAT are hard to solve. This section concludes by developing an incremental reduction from MES to group-MUS extraction, which produces much easier instances of SAT. Fig. 1 summarizes the approaches to MES extraction described in the remainder of this section.

3.1. From MUS extraction to MES extraction

A key definition in MUS extraction algorithms is that of *transition clause* [4]. A transition clause c is such that, if added to a satisfiable subformula $\mathcal{R}$, the resulting subformula is unsatisfiable. This definition can be generalized for MES extraction as follows. Let $\mathcal{R} \subsetneq \mathcal{F}$ denote a (reference) subformula of $\mathcal{F}$ which is to be extended to be equivalent to $\mathcal{F}$. Observe that

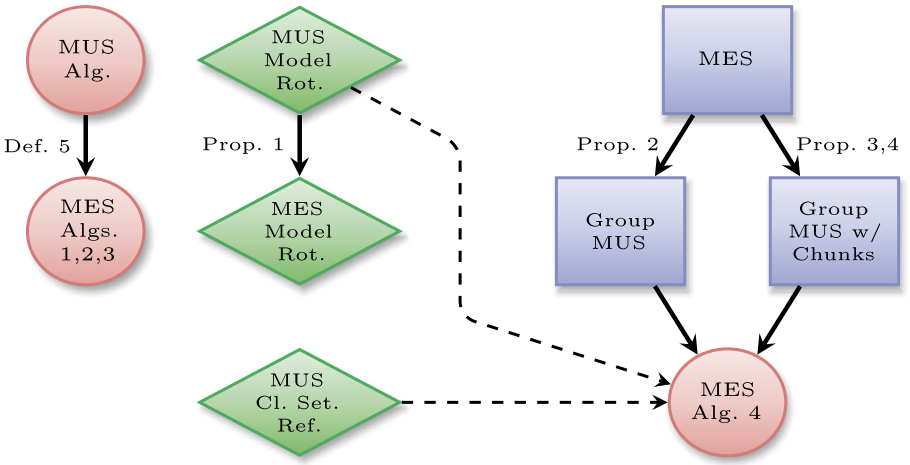

Fig. 1. Approaches to MES extraction. Algorithms are represented by circle (pink) shapes, optimization techniques are represented by diamonds (green), and the various function computation problems are represented by squares (blue).

$\mathcal{F} \models \mathcal{R}$, and so, in case $\mathcal{R} \not\models \mathcal{F}$, our goal is to extend $\mathcal{R}$ with a clause $c \in \mathcal{F} \setminus \mathcal{R}$, such that $\mathcal{R} \cup \{c\} \models \mathcal{F}$, and so $\mathcal{R} \cup \{c\} \equiv \mathcal{F}$. We will refer to such clauses c as *witnesses of equivalence* of the formulas $\mathcal{R} \cup \{c\}$ and $\mathcal{F}$, since they are *required* for the certification of the equivalence of the two formulas.

Definition 5 *(Witness of equivalence).* Let $\mathcal{S}$ be a subformula of $\mathcal{F}$, $\mathcal{S} \subsetneq \mathcal{F}$, with $\mathcal{S} \not\models \mathcal{F}$, and let $c \in \mathcal{F}$. If $\mathcal{S} \cup \{c\} \models \mathcal{F}$, then c is a *witness* of $\mathcal{S} \cup \{c\} \equiv \mathcal{F}$.

Observation 1. *If c is a witness of $\mathcal{S} \cup \{c\} \equiv \mathcal{F}$, then the clause c is irredundant in the formula $\mathcal{S} \cup \{c\}$ (see Definition 3).*

When constructing an MES, Definition 5 identifies a clause that is necessary for the MES being constructed (Observation 1). Hence, this definition enables us to adapt *any* MUS extraction algorithm for MES extraction. The remainder of this section illustrates how this can be achieved. Let $\mathcal{F}$ be a CNF formula partitioned as follows, $\mathcal{F} = \mathcal{E} \cup \mathcal{R} \cup \mathcal{S}$. A subformula $\mathcal{R}$ is redundant in $\mathcal{F}$ iff $\mathcal{F} \equiv \mathcal{E} \cup \mathcal{S}$, or, equivalently, $\mathcal{E} \cup \mathcal{S} \models \mathcal{R}$. Given an under-approximation $\mathcal{E}$ of an MES of $\mathcal{F}$, and a working subformula $\mathcal{S} \subsetneq \mathcal{F}$, the objective is to decide whether $\mathcal{E}$ and $\mathcal{S}$ entail all the other clauses of $\mathcal{F}$. Current MUS extraction algorithms can be organized as (see Section 2.1): (i) deletion-based [29,30], (ii) insertion-based [31,65], (iii) insertion-based with dichotomic search [32], (iv) QuickXplain [33], and (v) progression-based [34].

The pseudo-code for deletion-based MES extraction is shown in Algorithm 1. Note that throughout the article we use the function `SelectRemoveClause`($\mathcal{S}$), which returns some arbitrary clause from the given set $\mathcal{S}$ and as a side-effect it removes it from the set. At each step, the algorithm uses an MES under-approximation $\mathcal{E}$, a set of (remaining) clauses $\mathcal{S}$, and a target clause c, to check whether $\mathcal{E} \cup \mathcal{S} \models c$. If this is not the case, i.e. if $\mathcal{E} \cup \mathcal{S} \not\models c$, then c is witness of $\mathcal{E} \cup \mathcal{S} \cup \{c\} \equiv \mathcal{F}$, and so c is added to $\mathcal{E}$; otherwise, c is discarded. Observe that the deletion-based MES extraction algorithm corresponds to a direct implementation of the definition of redundant clause (e.g. see [3]). Correctness of the algorithm is shown by the following theorem.

Theorem 1. *Algorithm 1 is correct.*

Proof. The algorithm is terminating as the size of the set $\mathcal{S}$ decreases by 1 in each iteration and the algorithm stops when the set is empty.

To prove partial correctness consider the invariant that $\mathcal{E} \cup \mathcal{S} \equiv \mathcal{F}$ and all clauses in $\mathcal{E}$ are irredundant w.r.t. $\mathcal{E} \cup \mathcal{S}$. The invariant is established by the initialization since $\mathcal{S} = \mathcal{F}$ and $\mathcal{E} = \emptyset$. To prove that the invariant is preserved by the loop consider the sets $\mathcal{S}_0$ and $\mathcal{E}_0$ at beginning of an iteration and the sets $\mathcal{S}_1$, $\mathcal{E}_1$ at the end of the iteration. Note that $\mathcal{S}_1 = \mathcal{S}_0 \setminus \{c\}$ for $c =$ `SelectRemoveClause`($\mathcal{S}_0$). By induction hypothesis, $\mathcal{E}_0 \cup \mathcal{S}_0 = \mathcal{E}_0 \cup \mathcal{S}_1 \cup \{c\} \equiv \mathcal{F}$ and all clauses in $\mathcal{E}_0$ are irredundant w.r.t. $\mathcal{E}_0 \cup \mathcal{S}_0$. The invariant is preserved as c is added to $\mathcal{E}_0$ only if it is irredundant w.r.t. $\mathcal{E}_0 \cup \{c\} \cup \mathcal{S}_1$. Partial correctness of the algorithm follows from the invariant since upon the loop's termination, $\mathcal{S} = \emptyset$ and therefore $\mathcal{E}$ is equivalent to $\mathcal{F}$ and does not contain any irredundant clause. □

It is worth to note that the performance of the deletion-based algorithm, as well as other algorithms for MES extraction presented in this paper, is affected to a large degree by the clause-selection heuristic used in the implementation of the function `SelectRemoveClause`. This is similar to the case of the algorithms for the computation of MUSes, where a number of clause selection heuristics have been developed (e.g. [5,66]). In the implementation of the MES algorithms

Algorithm 1: Plain deletion-based MES extraction.

```
   Input  : Formula F
   Output: MES E
1  begin
2      S ← F
3      E ← ∅                                              // MES under-approximation
4      while S ≠ ∅ do
5          c ← SelectRemoveClause(S)                      // Get a clause from S
6          if SAT(E ∪ S ∪ {¬c}) then
7              E ← E ∪ {c}                                // Add clause c to E if E∪S⊭c
8      return E                                           // Final E is MES
9  end
```

Algorithm 2: Plain insertion-based MES extraction.

```
   Input  : Formula F = {c_1, ..., c_m}
   Output: MES E
1  begin
2      W ← F
3      E ← ∅                                              // MES under-approximation
4      while W ≠ ∅ do
5          (S, c_r) ← (∅, ∅)
6          while SAT(E ∪ S ∪ CNF(¬W)) do
7              c_r ← SelectRemoveClause(W)                // Extend S while E∪S⊭W
8              S ← S ∪ {c_r}
9          if S = ∅ then break                            // Fixpoint reached
10         E ← E ∪ {c_r}                                  // c_r is in MES
11         W ← S \ {c_r}
12     return E                                           // Final E is an MES
13 end
```

evaluated in this paper `SelectRemoveClause` returns clauses in the order of appearance in the input formula, however, clearly, the development of intelligent clause-selection heuristics for MES computation is an important objective, and is a subject of future research.

The pseudo-code for insertion-based linear and dichotomic search MES extraction are shown in Algorithms 2 and 3, respectively. Both algorithms iteratively add clauses to set $\mathcal{S}$ while $\mathcal{E} \cup \mathcal{S} \not\models \mathcal{W}$. The last clause included in $\mathcal{S}$ such that $\mathcal{E} \cup \mathcal{S} \models \mathcal{W}$ is the witness of equivalence. The main difference between Algorithms 2 and 3 is how the witness of equivalence is searched for. The following theorem shows correctness of Algorithm 2, correctness of Algorithm 3 is shown similarly.

Theorem 2. *Algorithm 2 is correct.*

Proof. The inner loop of the algorithm is terminating as the size of $\mathcal{W}$ decreases by 1 in each iteration and therefore the loop is guaranteed to stop when $\mathcal{W} = \emptyset$ because $\mathcal{E} \cup \mathcal{S} \cup \mathrm{CNF}(\neg\mathcal{W})$ is necessarily unsatisfiable for $\mathcal{W} = \emptyset$ as $\neg\mathcal{W} = \bot$ in such case. Note, however, that the loop might stop earlier. The outer loop is terminating as at least one clause is removed from $\mathcal{W}$ and added to $\mathcal{E}$ in each iteration of the outer loop (the clause c_r). Hence, the loop is guaranteed to terminate when $\mathcal{W} = \emptyset$. Again, the loop might terminate earlier, which is when the inner loop performs no iterations and leaves the set $\mathcal{S}$ empty, which breaks the outer loop on line 9.

To prove partial correctness consider the invariant that $\mathcal{E} \cup \mathcal{W} \equiv \mathcal{F}$, $\mathcal{E} \cup \mathcal{W} \subseteq \mathcal{F}$, and that all clauses in $\mathcal{E}$ are irredundant w.r.t. $\mathcal{E} \cup \mathcal{W}$. The invariant is established by the initialization since $\mathcal{W} = \mathcal{F}$ and $\mathcal{E} = \emptyset$. To prove that the invariant is preserved by the outer loop, consider the sets $\mathcal{W}_0$ and $\mathcal{E}_0$ at the beginning of an iteration and the sets $\mathcal{W}_1$ and $\mathcal{E}_1$ at the end of the iteration. Note that $\mathcal{E}_1 = \mathcal{E}_0 \cup \{c_r\}$ for c_r obtained from the inner loop. We observe that the inner loop moves clauses from $\mathcal{W}_0$ into the intermediate set $\mathcal{S}$ until the SAT tests returns "no". When the SAT test returns "no", then the negation of the argument formula is a tautology. Hence $\mathcal{E}_0 \cup \mathcal{S} \models (\mathcal{W}_0 \setminus \mathcal{S})$ once the inner loop terminates. This guarantees that $\mathcal{E}_0 \cup \mathcal{S} \equiv \mathcal{E}_0 \cup \mathcal{W}_0$. From induction hypothesis we have $\mathcal{E}_0 \cup \mathcal{S} \equiv \mathcal{F}$. Hence, $\mathcal{E}_1 \cup \mathcal{W}_1 \equiv \mathcal{F}$ since $\mathcal{E}_1 \cup \mathcal{W}_1 = \mathcal{E}_0 \cup \mathcal{S}$. To establish that all clauses in $\mathcal{E}_1$ are irredundant for $\mathcal{E}_1 \cup \mathcal{W}_1$ we first consider the clauses in $\mathcal{E}_0$. Since $\mathcal{E}_1 \cup \mathcal{W}_1 \subseteq \mathcal{E}_0 \cup \mathcal{W}_0$, no clause in $\mathcal{E}_0$ could have become redundant. The clause c_r is irredundant due to the inner loop, which guarantees that $\mathcal{E}_0 \cup \mathcal{S} \equiv \mathcal{F}$ and $\mathcal{E}_0 \cup \mathcal{S} \setminus \{c_r\} \not\equiv \mathcal{F}$ (c_r is a witness of equivalence for $\mathcal{E}_0 \cup \mathcal{S}$). Partial correctness of the algorithm follows directly from the invariant as $\mathcal{W}$ is empty upon termination and therefore $\mathcal{E} \equiv \mathcal{F}$ with no redundant clauses. □

Algorithm 3: MES extraction with dichotomic search.

```
   Input  : Formula F = {c_1, ..., c_m}
   Output: MES E
1  begin
2      W ← F
3      E ← ∅                                                  // MES under-approximation
4      while W ≠ ∅ do
5          (min, mid, max) ← (0, 0, |W|)
6          repeat
7              S ← {c_1, ..., c_mid}                          // Extract sub-sequence of W
8              if SAT(E ∪ S ∪ CNF(¬(W \ S))) then
9                  min ← mid + 1                              // Extend S if E ∪ S ⊭ W \ S
10             else
11                 max ← mid                                  // Reduce S if E ∪ S ⊨ W \ S
12             mid ← ⌊(min + max)/2⌋
13         until min = max
14         if min > 0 then
15             E ← E ∪ {c_min}
16         W ← {c_i | i < min}
17     return E                                               // Final E is an MES
18 end
```

Recent experimental data indicates that deletion-based MUS extraction algorithms (cf. the variant in [23]), as well as progression-based [34], are the most efficient currently available algorithms for practical instances. This is despite the fact that the insertion-based algorithms have a number of theoretical advantages over the deletion-based (see [23] for a detailed discussion). Since insertion-based MES algorithms require SAT solver calls with (possibly large) complemented formulas, deletion-based MES extraction algorithms are expected to outperform the insertion-based ones. This is confirmed by the results in Section 7.

Efficient MUS extraction algorithms [21–23] *must* use a number of additional techniques for reducing the number of SAT solver calls. These techniques include *clause set refinement* [21,23] and *model rotation* [23,24]. The next section shows how model rotation can be used in MES extraction. In contrast, clause set refinement *cannot* be applied in the algorithms described above.

Example 1. Let $\mathcal{F} = (x_1) \land (x_1 \lor x_3) \land (x_2)$, and consider the execution of Algorithm 1. First, clause (x_1) is removed and the resulting formula is satisfiable with $x_1 = 0$, $x_2 = x_3 = 1$; hence (x_1) is irredundant. Second, clause $(x_1 \lor x_3)$ is removed and the resulting formula is unsatisfiable; hence $(x_1 \lor x_3)$ is redundant. Moreover, the computed unsatisfiable core is $\{(x_1), (\neg x_1)\}$, where $(\neg x_1)$ is taken from $\neg(x_1 \lor x_3)$. However, (x_2) is not in the unsatisfiable core, but it cannot be removed. Subsequently, $(x_1 \lor x_3)$ is forgotten while (x_2) is deemed irredundant, which results in the output of the algorithm: $(x_1) \land (x_2)$.

As illustrated by the previous example, since MES extraction algorithms require adding the negation of a subformula of $\mathcal{F}$, the computed unsatisfiable cores depend on this negation, which changes as the algorithm executes. Thus, a computed unsatisfiable core provides no information about which clauses need not be considered further. Despite this negative result, Sections 3.3 and 3.4 develop solutions for MES extraction that enable clause set refinement.

3.2. Using model rotation in MES extraction

The definition of irredundant clause (see Definition 3) can be associated with specific truth assignments, that can serve as *witnesses* of irredundancy.

Proposition 1. *A clause $c \in \mathcal{F}$ is irredundant in $\mathcal{F}$ if and only if there exists a truth assignment μ, such that $(\mathcal{F} \setminus \{c\})[\mu] = 1$ and $c[\mu] = 0$.*

Proof. Follows directly from the definition of an irredundant clause and the semantics of $\mathcal{F} \setminus \{c\} \not\models c$. □

In the context of MUS extraction, Proposition 1 is used as a basis of *model rotation*, cf. Section 2.2. Proposition 1 has also been used in the context of local search for MUS extraction [67].

In the context of MES extraction, model rotation can be instrumented as follows. If a SAT solver call returns satisfiable (see, for example, line 6 in Algorithm 1), then we use the model computed by the solver to look for other irredundant clauses using Proposition 1. For instance in Algorithm 1 once we obtain a model μ of $\mathcal{E} \cup \mathcal{S} \cup \{\neg c\}$, we modify μ to μ_l

by flipping the value of a literal of $l \in c$ and look for a clause $d \in \mathcal{S}$ so that $(\mathcal{E} \cup \mathcal{S} \setminus \{d\})[\mu_l] = 1$ and $d[\mu_l] = 0$ (note that $c[\mu_l] = 1$). Such clause d is deemed irredundant due to Proposition 1. Note that such clause can be discovered without a second SAT call. This process goes over all literals $l \in c$ and it continues recursively from the newly detected irredundant clause (if any).

Model rotation for MES extraction can be improved further. Since the formula is satisfiable, model rotation may reach assignments where *all* clauses are satisfied (note that this situation is not possible in MUS extraction). In this case, rather than terminating the process, clauses with the smallest number of satisfied literals are selected, the satisfied literals are flipped, and again one checks whether the formula has a single falsified clause. As demonstrated in Section 7, this *improved model rotation* is very effective for redundancy removal.

3.3. A reduction of MES to group-MUS

As argued in Section 3.1, although MUS extraction algorithms can be modified for MES extraction, clause set refinement [21–23] cannot be used. In the context of MUS computation, this technique is *paramount* for reducing the number of SAT solver calls in instances with many redundant clauses. This section develops a reduction of MES computation problem to group-MUS computation problem [25,22] — recall Definition 2. A key advantage of this reduction is that it enables clause set refinement.

Proposition 2 *(MES to group-MUS reduction). Given a CNF formula $\mathcal{F}$, and any $\mathcal{E} \subseteq \mathcal{F}$, let $R_{\mathcal{F}}(\mathcal{E})$ be the group-CNF formula $\mathcal{D} \cup \bigcup_{c \in \mathcal{E}} \mathcal{G}_c$, where $\mathcal{D} = \text{CNF}(\neg\mathcal{F})$ is the don't care group, and $\mathcal{G}_c = \{c\}$. Then, $\mathcal{E}$ is an MES of $\mathcal{F}$ if and only if $R_{\mathcal{F}}(\mathcal{E})$ is a group-MUS of $R_{\mathcal{F}}(\mathcal{F})$.*

Proof. We prove both directions simultaneously. Since $R_{\mathcal{F}}(\mathcal{E}) = \text{CNF}(\neg\mathcal{F}) \cup \mathcal{E}$, we have that $\mathcal{E} \models \mathcal{F}$ (and so $\mathcal{E} \equiv \mathcal{F}$), if and only if $R_{\mathcal{F}}(\mathcal{E})$ is unsatisfiable. Let c be any clause of $\mathcal{E}$. Then, by Proposition 1, c is irredundant in $\mathcal{E}$ if and only if there exists an assignment μ, such that $(\mathcal{E} \setminus \{c\})[\mu] = 1$ and $c[\mu] = 0$. Equivalently, there is an extension μ' of μ such that $\text{CNF}(\neg\mathcal{F})[\mu'] = 1$ and $(\mathcal{E} \setminus \{c\})[\mu'] = 1$, i.e. $R_{\mathcal{F}}(\mathcal{E} \setminus \{c\})$ is satisfiable. □

Expressing the MES computation problem as a group-MUS computation problem enables the use of optimization techniques for (group-)MUS extraction in computing irredundant CNF formulas. Most importantly, the clause-set refinement becomes usable and effective again. We demonstrate this with the following example.

Example 2. Consider the CNF formula $\mathcal{F} = (x_1) \wedge (x_1 \vee y_i \vee y_j)$, with $1 \leq i < j \leq k$, and $k \geq 2$. All clauses with a literal in the y variables are redundant, for a total of $k(k-1)/2$ redundant clauses. The reduction in Proposition 2 produces the group-CNF formula $R_{\mathcal{F}}(\mathcal{F})$ with the don't care group $\mathcal{D} = \text{CNF}(\neg\mathcal{F})$, and a singleton group for each clause in $\mathcal{F}$, i.e. $\mathcal{G}_{11} = \{(x_1)\}$, $\mathcal{G}_{ij} = \{(x_1 \vee y_i \vee y_j)\}$, with $1 \leq i < j \leq k$. For this example, let us assume that the group-MUS of $R_{\mathcal{F}}(\mathcal{F})$ is computed using a deletion-based algorithm. Let $\mathcal{G}_{11}$ be the first group to be analyzed. This is done by removing the clause in $\mathcal{G}_{11}$ from the formula. The resulting formula has a model μ, with $\mu(x_1) = 0$ and $\mu(y_i) = \mu(y_j) = 1$, with $1 \leq i < j \leq k$. As a result, (x_1) is declared irredundant, and added back to the formula. Afterwards, pick one of the other groups, e.g. $\mathcal{G}_{12}$. If the corresponding clause is removed from the formula, the resulting formula is unsatisfiable, and so the clause is declared redundant. More importantly, a CDCL SAT solver will produce an unsatisfiable core $\mathcal{U} \subseteq \{x_1\} \cup \text{CNF}(\neg\mathcal{F})$. Hence, clause set refinement serves to eliminate *all* of the remaining groups of clauses, and so the MES is computed with *two* SAT solver calls. As noted earlier, MES algorithms based on adapting existing MUS algorithms are unable to implement clause set refinement, and so cannot drop $k(k-1)/2$ clauses after the second SAT solver call.

Observe that the *group-MUS approach* to the MES extraction problem, i.e. using the reduction in Proposition 2, is *independent* of the actual group-MUS extraction algorithm used. Hence, any existing group-MUS extraction algorithm can be used for the MES extraction problem. In practice, given the significant performance difference between deletion-based and insertion-based MUS extraction algorithms [23], our implementation is based on deletion-based group-MUS extraction and its most recent instantiation, i.e. the *hybrid* approach in [23].

3.4. Incremental reduction of MES to group-MUS

A major drawback of the group-MUS approach is that for large input formulas the resulting instances of SAT can be hard. This is due to the CNF encoding of $\neg\mathcal{F}$ that produces a large disjunction of auxiliary variables. A solution to this issue is based on an *incremental* reduction of MES extraction to group-MUS extraction. Let $\mathcal{T}$ be any subset of clauses of $\mathcal{F}$ — we refer to clauses of $\mathcal{T}$ as *target* clauses, and to the set $\mathcal{T}$ itself as a *chunk* of $\mathcal{F}$. The incremental reduction is based on the observation that in group-MUS approach the redundancy of any target clause $c \in \mathcal{T}$ can be established by analyzing c with respect to $\text{CNF}(\neg\mathcal{T})$ rather than $\text{CNF}(\neg\mathcal{F})$. This observation is stated precisely below.

Proposition 3. *Let $\mathcal{T} \subseteq \mathcal{F}$ be a set of target clauses. For any $\mathcal{E} \subseteq \mathcal{T}$, let $R_{\mathcal{T}}(\mathcal{E})$ be the group-CNF formula $\mathcal{D} \cup \bigcup_{c \in \mathcal{E}} \mathcal{G}_c$, where $\mathcal{D} = \mathcal{F} \setminus \mathcal{T} \cup \text{CNF}(\neg\mathcal{T})$ is the don't care group, and $\mathcal{G}_c = \{c\}$. Then, $\mathcal{E}$ is irredundant in $\mathcal{F}$ and $\mathcal{F} \setminus \mathcal{T} \cup \mathcal{E} \equiv \mathcal{F}$ if and only if $R_{\mathcal{T}}(\mathcal{E})$ is a group-MUS of $R_{\mathcal{T}}(\mathcal{T})$.*

Note that $R_{\mathcal{T}}(\mathcal{T})$ is unsatisfiable. Also, in the case when the chunk $\mathcal{T}$ is taken to be the whole formula $\mathcal{F}$, the group-CNF formula $R_{\mathcal{T}}(\mathcal{E})$ is exactly the formula $R_{\mathcal{F}}(\mathcal{E})$ from Proposition 2, and so the claim of Proposition 2 is a special case of the claim of Proposition 3. We omit the proof of Proposition 3 as it essentially repeats the steps of the proof of Proposition 2, using the definition of $R_{\mathcal{T}}(\mathcal{E})$ instead of $R_{\mathcal{F}}(\mathcal{E})$.

For the general case, consider a partition $\mathcal{F}_1, \ldots, \mathcal{F}_k$ of $\mathcal{F}$ into chunks. Then, an MES of $\mathcal{F}$ can be computed by applying the group-MUS approach of Proposition 3 to each chunk. Proposition 3 is applied in order, with already computed irredundant subformulas replacing the original (redundant) subformulas. Explicitly, for the iteration j, $1 \leq j \leq k$, the input group-CNF formula in Proposition 3 is defined as follows:

$$\mathcal{D} \cup \bigcup_{c \in \mathcal{F}_j} \mathcal{G}_c, \quad \text{with } \mathcal{D} = \mathcal{E}_1 \cup \ldots \cup \mathcal{E}_{j-1} \cup \mathcal{F}_{j+1} \cup \ldots \mathcal{F}_k \cup \text{CNF}(\neg\mathcal{F}_j) \tag{1}$$

as a don't care group, where $\mathcal{E}_i$, $1 \leq i < j$, is the computed irredundant set of clauses in $\mathcal{F}_i$, $1 \leq i < j$, and, as before, $\mathcal{G}_c = \{c\}$.

Proposition 4. *Let $\mathcal{F}_1, \ldots, \mathcal{F}_k$ be a partition of $\mathcal{F}$ into chunks. For $1 \leq j \leq k$, let $\mathcal{E}_j$ be the set of clauses obtained from applying group-MUS extraction to the formula in (1), and by considering each $\mathcal{F}_j$ as the set of target clauses. Let $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \cup \mathcal{E}_k$. Then $\mathcal{E}$ is an MES of $\mathcal{F}$.*[1]

Proof. For $1 \leq j \leq k$, let $\mathcal{I}_j = (\bigcup_{r \leq j} \mathcal{E}_r \cup \bigcup_{r > j} \mathcal{F}_r)$, with $\mathcal{I}_0 = \mathcal{F}$. The proposition follows from the following inductive invariant: for $1 \leq j \leq k$, $\mathcal{I}_j \equiv \mathcal{F}$, and the formula $\bigcup_{r \leq j} \mathcal{E}_r$ irredundant in $\mathcal{F}$. The claim of the proposition corresponds to the case $j = k$.

Base case: letting $\mathcal{T} = \mathcal{F}_1$ and $\mathcal{F} = \mathcal{I}_0$ in Proposition 3, we have that $\mathcal{E}_1$ is irredundant in $\mathcal{F}$, and $\mathcal{I}_1 = (\mathcal{F} \setminus \mathcal{F}_1 \cup \mathcal{E}_1) \equiv \mathcal{F}$.

Inductive step: we are applying Proposition 3 to the formula $\mathcal{I}_{j-1}$, letting the target set of clauses $\mathcal{T}$ be the chunk $\mathcal{F}_j$. Then, from the proposition, $\mathcal{E}_j$ is irredundant in $\mathcal{I}_{j-1}$, and since, by the inductive hypothesis, $\mathcal{I}_{j-1} \equiv \mathcal{F}$, we conclude that $\mathcal{E}_j$ is irredundant in $\mathcal{F}$. Also from the proposition we obtain that $\mathcal{I}_j \equiv \mathcal{F}$, thus concluding the proof of the invariant. □

While Proposition 4 can be used to compute an MES of an input formula $\mathcal{F}$ by iteratively calling a group-MUS extractor, it can also be integrated into a unified algorithm to enable certain optimizations (e.g. incremental SAT solving). Algorithm 4 shows the pseudo-code for deletion-based group-MUS approach to MES extraction using chunks. Note that the function SAT$(\cdot)$ returns a triple $(\text{st}, \nu, \mathcal{U})$, which contains the formula's satisfiability in the element st; a satisfying assignment ν when the formula is satisfiable; and an unsatisfiable core $\mathcal{U}$ when the formula is unsatisfiable.

Different chunk sizes can be considered. Concrete examples include chunks of size 1 or a single chunk aggregating all clauses (i.e. the reduction to group-MUS defined in Proposition 2). For chunks of size great than 1, the group-MUS approach has the ability to prove several clauses redundant by using clause-set refinement. Generally, the chunk size in the group-MUS approach controls the *trade-off* between the potential power of clause-set refinement and the difficulty of the instances of SAT given to the SAT solver.

Nevertheless, an essential issue with Algorithm 4 is the selection of the size of the chunks. The current implementation of the algorithm uses chunks of fixed size, independently of the size of the formula. Alternative solutions include using chunk sizes dependent on the size of the formula, and also *adaptive* chunk sizes.

4. Backbone-based pruning

What makes clause-redundancy analysis difficult in practice is the large number of clauses found in practical formulas. Typically, however, the number of variables is significantly smaller than the number of clauses. This motivates the technique presented in this section. The technique first computes for the given formula its *backbone*, which are literals that are true in all of the formula's models [68–70]. Having information about the backbone enables as to identify *certain* redundant clauses. And, it is possible to compute the backbone with a number of SAT calls proportional to the number of variables of the formula.

Definition 6 *(Backbone).* A literal l is a *backbone literal* of a formula $\mathcal{F}$ if l is true in all models of the formula, i.e. $\mathcal{F} \models l$. The *backbone* of a formula comprises its backbone literals.

[1] The proposition is stated slightly informally as to avoid additional notation.

Algorithm 4: Deletion-based group-MUS extraction of MES with chunks.

Input : Formula $\mathcal{F} = \{\mathcal{F}_1, \ldots, \mathcal{F}_k\}$ with k chunks
Output: MES $\mathcal{E}$

begin
  for $j \leftarrow 1$ **to** k **do** // Analyze each of the k chunks
    $\mathcal{D} = \mathcal{E}_1 \cup \ldots \cup \mathcal{E}_{j-1} \cup \mathcal{F}_{j+1} \cup \ldots \cup \mathcal{F}_k \cup \text{CNF}(\neg\mathcal{F}_j)$ // Don't care group
    $\mathcal{W} = \mathcal{F}_j$ // Target group of clauses $\mathcal{F}_j$
    $\mathcal{E}_j \leftarrow \emptyset$ // Irredundant clauses in chunk j
    while $\mathcal{W} \neq \emptyset$ **do**
      $c \leftarrow \texttt{SelectRemoveClause}(\mathcal{W})$
      $(\text{st}, \nu, \mathcal{U}) = \text{SAT}(\mathcal{D} \cup \mathcal{E}_j \cup \mathcal{W})$
      if st = true **then** // If SAT, c is irredundant in $\mathcal{F}$
        $\mathcal{E}_j \leftarrow \mathcal{E}_j \cup \{c\}$
        $(\mathcal{W}, \mathcal{E}_j) \leftarrow \texttt{Rotate}(\mathcal{W}, \mathcal{E}_j, \nu)$ // Apply model rotation
      else
        $\mathcal{W} \leftarrow \mathcal{U} \cap \mathcal{W}$ // Clause-set refinement
  $\mathcal{E} = \mathcal{E}_1 \cup \ldots \cup \mathcal{E}_k$
  return $\mathcal{E}$ // $\mathcal{E}$ is an MES
end

Note that a test whether a literal l is a backbone literal can be achieved by a single SAT call. In particular, l is a backbone literal of $\mathcal{F}$ iff the formula $\mathcal{F} \wedge (\neg l)$ is unsatisfiable. This means that the whole backbone can be computed with a number of SAT calls proportional to the number of *variables* of the formula. However, a number of optimization for computing the backbone exist [71].

Knowing that a literal is a backbone literal is not itself sufficient for discovering redundant clauses. If, however, we additionally know that l is a backbone literal *because* of some subformula $\mathcal{R} \subseteq \mathcal{F}$, all clauses of $\mathcal{F} \setminus \mathcal{R}$ that contain l are redundant.

Proposition 5. *Let l be a backbone literal of a formula $\mathcal{F}$, i.e. $\mathcal{F} \models l$. Further, let $\mathcal{R} \subseteq \mathcal{F}$ be a subformula of $\mathcal{F}$ such that $\mathcal{R} \models l$. The following set $\mathcal{S}$ of clauses is redundant:*

$$\mathcal{S} = \{c \in \mathcal{F} \mid l \in c \wedge c \notin \mathcal{R}\}$$

Proof. Since $\mathcal{S}$ is disjoint from $\mathcal{R}$, it holds for any clause $c \in \mathcal{S}$ that $\mathcal{F} \setminus \{c\} \models l$. Because $l \in c$, then also $\mathcal{F} \setminus \{c\} \models c$ (see Definition 3). □

Proposition 5 enables us to remove as redundant *all* the clauses in $\mathcal{S}$. This is because once any clause $c \in \mathcal{S}$ is a removed from $\mathcal{F}$, the same proposition applies to the formula $\mathcal{F} \setminus \{c\}$ and the same $\mathcal{R}$.

Proposition 5 suggests how to modify a backbone-computation algorithm into a technique for identifying certain redundant clauses (but not necessarily all). Most complete algorithms for backbone-computation detect that l is a backbone by issuing a SAT call on the formula $\mathcal{F}_l = \mathcal{F} \wedge (\neg l)$. Since l is a backbone, $\mathcal{F}_l$ is unsatisfiable. Now let $\mathcal{R}' \subseteq \mathcal{F}_l$ be an unsatisfiable core of $\mathcal{F}_l$ (see Section 2.2 for description of unsatisfiable cores). Since $\mathcal{R}'$ is unsatisfiable, it holds that $\mathcal{R}' \setminus \{\neg l\} \models l$. Hence, $\mathcal{R} = \mathcal{R}' \setminus \{\neg l\}$ provides us with a reason for why l is a backbone at which point we can remove all clauses that contain l and are not in $\mathcal{R}$.

A variety of techniques exist to speed up backbone computation [71]. A number of these techniques enable us to filter out literals that are certainly *not* backbone literals. An example of such technique is the filtering of all literals that are false in some model of $\mathcal{F}$. Such techniques can be added to our redundancy pruning algorithm. However, we should note that there is one technique that is not in general sound. If a literal l is a backbone literal, adding this literal as a unit clause to $\mathcal{F}$ does not change the models of $\mathcal{F}$. Hence, whenever a backbone-computation algorithm detects that l is a backbone, it adds it to $\mathcal{F}$, which will help future SAT calls. In redundancy pruning, however, this is *not sound* because the newly added clause l is not part of the original formula. For instance, for the formula $(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee y)$ we may detect that x is a backbone literal. Adding x to the formula enables us to derive that y is a backbone literal with the reason $(x) \wedge (\neg x \vee y)$. This would enable us marking the clause $x \vee y$ as redundant.

We should stress that the above described pruning algorithm is not complete. For instance, the formula $(x \vee y) \wedge (x \vee y \vee z)$ is redundant but has an empty backbone. Further, the technique is highly dependent on the size of the cores that will be returned by the SAT solver. If, for instance, the reason $\mathcal{R}$ is the whole formula $\mathcal{F}$, no clauses will be pruned. Nevertheless, backbone-based pruning is likely to be a method cheaper than complete MES computation because the number of a SAT calls is proportional to the total number of variables rather than clauses. Hence, it can be used as an MES approximation technique or as a preprocessing step to a complete algorithm.

5. Certification of correctness

In some applications, it is paramount to guarantee that the computed subformula is indeed irredundant and, possibly more importantly, that each computed irredundant subformula $\mathcal{E}$ is equivalent to the original CNF formula $\mathcal{F}$. For such applications, it is useful to protect the user from possible bugs in the reduction algorithm by independent certification.

Given a computed CNF formula $\mathcal{E}$, it is simple to validate whether it is irredundant. Essentially, one can run one of the algorithms outlined in earlier sections. The problem of validating whether $\mathcal{E} \equiv \mathcal{F}$ looks more challenging.

To check whether $\mathcal{E} \equiv \mathcal{F}$, it suffices to exhibit a truth assignment that is a model of one formula and not of the other. This condition corresponds to testing the satisfiability of the following CNF formula:

$$\big(\mathcal{E} \wedge \text{CNF}(\neg\mathcal{F})\big) \vee \big(\mathcal{F} \wedge \text{CNF}(\neg\mathcal{E})\big) \tag{2}$$

Clearly, the resulting instances of SAT are expected to be hard to solve, given the disjunction and the negation of formulas in (2). Nevertheless, it is also the case that $\mathcal{E} \subseteq \mathcal{F}$ and so, $\mathcal{F} \models \mathcal{E}$. Hence it is only necessary to check whether $\mathcal{E} \models \mathcal{F}$. Since $\mathcal{F} = \mathcal{E} \wedge \mathcal{R}$, $\mathcal{E} \models \mathcal{F}$ can be represented as $\mathcal{E} \models \mathcal{E} \wedge \mathcal{R}$, or equivalently, $\mathcal{E} \wedge (\neg\mathcal{E} \vee \neg\mathcal{R}) \models \bot$, which simplifies to $\mathcal{E} \wedge \neg\mathcal{R} \models \bot$. This condition corresponds to testing the satisfiability of the following CNF formula:

$$\mathcal{E} \cup \text{CNF}(\neg\mathcal{R}) \tag{3}$$

If $\mathcal{E} \cup \text{CNF}(\neg\mathcal{R})$ has a model, then one can satisfy $\mathcal{E}$ while unsatisfying one or more clauses in $\mathcal{R}$. Hence, the two formulas would not be equivalent.

Although easier than (2), (3) can still result in hard instances of SAT (similarly to what happens with the reduction of MES to group-MUS extraction). A technique to reduce the complexity of the resulting instances of SAT is to partition $\mathcal{R}$ into chunks, and check each chunk separately for equivalence. The objective is to check whether $\mathcal{E} \models \mathcal{R}$. If this condition holds, then $\mathcal{E} \equiv \mathcal{F}$; otherwise $\mathcal{E} \not\equiv \mathcal{F}$. If $\mathcal{R}$ is partitioned into a number of subformulas (or chunks), we get $\mathcal{R} = \mathcal{R}_1 \wedge \ldots \wedge \mathcal{R}_k$, and so the condition becomes, $\mathcal{E} \models \mathcal{R}_1 \wedge \ldots \wedge \mathcal{R}_k$, that can also be represented as $\mathcal{E} \wedge (\neg\mathcal{R}_1 \vee \ldots \vee \neg\mathcal{R}_k) \models \bot$. This condition holds if and only if, $\forall_{1 \leq j \leq k}\ \mathcal{E} \wedge \neg\mathcal{R}_j \models \bot$. Hence, the use of chunks allows splitting a potentially hard (and believed unsatisfiable) instance of SAT, into k (likely) easier (and also believed unsatisfiable) instances of SAT.

6. Extension to group-MES computation

The MES computation and certification algorithms presented in the previous sections can be extended to the computation of group-oriented MESes of CNF formulas, defined as follows.

Definition 7 *(Group-oriented MES).* Given an explicitly partitioned CNF formula $\mathcal{F} = \mathcal{D} \cup \mathcal{G}_1 \cup \ldots \cup \mathcal{G}_n$ (a *group-CNF formula*), a *group oriented MES* (or, *group-MES*) of $\mathcal{F}$ is a subset $\mathcal{F}' = \mathcal{D} \cup \mathcal{G}_{i_1} \cup \ldots \cup \mathcal{G}_{i_k}$ of $\mathcal{F}$ (with $1 \leq i_j \leq n$ for $1 \leq j \leq k \leq n$) such that $\mathcal{F} \equiv \mathcal{F}'$, and for every $1 \leq j \leq k$, $\mathcal{F} \not\equiv \mathcal{F}' \setminus \mathcal{G}_{i_j}$.

Recall that the first set of MES computation algorithms presented in this paper are based on the conversion of MUS computation algorithms to MES computation via the definition of witness of equivalence (Definition 5). To adopt this set of algorithms to group-MES setting, we extend this definition to the group CNF setting in the following manner.

Definition 8 *(Witness of equivalence (group CNF)).* Let $\mathcal{F} = \mathcal{D} \cup \mathcal{G}_1 \cup \ldots \cup \mathcal{G}_n$ be a group-CNF formula, and let $S = \mathcal{D} \cup \mathcal{G}_{i_1} \cup \ldots \cup \mathcal{G}_{i_k}$ be a subformula of $\mathcal{F}$, with $\mathcal{S} \subsetneq \mathcal{F}$ and $\mathcal{S} \not\models \mathcal{F} \setminus \mathcal{S}$, and let $\mathcal{G} \subseteq \mathcal{F} \setminus \mathcal{S}$ be one of the groups of $\mathcal{F}$. If $\mathcal{S} \cup \mathcal{G} \models \mathcal{F} \setminus (\mathcal{S} \cup \mathcal{G})$, then $\mathcal{G}$ is a *witness* of $\mathcal{S} \cup \mathcal{G} \equiv \mathcal{F}$.

Then, the algorithms from Section 3.1 are extended to group-MES setting by changing the "granularity" from clause-based to group-based. As an illustrative example, consider the pseudo-code of the deletion-based group-MES extraction algorithm presented in Algorithm 5 and contrast it with the deletion-based algorithm for plain MES computation problem (Algorithm 1). As with the previously described algorithms, the CNF representation of the formula $\neg\mathcal{G}$ is constructed as Section 2.

Model rotation can be extended to group-MES setting via the following generalization of Proposition 1:

Proposition 6. *Let $\mathcal{F} = \mathcal{D} \cup \mathcal{G}_1 \cup \ldots \cup \mathcal{G}_n$ be a group-CNF formula. A group $G_i \in \mathcal{F}$ is irredundant in $\mathcal{F}$ (i.e. $\mathcal{F} \setminus \mathcal{G}_i \models \mathcal{G}_i$) if and only if there exists a truth assignment μ, such that $(\mathcal{F} \setminus \mathcal{G}_i)[\mu] = 1$ and $\mathcal{G}_i[\mu] = 0$, i.e. there exists a clause $c \in \mathcal{G}_i$ such that $c[\mu] = 0$.*

As it is the case with the application of model rotation in the group-MUS setting, the algorithm must be adjusted so that it constructs new assignments that satisfy all clauses of the currently falsified group, and the group $\mathcal{D}$.

The MES computation algorithms based on the reduction to group-MUS problem (see Section 3.3) can also be extended to the group-MES setting. Noteworthy is the fact that the don't care group $\mathcal{D} = \text{CNF}(\neg\mathcal{F})$ used in Proposition 2 to construct the group-CNF formula $R_{\mathcal{F}}(\mathcal{E})$ from a subformula $\mathcal{E} \subseteq \mathcal{F}$ does not need to include the negation of the don't care group

Algorithm 5: Deletion-based group-MES extraction.

Input : Group-CNF formula $\mathcal{F} = \mathcal{D} \cup \mathcal{G}_1 \ldots \cup \mathcal{G}_n$
Output: Group-MES $\mathcal{E}$ of $\mathcal{F}$

```
1  begin
2      S ← F
3      E ← ∅                                              // MES under-approximation
4      while S ≠ ∅ do
5          G ← SelectRemoveGroup(S)                       // Get a group from S (G ≠ D)
6          W ← G
7          if SAT(E ∪ S ∪ CNF(¬G)) then
8              E ← E ∪ G                                  // Add group G to E if E ∪ S ⊭ G
9      return E                                           // Final E is a group-MES of F
10 end
```

on the formula used as an input to the group-MES computation problem. The following proposition provides the necessary details.

Proposition 7 *(Group-MES to group-MUS reduction). Let $\mathcal{F} = \mathcal{D} \cup \mathcal{G}_1 \cup \ldots \cup \mathcal{G}_n$ be a group-CNF formula, and let $\mathcal{E} = \mathcal{D} \cup \mathcal{G}_{i_1} \cup \ldots \cup \mathcal{G}_{i_k}$ be a subformula of $\mathcal{F}$ (possibly $\mathcal{F}$ itself). Let $R_{\mathcal{F}}(\mathcal{E})$ be the group-CNF formula $\mathcal{D}^* \cup \mathcal{G}_{i_1} \cup \ldots \cup \mathcal{G}_{i_k}$, where $\mathcal{D}^* = \mathcal{D} \cup \text{CNF}(\neg(\mathcal{F} \setminus \mathcal{D}))$ is the don't care group. Then, $\mathcal{E}$ is a group-MES of $\mathcal{F}$ if and only if $R_{\mathcal{F}}(\mathcal{E})$ is a group-MUS of $R_{\mathcal{F}}(\mathcal{F})$.*

Notice that in plain-MES setting, the don't care group $\mathcal{D}$ of the formula $\mathcal{F}$ is empty, and so the don't care group $\mathcal{D}^*$ of the formula $R_{\mathcal{F}}(\mathcal{E})$ is exactly the formula CNF($\neg\mathcal{F}$), as in Proposition 2. Thus, Proposition 7 is the extension of Proposition 2 to the group-MES setting.

Proof. First, we show that $\mathcal{E} \models \mathcal{F}$ if and only if the formula $R_{\mathcal{F}}(\mathcal{E})$ is unsatisfiable. Note that $\mathcal{E} \equiv \mathcal{F}$ directly follows as $\mathcal{F} \models \mathcal{E}$ holds trivially since $\mathcal{E} \subseteq \mathcal{F}$. $\mathcal{E} \models \mathcal{F}$ if and only $\mathcal{E} \wedge \neg\mathcal{F}$ is unsatisfiable. We have, $\mathcal{E} \wedge \neg\mathcal{F} \equiv \mathcal{E} \wedge \neg(\mathcal{D} \wedge (\mathcal{F} \setminus \mathcal{D})) \equiv (\mathcal{E} \wedge \neg\mathcal{D}) \vee (\mathcal{E} \wedge \neg(\mathcal{F} \setminus \mathcal{D}))$. Since $\mathcal{D} \subseteq \mathcal{E}$ the first disjunct is unsatisfiable, and so $\mathcal{E} \wedge \neg\mathcal{F} \equiv \mathcal{E} \wedge \neg(\mathcal{F} \setminus \mathcal{D})$, and the latter formula is equisatisfiable with $R_{\mathcal{F}}(\mathcal{E})$.

Let now, $\mathcal{G}_i$ be any group of $\mathcal{E}$. Then, by Proposition 6, $\mathcal{G}_i$ is irredundant in $\mathcal{E}$ if and only if there exists an assignment μ, such that $(\mathcal{E} \setminus \mathcal{G}_i)[\mu] = 1$ and $\mathcal{G}_i[\mu] = 0$. Since $\mathcal{D} \subseteq (\mathcal{E} \setminus \mathcal{G}_i)$, we have $\mathcal{D}[\mu] = 1$, and since $\mathcal{G}_i \subseteq (\mathcal{F} \setminus \mathcal{D})$, $(\mathcal{F} \setminus \mathcal{D})[\mu] = 0$ and so for an extension μ' of μ, CNF($\neg(\mathcal{F} \setminus \mathcal{D})$)[$\mu'$] = 1. Thus, $\mathcal{D}^*[\mu'] = 1$, and so $R_{\mathcal{F}}(\mathcal{E} \setminus \mathcal{G}_i)$ is satisfiable. The "if" direction is shown by following the above steps in reverse order. □

Propositions 3 and 4, which are used to develop the group-MUS based MES extraction algorithm with chunks (Algorithm 4), can be extended to the group-MES setting analogously. We omit the cumbersome details.

7. Experimental results

The algorithms described in the previous sections were implemented within the MUS extraction framework of a state-of-the-art MUS extractor `MUSer2`[2]; the framework was configured to use SAT solver `picosat-935` [39] in the incremental mode. The experiments were performed on an HPC cluster, where each node is dual quad-core Intel Xeon E5450 3 GHz with 32 GB of memory. Each algorithm was run with a timeout of 1800 seconds and a memory limit of 4 GB per input instance. To evaluate the algorithms, we selected 300 problem instances from practical application domains of SAT used in past SAT competitions.[3] The instances were selected using the following criteria: the instance is solvable within 1 second by `picosat-935`, and the number of clauses in the instance is less than 100,000. These criteria were derived from the worst-case analysis of deletion-based MES algorithms (whereby the number of SAT calls is linear in the size of the input formula) and our previous experience with the effects various optimization techniques in the context of MUS extraction.

We will refer to the evaluated algorithms using the following abbreviations: DEL represents the implementation of the deletion-based algorithm (Algorithm 1); INS (resp. DICH) is the implementation of insertion-based (resp. dichotomic) algorithms from Section 3.1; +MR indicates the addition of model rotation, while +IMR indicates the addition of the improved version of model rotation (see Section 3.2); GRP-MUS is the implementation of the reduction of MES to group-MUS from Section 3.3; CHUNK-*x* is the implementation of the deletion-based chunked group-oriented MUS algorithm from Section 3.3 with chunk size *x*; VBS refers to a *virtual best solver*[4] — the composition of the VBS will be described when appropriate.

[2] http://logos.ucd.ie/wiki/doku.php?id=muser.

[3] http://www.satcompetition.org/.

[4] VBS represents a solver obtained from running a number of algorithms in parallel. It can be seen as a naive portfolio solver — we note that portfolio solvers are the current state of the art in SAT solving.

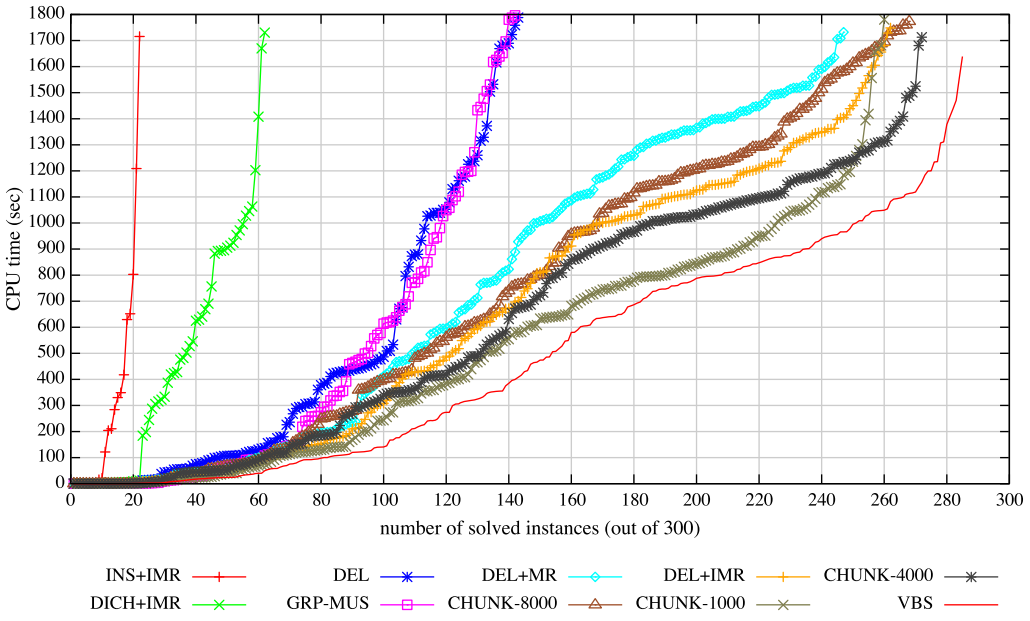

Fig. 2. Cactus plot with the run times of all algorithms.

7.1. Evaluation of algorithms' performance, and their performance profile

The cactus plot[5] in Fig. 2 provides an overview of the performance of the algorithms discussed in this paper.

A number of conclusions can be drawn from the plot in Fig. 2. First, we note that the improvements to the plain deletion-based algorithm (DEL) suggested in Section 3.1, namely the addition of model rotation (DEL+MR) and the improved model rotation (DEL+IMR), have a very significant positive effect on the performance of the algorithm; also observe that the improved model rotation provides a notable boost over model rotation. Further, it is clear that the insertion-based and the dichotomic algorithms, even with the addition of IMR, do not scale. As a side note, the gap between the performance of these and the deletion-based algorithm in the MES setting is *significantly* larger than that in the MUS setting (see for example [23]) — this is due to the addition of the negation of the working formula, which is unavoidable in the MES setting. We also observe the weak performance of the GRP-MUS approach — this is not surprising, since for large formulas, GRP-MUS can produce hard instances of SAT; this deficiency, in fact, was the motivation for the chunked approach. While DEL+IMR is among the best performing algorithms, on most of the instances it is significantly outperformed by the chunked group-oriented MUS algorithm with chunk size 1000. However, CHUNK-1000 loses to DEL+IMR on some of the hard instances — increasing the chunk size to 4000 pushes the performance of the algorithm ahead, however a further increase of chunk size to 8000 begins to affect the performance negatively. The VBS in Fig. 2 is constructed from DEL+IMR, GRP-MUS, CHUNK-1000 and CHUNK-4000. The former two are taken because they represent the extremes of the chunked approach — chunks of size 1 for DEL+IMR and a single chunk of the size of the input formula for GRP-MUS. The fact that the results for the VBS configuration (285 solved instances) are clearly superior to any of the individual algorithms indicates that the proposed algorithms are highly complementary, and are suitable for a multi-core/portfolio implementation. We stress that the plain deletion-based DEL is, to our knowledge, the current published state-of-the-art in MES computation, and so the algorithms proposed in this paper constitute a significant advancement, with the best algorithms solving more than twice the number of instances within the timeout.

The scatter plots in Fig. 3 provide additional insights into the performance of some of the algorithms. The color code indicates the amount of redundant clauses in the formulas: yellow (lighter) indicates close to 100% redundant clauses, whereas blue (darker) indicates less than 20% redundant clauses.

The top-left plot (DEL vs. DEL+IMR) demonstrates the impact of improved model rotation (IMR) on the performance of the deletion-based approach. We note that while IMR allows to solve significantly more instances (263 vs. 144 — see Fig. 2), the technique has little impact on many highly redundant instances (yellow). This behavior is expected as IMR can only help to detect irredundant clauses, and it was the motivation for the development of the group-MUS approach. The top-right plot of Fig. 3 (GRP-MUS vs. DEL+IMR) demonstrates the effectiveness of the group-MUS approach on the highly redundant instances. The plot clearly shows that the direct and the group-MUS approaches are complementary — the former excels on mostly irredundant instances, while the latter is best on highly redundant ones. This observation provides additional, em-

[5] Cactus plots show the sorted run times of algorithms over all instances and are commonly used in SAT competitions to compare performance of multiple solvers.

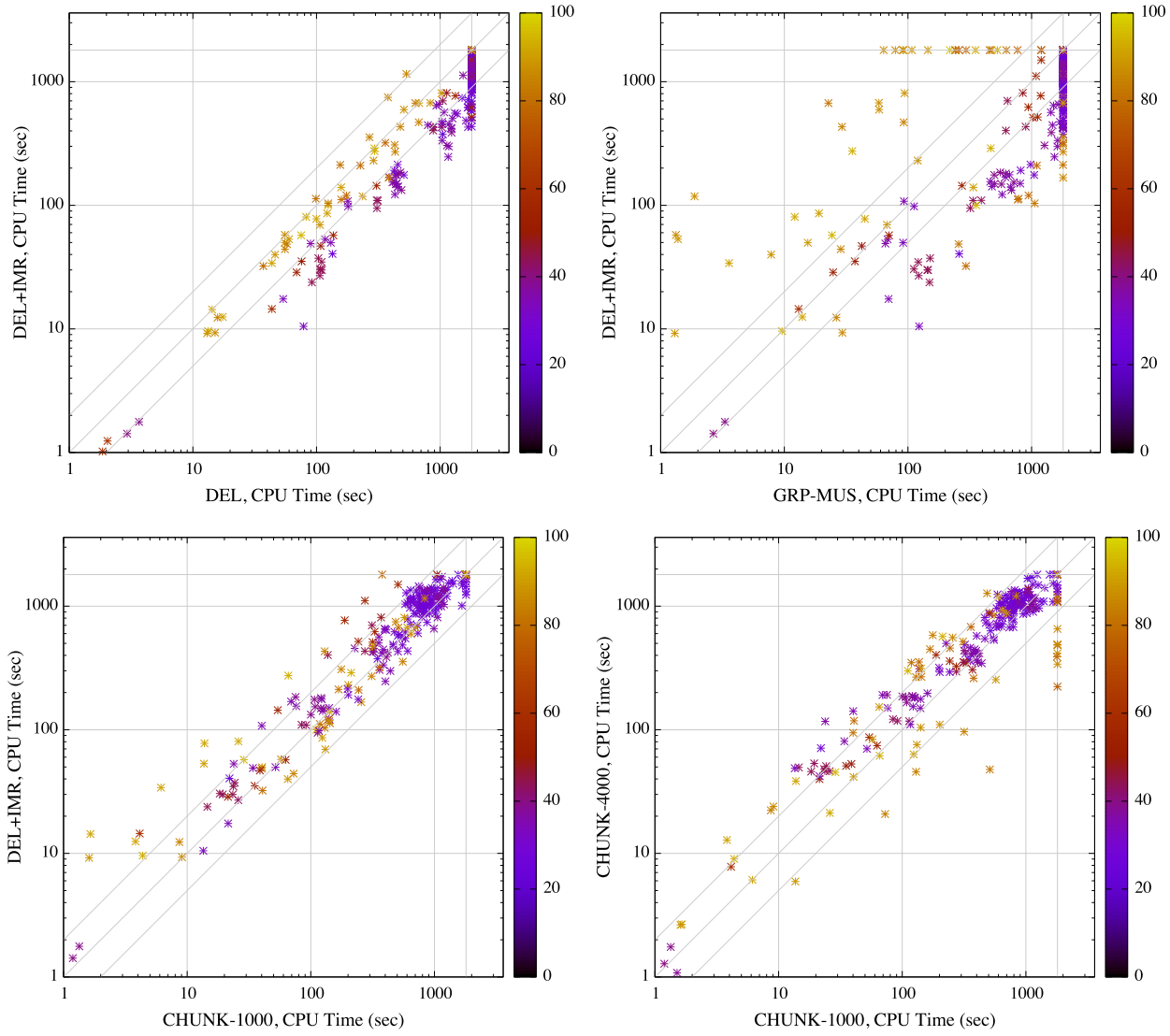

Fig. 3. Selected scatter plots (timeout of 1800 seconds is indicated by the extra grid lines); color range represents % of redundant clauses in an instance: the lighter the point, the more redundant the instance.

pirical, justification for the development of the chunked group-oriented MUS algorithm. The bottom-left plot (CHUNK-1000 vs. DEL+IMR) confirms the effectiveness of the chunked approach — observe the significant performance improvements on instances with diverse degrees of redundancy. Finally, the plot on the bottom-right (CHUNK-1000 vs. CHUNK-4000) demonstrates that even though the increase of chunk size from 1000 to 4000 affects the performance negatively on most instances, it is essential for handling difficult, highly redundant, problems.

7.2. Evaluation of backbone-based pruning

The backbone-based pruning algorithm (see Section 4) was implemented as a stand-alone preprocessor on top of the SAT solver `Minisat 2.2` [40]. To evaluate the effectiveness of pruning, we ran the preprocessor on the 300 selected instances (with 1800 seconds CPU time limit, and 4 GB memory limit), and re-run the experiments described in Section 7.1 on the pruned instances, treating any instance for which the total CPU time of preprocessing and MES extraction exceeded 1800 seconds as a time-out.

The left-hand side plot on the top of Fig. 4 shows a cumulative histogram of a percentage of redundant clauses removed during the preprocessing. The plot demonstrates that despite the fact that the backbone-based pruning technique is incomplete and highly dependent on the quality of the unsatisfiable cores returned by the SAT solver, on the selected instances it is very effective: only on 20 instances out of 300, the pruning removed less than 50% of redundant clauses. Furthermore, on close to 250 instances, the technique removed 70% or more of redundant clauses, and, in some cases, elim-

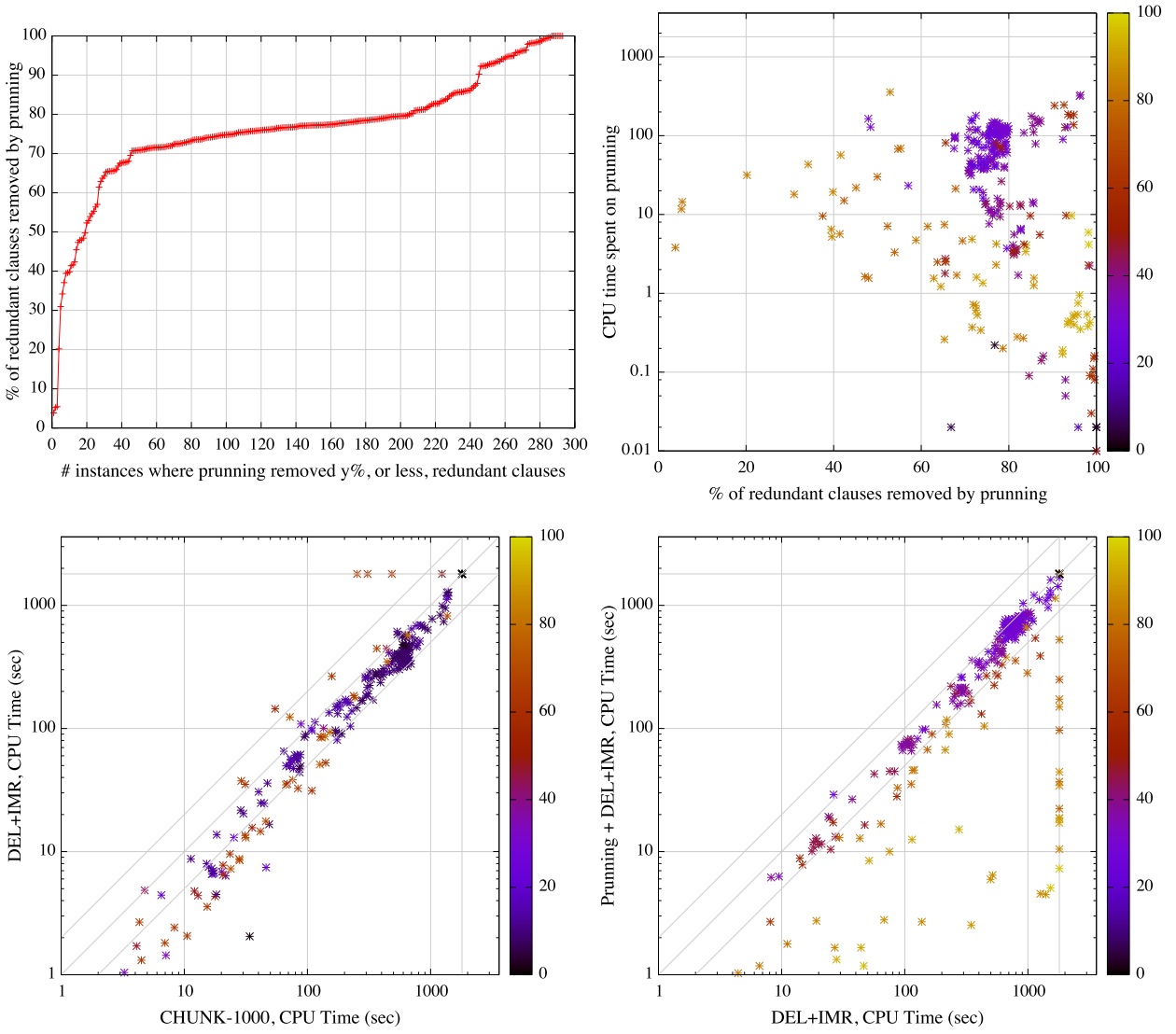

Fig. 4. Evaluation of the effectiveness of backbone-based pruning. Top-left: cumulative histogram of the % of redundant clauses removed by pruning; note that the histogram is rotated 90° to be consistent with the cactus plot in Fig. 2. Top-right: performance of pruning in terms of CPU time. Bottom-left: performance of DEL+IMR vs. CHUNK-1000 on *pruned* instances. Bottom-right: CPU time of DEL+IMR with and without pruning. Color range represents % of redundant clauses in an instance (in bottom-right plot the percentage is for pruned instances). Timeout of 1800 seconds is indicated by the extra grid lines.

inated the redundant clauses entirely. The plot on the top-right-hand side of Fig. 4 shows that the preprocessing is not only effective, but also very fast: all non-timed-out instances were preprocessed in under 400 seconds, with the vast majority under 100 seconds. In fact, among the 300 instances, the preprocessor timed out on a single instance only. In addition, the pruning technique appears to be very robust — this is witnessed by the fact that even on the instances with a low degree of redundancy (points with blue/dark color), the preprocessing often succeeds to remove over 60% of redundant clauses.

Recall from Section 3.4 that the chunking MES computation algorithm is designed to strike a balance between the deletion-based approach, that performs well on instances with few redundant clauses, and the group-MUS based algorithm, geared towards the instances with high degree of redundancy. Given that on the instances selected for our experiments the backbone-based pruning succeeds to remove a large amount of redundant clauses, it comes as no surprise that the relative effectiveness of the chunking approach, compared to the deletion-based approach, diminishes — this is demonstrated by the scatter plot on the bottom-left of Fig. 4, as well as by the data in Table 1. Note that the degree of redundancy of the instances (represented by the color of points) in this plot is the degree of redundancy in the *pruned* instances. Even though the CHUNK-1000 algorithm succeeds to solve 4 more instances within the timeout, on most of the (pruned) instances DEL+IMR is significantly faster. The bottom-right plot of Fig. 4 demonstrates the impact of the backbone-based pruning on the overall runtime of MES extraction. The results clearly show that, at least on the instances selected for

Table 1
The number of solved instances (out of 300), and the descriptive statistics of the CPU runtime (sec) of MES algorithms on instances preprocessed with backbone-based pruning. The average is taken over the solved instances only, while the median is over all instances.

	DEL	GRP-MUS	DEL+IMR	CHUNK-1000	CHUNK-4000	CHUNK-8000
Num. solved	282	95	289	**293**	175	136
Med. CPU time	459.88	1800	183.35	276.68	919.30	1800
Avg. CPU time	538.77	180.38	242.76	340.37	359.35	409.64

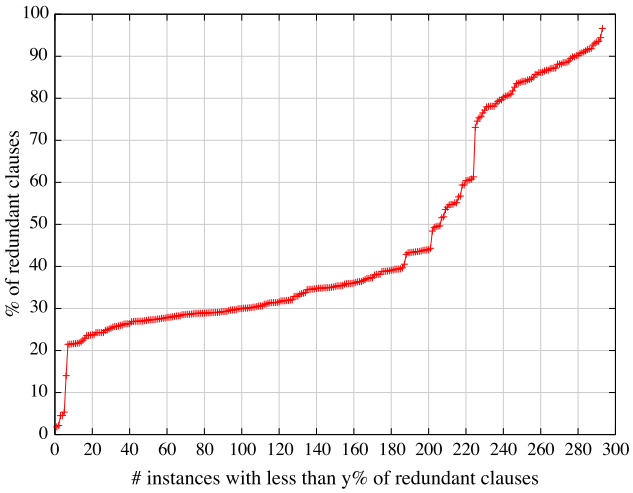

Fig. 5. Cumulative histogram of the % of redundant clauses in the set of instances; note that the histogram is rotated 90° to be consistent with the cactus plot in Fig. 2.

our experiments, the technique is very effective. This is particularly the case on many instances with a high degree of redundancy (yellow/light points). Although the technique might not be as effective on other classes of problems, given the fact that it is relatively inexpensive (see top-right plot of Fig. 4), it should be included in any MES computation flow.

7.3. Degree of redundancy in the selected benchmarks

The scatter plot in Fig. 5 presents the cumulative histogram of the degree of redundancy in the problem instances used in our experiments.

Note that the instances were selected from the SAT competition benchmark sets *prior* to the experiments, and so the selection was not biased by the degree of redundancy. Nevertheless, approximately 2/3 of the instances have between 20% and 50% redundant clauses, the remaining instances have over 50% redundant clauses, and close to 5% of the instances have in excess of 90% redundant clauses. We conclude that problem instances from practical applications may exhibit very significant levels of redundancy.

8. Conclusions

This paper proposes novel algorithms for computing MESes. The main contributions of the paper can be summarized as follows: (i) Adapting existing MUS extraction algorithms for MES extraction; (ii) Development of model rotation for MES extraction, and analysis of why clause set refinement cannot be applied; (iii) Reduction of MES to group-MUS extraction, which enables both model rotation and clause set refinement to be applied; (iv) Use of chunks for incremental reduction of MES to group-MUS extraction, aiming at reducing the hardness of instances of SAT when the formula includes the negation of a CNF formula; (v) Extension of the techniques to the group-MES computation problem; (vi) Development of a very effective technique for pruning redundant clauses using backbone computation; and (vii) Development of a solution for the independent certification of the correctness of computed MESes. The experimental results indicate that the algorithms proposed in this paper improve the direct approach [3] significantly, more than doubling the number of instances that can be solved, and with significant performance gains, which can exceed two orders of magnitude when combined with backbone-based pruning.

The experimental evaluation carried out in the paper demonstrates that the developed algorithms are relevant for the practical applications of SAT. Indeed, our results show that many real-world SAT instances have significant percentages of redundant clauses. These findings clearly motivate revisiting the CNF encoding techniques used for generating these instances.

The extension of the MES algorithms to the group-MES computation problem proposed in this paper could enable the identification and the removal of redundant constraints in other domains, such as CSP [16–18], SMT [19], and Ontologies [20]. The proposed algorithms could be applied in the context of Quantified Boolean Formulas (QBF) [13] as a *preprocessing technique* (propositional simplifications have been successfully applied to QBF in the past [14,15]). Another interesting direction for future work is the development of efficient algorithms for the computation of *Maximal Non-equivalent Subformulas* (MNSes) [35], and *Minimal Distinguishing Subformulas* (MDSes), which are the generalizations of the well-known concepts of Maximal Satisfiable Subformulas (MSSes) and Minimal Correction Subset (MCSes). An extension of the recently proposed effective algorithms for the computation of Minimal Correction Subsets (cf. [72]) using the idea of a witness of equivalence is a possible approach to development of such algorithms.

The backbone-based technique enables removing certain redundant clauses that contain a backbone literal. This technique could be extended to removing redundant clauses that contain a certain implicate of the formula, i.e. a clause implied by the formula (note that a backbone literal is an implicate of size 1). However, implicate computation is in general very expensive [1] and therefore some discriminating heuristics would need to be devised. Finally, the development of clause-selection and clause-partitioning heuristics for the MES computation problem is an additional future research direction from our work.

References

[1] A. Darwiche, P. Marquis, A knowledge compilation map, J. Artif. Intell. Res. 17 (2002) 229–264.
[2] P. Liberatore, Redundancy in logic I: CNF propositional formulae, Artif. Intell. 163 (2) (2005) 203–232.
[3] Y. Boufkhad, O. Roussel, Redundancy in random SAT formulas, in: AAAI, 2000, pp. 273–278.
[4] É. Grégoire, B. Mazure, C. Piette, On approaches to explaining infeasibility of sets of Boolean clauses, in: ICTAI, 2008, pp. 74–83.
[5] C. Desrosiers, P. Galinier, A. Hertz, S. Paroz, Using heuristics to find minimal unsatisfiable subformulas in satisfiability problems, J. Comb. Optim. 18 (2) (2009) 124–150.
[6] J. Marques-Silva, Minimal unsatisfiability: models, algorithms and applications, in: ISMVL, 2010, pp. 9–14.
[7] S.D. Prestwich, Cnf encodings, in: Biere et al. [74], pp. 75–97.
[8] K. Pipatsrisawat, A. Darwiche, On the power of clause-learning SAT solvers as resolution engines, Artif. Intell. 175 (2) (2011) 512–525.
[9] A. Atserias, J.K. Fichte, M. Thurley, Clause-learning algorithms with many restarts and bounded-width resolution, J. Artif. Intell. Res. 40 (2011) 353–373.
[10] S.T. To, T.C. Son, E. Pontelli, Conjunctive representations in contingent planning: prime implicates versus minimal CNF formula, in: W. Burgard, D. Roth (Eds.), AAAI, AAAI Press, 2011, pp. 1023–1028.
[11] S.T. To, T.C. Son, E. Pontelli, On the use of prime implicates in conformant planning, in: M. Fox, D. Poole (Eds.), AAAI Conference on Artificial Intelligence, AAAI Press, 2010, pp. 1205–1210.
[12] M. Niepert, D.V. Gucht, M. Gyssens, Logical and algorithmic properties of stable conditional independence, Int. J. Approx. Reason. 51 (5) (2010) 531–543.
[13] H. Kleine Büning, U. Bubeck, Theory of quantified Boolean formulas, in: Biere et al. [74], pp. 735–760.
[14] F. Lonsing, A. Biere, Failed literal detection for QBF, in: K.A. Sakallah, L. Simon (Eds.), SAT, vol. 6695, Springer, 2011, pp. 259–272.
[15] A. Biere, F. Lonsing, M. Seidl, Blocked clause elimination for QBF, in: N. Bjørner, V. Sofronie-Stokkermans (Eds.), CADE, Springer, 2011, pp. 101–115.
[16] A. Dechter, R. Dechter, Removing redundancies in constraint networks, in: K.D. Forbus, H.E. Shrobe (Eds.), AAAI, 1987, pp. 105–109.
[17] C.W. Choi, J.H.-M. Lee, P.J. Stuckey, Removing propagation redundant constraints in redundant modeling, ACM Trans. Comput. Log. 8 (4) (2007).
[18] A. Chmeiss, V. Krawczyk, L. Sais, Redundancy in CSPs, in: ECAI, 2008, pp. 907–908.
[19] C. Scholl, S. Disch, F. Pigorsch, S. Kupferschmid, Computing optimized representations for non-convex polyhedra by detection and removal of redundant linear constraints, in: S. Kowalewski, A. Philippou (Eds.), TACAS, 2009, pp. 383–397.
[20] S. Grimm, J. Wissmann, Elimination of redundancy in ontologies, in: ESWC, 2011, pp. 260–274.
[21] N. Dershowitz, Z. Hanna, A. Nadel, A scalable algorithm for minimal unsatisfiable core extraction, in: SAT, 2006, pp. 36–41.
[22] A. Nadel, Boosting minimal unsatisfiable core extraction, in: FMCAD, 2010, pp. 121–128.
[23] J. Marques-Silva, I. Lynce, On improving MUS extraction algorithms, in: SAT, 2011, pp. 159–173.
[24] A. Belov, J. Marques-Silva, Accelerating MUS extraction with recursive model rotation, in: FMCAD, 2011, pp. 37–40.
[25] M.H. Liffiton, K.A. Sakallah, Algorithms for computing minimal unsatisfiable subsets of constraints, J. Autom. Reason. 40 (1) (2008) 1–33.
[26] A. Belov, M. Janota, I. Lynce, J. Marques-Silva, On computing minimal equivalent subformulas, in: Milano [75], pp. 158–174.
[27] D.A. Plaisted, S. Greenbaum, A structure-preserving clause form translation, J. Symb. Comput. 2 (3) (1986) 293–304.
[28] A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press, 2009.
[29] J.W. Chinneck, E.W. Dravnieks, Locating minimal infeasible constraint sets in linear programs, INFORMS J. Comput. 3 (2) (1991) 157–168.
[30] R.R. Bakker, F. Dikker, F. Tempelman, P.M. Wognum, Diagnosing and solving over-determined constraint satisfaction problems, in: IJCAI, 1993, pp. 276–281.
[31] J.L. de Siqueira N., J.-F. Puget, Explanation-based generalisation of failures, in: ECAI, 1988, pp. 339–344.
[32] F. Hemery, C. Lecoutre, L. Sais, F. Boussemart, Extracting MUCs from constraint networks, in: ECAI, 2006, pp. 113–117.
[33] U. Junker, QuickXplain: preferred explanations and relaxations for over-constrained problems, in: AAAI, 2004, pp. 167–172.
[34] J. Marques-Silva, M. Janota, A. Belov, Minimal sets over monotone predicates in boolean formulae, in: N. Sharygina, H. Veith (Eds.), CAV, Springer, 2013, pp. 592–607.
[35] O. Kullmann, Constraint satisfaction problems in clausal form II: minimal unsatisfiability and conflict structure, Fundam. Inform. 109 (1) (2011) 83–119.
[36] L. Zhang, S. Malik, Validating SAT solvers using an independent resolution-based checker: practical implementations and other applications, in: DATE [73], pp. 10880–10885.
[37] E.I. Goldberg, Y. Novikov, Verification of proofs of unsatisfiability for CNF formulas, in: DATE [73], pp. 10886–10891.
[38] A. Belov, I. Lynce, J. Marques-Silva, Towards efficient MUS extraction, AI Commun. 25 (2) (2012) 97–116.
[39] A. Biere, Picosat essentials, J. Satisf. Boolean Model. Comput. 4 (2–4) (2008) 75–97.
[40] N. Eén, N. Sörensson, An extensible SAT-solver, in: E. Giunchiglia, A. Tacchella (Eds.), SAT, Springer, 2003, pp. 502–518.
[41] N. Eén, N. Sörensson, Temporal induction by incremental SAT solving, Electron. Notes Theor. Comput. Sci. 89 (4) (2003) 543–560.
[42] S.M. González, P. Meseguer, Boosting MUS extraction, in: I. Miguel, W. Ruml (Eds.), SARA, 2007, pp. 285–299.
[43] É. Grégoire, B. Mazure, C. Piette MUST, Provide a finer-grained explanation of unsatisfiability, in: C. Bessiere (Ed.), CP, 2007, pp. 317–331.
[44] G. Ausiello, A. D'Atri, D. Saccà, Minimal representation of directed hypergraphs, SIAM J. Comput. 15 (2) (1986) 418–431.
[45] P.L. Hammer, A. Kogan, Optimal compression of propositional horn knowledge bases: complexity and approximation, Artif. Intell. 64 (1) (1993) 131–145.

[46] P. Liberatore, Redundancy in logic II: 2CNF and Horn propositional formulae, Artif. Intell. 172 (2–3) (2008) 265–299.
[47] P. Liberatore, Redundancy in logic III: non-monotonic reasoning, Artif. Intell. 172 (11) (2008) 1317–1359.
[48] O. Fourdrinoy, É. Grégoire, B. Mazure, L. Saïs, Eliminating redundant clauses in SAT instances, in: CPAIOR, 2007, pp. 71–83.
[49] C. Piette, Let the solver deal with redundancy, in: ICTAI, 2008, pp. 67–73.
[50] M. Järvisalo, M.J.H. Heule, A. Biere, Inprocessing rules, in: Proceedings of the 6th International Joint Conference on Automated Reasoning, IJCAR'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 355–370.
[51] G.D. Hachtel, F. Somenzi, Logic Synthesis and Verification Algorithms, Springer, 2006.
[52] R. Brayton, G. Hachtel, C. McMullen, A. Sangiovanni-Vincentelli, Logic Minimization Algorithms for VLSI Minimization, 1985.
[53] W.V. Quine, The problem of simplifying truth functions, Am. Math. Mon. (1952) 521–531.
[54] W.V. Quine, A way to simplify truth functions, Am. Math. Mon. (1955) 627–631.
[55] E.J. McCluskey, Minimization of Boolean functions, Bell Syst. Tech. J. 35 (6) (1956) 1417–1444.
[56] O. Coudert, Two-level logic minimization: an overview, Integration 17 (2) (1994) 97–140.
[57] D. Buchfuhrer, C. Umans, The complexity of Boolean formula minimization, J. Comput. Syst. Sci. 77 (1) (2011) 142–153.
[58] C. Umans, T. Villa, A.L. Sangiovanni-Vincentelli, Complexity of two-level logic minimization, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 25 (7) (2006) 1230–1246.
[59] C.-W.J. Chang, M. Marek-Sadowska, Theory of wire addition and removal in combinational boolean networks, Microelectron. Eng. 84 (2) (2007) 229–243.
[60] Y.-C. Chen, C.-Y. Wang, Logic restructuring using node addition and removal, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 31 (2) (2012) 260–270.
[61] A. Kaiser, W. Küchlin, Detecting inadmissible and necessary variables in large propositional formulae, in: Intl. Joint Conf. on Automated Reasoning (Short Papers), University of Siena, 2001, pp. 96–102.
[62] M. Janota, Do SAT solvers make good configurators?, in: Workshop on Analyses of Software Product Lines (ASPL), 2008, pp. 191–195.
[63] M. Janota, SAT solving in interactive configuration, Ph.D. thesis, University College Dublin, Nov. 2010.
[64] M. Codish, Y. Fekete, A. Metodi, Backbones for equality, in: V. Bertacco, A. Legay (Eds.), Haifa Verification Conference, Springer, 2013, pp. 1–14.
[65] H. van Maaren, S. Wieringa, Finding guaranteed MUSes fast, in: SAT, 2008, pp. 291–304.
[66] S. Wieringa, Understanding, improving and parallelizing MUS finding using model rotation, in: Milano [75], pp. 672–687.
[67] C. Piette, Y. Hamadi, L. Saïs, Efficient combination of decision procedures for MUS computation, in: FroCos, 2009, pp. 335–349.
[68] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, L. Troyansk, Determining computational complexity from characteristic 'phase transitions', Nature 400 (1999) 133–137.
[69] B. Bollobás, C. Borgs, J.T. Chayes, J.H. Kim, D.B. Wilson, The scaling window of the 2-SAT transition, Random Struct. Algorithms 18 (3) (2001) 201–256.
[70] P. Kilby, J.K. Slaney, S. Thiébaux, T. Walsh, Backbones and backdoors in satisfiability, in: AAAI Conference on Artificial Intelligence, 2005, pp. 1368–1373.
[71] M. Janota, I. Lynce, J. Marques-Silva, Algorithms for computing backbones of propositional formulae, AI-Com RCRA 2012 special issue, available at http://sat.inesc-id.pt/~mikolas/.
[72] J. Marques-Silva, F. Heras, M. Janota, A. Previti, A. Belov, On computing minimal correction subsets, in: F. Rossi (Ed.), IJCAI'13, AAAI Press, 2013, pp. 615–622.
[73] Design, Automation and Test in Europe Conference and Exposition (DATE 2003), vol. 2003, IEEE Computer Society, Munich, Germany, March 2003, pp. 3–7, 2003.
[74] A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press, 2009.
[75] M. Milano (Ed.), Principles and Practice of Constraint Programming — 18th International Conference, CP 2012, Proceedings, Québec City, QC, Canada, October 8–12, 2012, Springer, 2012.