The MaxSAT Problem in the Real-Valued MV-Algebra

Zuzana Haniková[1(✉)], Felip Manyà[2], and Amanda Vidal[2]

[1] Institute of Computer Science of the Czech Academy of Sciences, Prague, Czech Republic
zuzana@cs.cas.cz

[2] Artificial Intelligence Research Institute (IIIA, CSIC), Bellaterra, Spain
{felip,amanda}@iiia.csic.es

Abstract. This work addresses the maximum satisfiability (MaxSAT) problem for a multiset of arbitrary formulas of the language of propositional Łukasiewicz logic over the MV-algebra whose universe is the real interval [0,1]. First, we reduce the MaxSAT problem to the SAT problem over the same algebra. This solution method sets a benchmark for other approaches, allowing a classification of the MaxSAT problem in terms of metric reductions introduced by Krentel. We later define an alternative analytic method with preprocessing in terms of a Tseitin transformation of the input, followed by a reduction to a system of linear constraints, in analogy to the earlier approaches of Hähnle and Olivetti. We discuss various aspects of these approaches to solving the problem.

Keywords: Maximum satisfiability · Satisfiability · Łukasiewicz logic · MV-algebra

1 Introduction

Satisfiability is a semantic problem: it relates not just to a logic (here, the infinite-valued Łukasiewicz logic), but to a semantics interpreting that logic (here, the MV-algebra on the real unit interval with natural order, called "standard MV-algebra" and denoted $[0,1]_{Ł}$).

A propositional formula $\varphi(x_1,\ldots,x_n)$ of the language of Łukasiewicz logic is *satisfiable* in an MV-algebra $\mathcal{A}$ provided there is an assignment of elements of the universe of $\mathcal{A}$ to $x_1,\ldots,x_n$ that yields the value $1^{\mathcal{A}}$ (i.e., the top element in the lattice order of $\mathcal{A}$). This definition determines, for a given MV-algebra $\mathcal{A}$, a unique set of its satisfiable formulas $\mathbf{SAT}(\mathcal{A})$. The satisfiability notion extends immediately to a *finite list* of formulas $\langle\varphi_1,\ldots,\varphi_m\rangle$, which is satisfiable in $\mathcal{A}$ if and only if so is the conjunction of the formulas on the list.[1]

[1] It is important to specify which MV-algebra is considered, since for many infinite MV-algebras $\mathcal{A}$, and even many subalgebras of $[0,1]_{Ł}$, the set $\mathbf{SAT}(\mathcal{A})$ is distinct from $\mathbf{SAT}([0,1]_{Ł})$ [16, Theorem 6.6]. Some extant works on satisfiability refer to "infinite-valued Łukasiewicz logic" while in fact working with the algebra $[0,1]_{Ł}$.

R. Ramanayake and J. Urban (Eds.): TABLEAUX 2023, LNAI 14278, pp. 386–404, 2023.
https://doi.org/10.1007/978-3-031-43513-3_21

This paper works with the standard MV-algebra $[0,1]_{\text{Ł}}$ without mentioning it explicitly from now on; thus we write **SAT** for **SAT**$([0,1]_{\text{Ł}})$ and likewise for the MaxSAT problems considered in this paper. If another algebra, distinct from $[0,1]_{\text{Ł}}$, is considered, it will be indicated explicitly.

The focus of this paper is not on satisfiability, but on maximum satisfiability, an optimization problem (with a natural decision version): given a multiset (i.e., a list) of arbitrary formulas of the language of Łukasiewicz logic, find the *maximum number* among them that can be satisfied under a single assignment, over all assignments. The formulas are not required to be in a normal form. It has been recognized early on by Mundici [22] that formulas of Łukasiewicz logic are a suitable device for *counting*; his paper gives a reduction of the (decision version of) the Boolean MaxSAT problem to the problem **SAT**; see also [25].

The MaxSAT problem for a list of arbitrary formulas over the three-element MV-chain has been addressed in [19], using semantic tableaux; the approach generalizes to other finite MV-chains, but not to MV-chains with infinitely many elements. Earlier results in satisfiability go back to Mundici's proof of the **NP**-completeness of the **SAT** problem, obtained by bounding the denominators of a satisfying assignment. This line of research was continued in [1,2], see also [27].

Our main contribution consists in showing that the MaxSAT problem can be reduced to the **SAT** problem, in Sect. 3, and can then be used as a benchmark to assess the analytic method in Sect. 4; a similar analysis could then be performed with any other calculi for the maximum satisfiability problem.

This paper is structured as follows. Section 2 defines the problem and introduces technical tools. Section 3 gives a method for solving the MaxSAT problem in $[0,1]_{\text{Ł}}$ based on a Cook reduction of MaxSAT to the **SAT** problem. Section 4 outlines an analytic method with preprocessing via a Tseitin transformation, using a variant of the approach of [12,24], where each branch of a tableau tree ends with solving a system of linear constraints. The method is proved sound and complete. Eliminating the branching of the tree can also be achieved, using established tools.

2 Problem Formulation and Preliminaries

The language of propositional Łukasiewicz logic Ł, denoted $\mathcal{L}(\text{Ł})$, has two basic connectives: $\neg$ (negation, unary) and $\oplus$ (strong disjunction, binary). Other connectives are definable: 1 is $x \oplus \neg x$; 0 is $\neg 1$; $x \odot y$ is $\neg(\neg x \oplus \neg y)$ (strong conjunction); $x \to y$ is $\neg x \oplus y$; $x \leftrightarrow y$ is $(x \to y) \odot (y \to x)$; $x \vee y$ is $(x \to y) \to y$ (weak disjunction); and $x \wedge y$ is $\neg(\neg x \vee \neg y)$ (weak conjunction).

Well-formed formulas of $\mathcal{L}(\text{Ł})$ are built up from an infinite set of propositional variables $\text{Var} = \{x_i\}_{i \in \mathbb{N}}$ using the connectives of $\mathcal{L}(\text{Ł})$. The basic language is a point of reference for complexity considerations; other connectives are used as shortcuts. If φ is a formula of $\mathcal{L}(\text{Ł})$ in the basic language, $|\varphi|$ denotes the *number of occurrences of* propositional variables in φ. Given that $\neg\neg\alpha \leftrightarrow \alpha$ is a theorem of Ł for any formula $\alpha \in \mathcal{L}(\text{Ł})$, we will assume double negation does not occur in formulas. With this convention in place, the number of occurrences of connectives in φ is bounded by $2|\varphi|$. Thus $|\varphi|$ is a good notion of *length* of φ. Moreover $||\varphi||$ denotes the number of *distinct* subformulas of φ.

MV-algebras can be introduced using Mundici's Γ-functor [10,20]: any MV-algebra is isomorphic to $\Gamma(\mathcal{G}, u)$ for a lattice-ordered Abelian group $\mathcal{G}$ with a strong unit u (in particular, define $x \oplus y = u \wedge (x + y)$ and $\neg x = u - x$ for $x, y \in G$; then $\Gamma(\mathcal{G}, u) = \langle [0, u], \oplus, \neg \rangle$ is an MV-algebra). The standard MV-algebra $[0,1]_{\text{Ł}}$ is $\Gamma(\mathbb{R}, 1)$, interpreting the basic connectives in $[0,1]$ as follows: for any assignment v, $v(\neg\varphi) = 1 - v(\varphi)$ and $v(\varphi \oplus \psi) = \min(1, v(\varphi) + v(\psi))$. Any assignment to variables of φ in language $\mathcal{L}(\text{Ł})$ extends to all its subformulas in the interpretation provided by $[0,1]_{\text{Ł}}$; this also determines the notion of satisfiability in $[0,1]_{\text{Ł}}$ and the set of satisfiable formulas of $[0,1]_{\text{Ł}}$, denoted **SAT**.

The interpretations of $\oplus$, $\odot$, $\wedge$ and $\vee$ are commutative and associative, so one can write $x_1 \oplus \cdots \oplus x_n$ without worrying about order and parentheses. We write x^n for $\underbrace{x \odot \cdots \odot x}_{n \text{ occurrences}}$ and nx for $\underbrace{x \oplus \cdots \oplus x}_{n \text{ occurrences}}$. Also, $\vee$ and $\wedge$ distribute over each other and $\odot$ distributes over $\vee$.

Unlike the Boolean MaxSAT problem over the two-element Boolean algebra, here we work with *arbitrary* formulas of $\mathcal{L}(\text{Ł})$. We formulate both the optimization and the decision version of the MaxSAT problem.

MaxSAT-OPT
Instance: multiset $\langle \varphi_1, \ldots, \varphi_m \rangle$ of formulas of $\mathcal{L}(\text{Ł})$ in variables $\{x_1, \ldots, x_n\}$.
Output: the maximum integer $k \leq m$ such that there is an assignment v to $\{x_1, \ldots, x_n\}$ that satisfies at least k formulas in the multiset $\langle \varphi_1, \ldots, \varphi_m \rangle$.

MaxSAT-DEC
Instance: multiset $\langle \varphi_1, \ldots, \varphi_m \rangle$ of formulas of $\mathcal{L}(\text{Ł})$ in variables $\{x_1, \ldots, x_n\}$ and a positive integer $k \leq m$.
Output: (Boolean) Is **MaxSAT-OPT**$(\langle \varphi_1, \ldots, \varphi_m \rangle(x_1, \ldots, x_n))$ at least k?

Let $\mathbf{A}$ be an integer $m \times n$ matrix. Let $\mathbf{x}$ be an n-vector of variables and $\mathbf{b}$ be an integer m-vector. The **solvability of the system of inequalities** $\mathbf{Ax} \leq \mathbf{b}$ in $\mathbb{R}$ can be tested in polynomial time [28].

More generally, for the system $\mathbf{Ax} \leq \mathbf{b}$, one can ask about the maximal size (number of lines) of a subsystem that is solvable in $\mathbb{R}$. This problem is known as the *maximum feasible subsystem* [4] of a system of linear constraints: the solution is a natural number k bounded by m (the total number of lines in the system). This problem is **NP**-hard. We shall refer to this problem as **Max-FS problem**. Notice that the system is not defined as a set, so the same constraint may appear multiple times.

There are many variants of the Max-FS problem, indeed many were already suggested in the paper [4]. We will use a variant that partitions the linear constraints into two groups: those that need to be satisfied by any feasible solution (often called *hard constraints*; the paper [4] refers to them as "mandatory") and those the satisfied number of which is to be maximized (often called *soft constraints*; [4] refers to them as "optional") over all feasible solutions. This variant of Max-FS problem will be called **Max-FS with hard and soft constraints** within this paper.

3 Canonical Method

First we give a polynomial-time, many-one (a.k.a. Karp) reduction of **MaxSAT-DEC** to **SAT**. Our reduction is similar to those used in [25] (which, in turn, refers to [22]) and in [15]. The differences arise from the fact that, in our case, an unsatisfied formula can take any value below 1 (but not necessarily 0), and this needs to be addressed in the definition of the set of formulas in the reduction.

Let $\langle \varphi_1, \ldots, \varphi_m \rangle(x_1, \ldots, x_n)$ and $k \leq m$ be an instance of **MaxSAT-DEC**. It is well known that one can implicitly define any rational value in $[0,1]_{\text{Ł}}$ with a formula of $\mathcal{L}(\text{Ł})$: an early example of suitable formulas can be found in [30]. Let $k \geq 2$ and y be a new variable, not among $(x_1, \ldots, x_n)$, and let

$$\rho_{1/k} \coloneqq \quad y \leftrightarrow \neg((k-1)y)$$

Then we have that $\rho_{1/k}$ implicitly defines the rational value $1/k$ in $[0,1]_{\text{Ł}}$ (see, e.g., [25, Lemma 2]): that is, an assignment v in $[0,1]_{\text{Ł}}$ sends $\rho_{1/k}$ to 1 if and only if it sends y to $1/k$. Moreover, the length of this formula is linear in $k \leq m$, therefore linear in the size of the instance on input.

For $1 \leq i \leq m$, consider a new variable $y_{i,k}$, let $\Phi_{\varphi_i,k}$ be the set of formulas

$$\{\ (\varphi_i \leftrightarrow k\, y_{i,k}) \vee \neg y_{i,k} \quad , \quad (y_{i,k} \leftrightarrow y) \vee \neg y_{i,k}\ \}$$

and let Φ_k be the list of formulas $\bigcup_{1 \leq i \leq m} \{\Phi_{\varphi_i,k}\}$.

Theorem 1. *The pair $\langle \varphi_1, \ldots, \varphi_m \rangle(x_1, \ldots, x_n)$ and k with $2 \leq k \leq m$ belongs to* **MaxSAT-DEC** *if and only if the set $\{\rho_{1/k}\} \cup \Phi_k \cup \{\bigoplus_{i=1}^m y_{i,k}\}$ belongs to* **SAT**.

Proof. For the left-to-right direction, assume v to be an assignment satisfying—without loss of generality—the first k formulas of the list. Consider then the assignment v' that coincides with v on the variables $x_1, \ldots, x_n$ and puts $v'(y) = 1/k$ and

$$v'(y_{i,k}) = \begin{cases} 1/k & \text{if } i \leq k \\ 0 & \text{otherwise.} \end{cases}$$

The assignment v' clearly satisfies $\rho_{1/k}$. Next, since $v'(y_{1,k}) = \ldots = v'(y_{k,k}) = 1/k$, also $v'(\bigoplus_{i=1}^m y_{i,k}) = 1$. Lastly, the formulas in Φ_k are satisfied under v': the formulas $(y_{i,k} \leftrightarrow y) \vee \neg y_{i,k}$ are trivially satisfied, since each $y_{i,k}$ is indeed sent to either $1/k$ (and hence, $v'(y)$) or to 0. For the other formulas in Φ_k, first $v'(\varphi_j) = 1$ and $kv'(y_{j,k}) = k1/k = 1$ for each $1 \leq j \leq k$, and $v'(\neg y_{j,k}) = 1$ for $k < j \leq m$, hence they are all satisfied.

For the right-to-left direction, let v be an assignment satisfying $\{\rho_{1/k}\} \cup \Phi_k \cup \{\bigoplus_{i=1}^m y_{i,k}\}$. From Φ_k and $\rho_{1/k}$ we know $v(y_{i,k})$ is either $1/k$ or 0. Therefore, for $v(\bigoplus_{i=1}^m y_{i,k}) = 1$, necessarily at least k many y-variables are evaluated to $1/k$. Assume, again without loss of generality, that $v(y_{1,k}) = \ldots = v(y_{k,k}) = 1/k$. From Φ_k, we get that $v((\varphi_i \leftrightarrow k\, y_{i,k}) \vee \neg y_{i,k}) = 1$ for each $1 \leq i \leq m$. In particular, since $v(\neg y_{j,k}) \neq 1$ for every $1 \leq j \leq k$, necessarily $v((\varphi_j \leftrightarrow k\, y_{j,k}))$ for each such j. Together with the previously observed fact that $y_{j,k} = 1/k$ for each such j, this implies that $v(\varphi_1) = \ldots = v(\varphi_k) = 1$, concluding the proof.

For $k = 1$, it is immediate that $\langle \varphi_1, \ldots, \varphi_m \rangle$ and k is in **MaxSAT-DEC** if and only if $(\ldots(\varphi_1 \vee \varphi_2) \vee \ldots) \vee \varphi_m$ is in **SAT**. Given that for $m = k = 1$ both problems coincide, we get:

Corollary 1. *The problem* **MaxSAT-DEC** *is* **NP***-complete.*

This reduction from **MaxSAT-DEC** to **SAT** provides a practical approach to the MaxSAT problem in $[0,1]_{\text{Ł}}$, provided that we use a competitive algorithm for solving **SAT** (i.e., the satisfiability problem in $[0,1]_{\text{Ł}}$). We could rely on either of the following two **SAT** solvers, which have been shown rather efficient. The first one is the tableau with constraints method proposed by Hähnle [12] that reduces **SAT** to Mixed Integer Programming (MIP) and can therefore use any available MIP solver. The second one is the Satisfiability Modulo Theory (SMT) methods proposed by Ansótegui et al. that reduces **SAT** to an SMT satisfiability problem and can use any available SMT solver [6,7,32]. These methods can take advantage of the latest developments and innovations in MIP and SMT solvers, avoiding the need to implement a **SAT** solver from scratch.

A polynomial-time Turing (a.k.a. Cook) reduction of **MaxSAT-OPT** to **MaxSAT-DEC** can be given, as we proceed to explain. It is this approach that prompts our referring to this method of solving **MaxSAT-OPT** as *canonical*, given its wide scope of applicability to optimization problems (see, e.g., [29]). The reduction uses an unspecified algorithm for **MaxSAT-DEC** as an *oracle*; as usual with oracle computations, any call to the oracle counts as one step in the computation and under this proviso, the oracle computation runs in time polynomial in the input size $(\Sigma_{i=1}^{m}|\varphi_i|)$. Indeed, given an instance $\langle \varphi_1, \ldots, \varphi_m \rangle$, it is easy to arrive at the optimal value for **MaxSAT-OPT** using binary search on the discrete, polynomial-size search space $\{1, \ldots, m\}$ of possible solutions, using at most $\lceil \log m \rceil$ oracle calls. Considering that **MaxSAT-DEC** is **NP**-complete by Corollary 1, we have the following:

Corollary 2. **MaxSAT-OPT** *is in* $\mathbf{FP}^{\mathbf{NP}}$.

For this conclusion, it is not important that the oracle solves **MaxSAT-DEC**; any oracle solving an **NP**-complete problem (an **NP**-oracle) would suit, and indeed one can use any algorithm for **SAT**, relying on Theorem 1. In view of the obvious reduction from **MaxSAT-DEC** to **MaxSAT-OPT**, the two problems are equivalent in the sense that if either has a polynomial-time algorithm, so does the other. This is standard, and it is why the decision version of an optimization problem is often considered *in lieu* of the problem as such.

Can one do better than $O(\log m)$ oracle calls? Below, we provide a classification of the problem in terms of Krentel's work [17] that suggests a negative answer subject to $\mathbf{P} \neq \mathbf{NP}$. Krentel ranks optimization problems in $\mathbf{FP}^{\mathbf{NP}}$ in terms of the number of calls to an **NP**-oracle. For $z : \mathbb{N} \longrightarrow \mathbb{N}$ a smooth function (i.e., z is non-decreasing and polynomial-time computable in unary representation), $\mathbf{FP}^{\mathbf{NP}}[z(n)]$ is the class of functions computable in polynomial time with an **NP** oracle with at most $z(|x|)$ oracle calls for instance x, where $|x|$ denotes the length of x. By definition, $\mathbf{FP}^{\mathbf{NP}}$ coincides with $\mathbf{FP}^{\mathbf{NP}}[n^{O(1)}]$ since a polynomial-time algorithm can make no more than a polynomial amount of oracle calls.

For Σ a finite alphabet let $f, g : \Sigma^* \longrightarrow \mathbb{N}$. A *metric reduction* [17] from f to g is a pair (h_1, h_2) of polynomial-time computable functions where $h_1 : \Sigma^* \longrightarrow \Sigma^*$ and $h_2 : \Sigma^* \times \mathbb{N} \longrightarrow \mathbb{N}$ such that $f(x) = h_2(x, g(h_1(x)))$ for all $x \in \Sigma^*$. The notion of a metric reduction is a natural generalization of polynomial-time many-one reduction to optimization problems. It follows from the definition that for each smooth function z as above, $\mathbf{FP}^{\mathbf{NP}}[z(n)]$ is closed under metric reductions.

Theorem 2. ([17], see also [29]) *Assume* $\mathbf{P} \neq \mathbf{NP}$.
Then $\mathbf{FP}^{\mathbf{NP}}[O(\log \log n)] \subsetneq \mathbf{FP}^{\mathbf{NP}}[O(\log n)] \subsetneq \mathbf{FP}^{\mathbf{NP}}[n^{O(1)}]$.

Recall that Boolean algebras form a subvariety of MV-algebras. In particular, in any Boolean algebra, the interpretations of the strong and the weak disjunction coincide, as do the interpretations of the strong conjunction and the weak conjunction. When mapping the Boolean connectives to the $\mathcal{L}(\text{Ł})$ connectives, we take $\neg$ for the Boolean negation, $\vee$ for the Boolean disjunction, and $\odot$ as the Boolean conjunction.

Moreover, in every nontrivial MV-algebra $\mathcal{A}$, the set consisting of its bottom element $0^{\mathcal{A}}$ and its top element $1^{\mathcal{A}}$ is closed under all operations of $\mathcal{A}$ and the subalgebra of $\mathcal{A}$ on the universe consisting of these two elements is isomorphic to the two-element Boolean algebra.

Now let us recall the MaxSAT problem in the two-element Boolean algebra for CNF formulas, given as multisets of clauses.

Classical-MaxSAT-OPT
Instance: multiset $\langle C_1, \ldots, C_m \rangle$ of Boolean clauses in variables $\{x_1, \ldots, x_n\}$.
Output: the maximum integer $k \leq m$ such that there is an assignment v in the two-element Boolean algebra on $\{0, 1\}$ to $\{x_1, \ldots, x_n\}$ that satisfies at least k clauses.

Krentel [17] proves the following result: **Classical-MaxSAT-OPT** is complete for $\mathbf{FP}^{\mathbf{NP}}[O(\log m)]$ under metric reductions.

We now prepare a few technical tools for eventually giving a metric reduction of **Classical-MaxSAT-OPT** to **MaxSAT-OPT**. Following [16, Def. 7.1], consider the language $\mathcal{L}(\text{Ł})$ including the definable connectives and define:

(i) a *literal* is a variable (such as x) or a negation thereof (such as $\neg x$).
(ii) A $(\odot, \vee)$*-formula* is built up from literals using arbitrary combination of $\odot$ and $\vee$.
(iii) In particular, a *clause* is built up from literals using only $\vee$.

Lemma 1. ([16, Thm. 7.4])

- *The interpretation of any* $(\odot, \vee)$*-formula with* n *variables in* $[0, 1]_{\text{Ł}}$ *is a convex function in* $[0, 1]^n$*;*
- *any* $(\odot, \vee)$*-formula (in particular, any clause) is satisfiable in* $[0, 1]_{\text{Ł}}$ *if and only if it is satisfiable in the two-element Boolean algebra* $\{0, 1\}$.

Lemma 2. *Let $C_1, \ldots, C_l$ be clauses in $\mathcal{L}(\text{Ł})$ in variables $\{x_1, \ldots, x_n\}$. Assume $\bar{a} \in [0,1]^n$ is such that $C_i(\bar{a}) = 1$ for each $1 \leq i \leq l$. Then there is an element $\bar{b} \in \{0,1\}^n$ such that $C_i(\bar{b}) = 1$ for $1 \leq i \leq l$.*

Proof. We construct $\bar{b}$ from $\bar{a}$ in n independent steps. Let $\bar{b}_1 := \bar{a}$. The j-th step takes a $\bar{b}_j$, assuming the property that $C_i(\bar{b}_j) = 1$ for each $1 \leq i \leq l$, and produces $\bar{b}_{j+1}$ with the same property, replacing the real value in the j-th coordinate of $\bar{b}_j$ with a Boolean value (i.e., either a 0 or a 1). Lastly, we set $\bar{b} := \bar{b}_{n+1}$: all coordinates of $\bar{b}$ are Boolean.

We describe the j-th step. We simplify notation by writing $\bar{b}'$ for $\bar{b}_j$. We thus have $\bar{b}' = \langle b'_1, \ldots, b'_n \rangle$. Consider the j-th component of this vector: if b'_j is 0 or 1, we set $\bar{b}_{j+1} := \bar{b}_j$, whereby the step is finished. If $0 < b'_j < 1$, define $\bar{b}'_0 := \langle b'_1, \ldots, b'_{j-1}, 0, b'_{j+1}, \ldots, b'_n \rangle$ and $\bar{b}'_1 := \langle b'_1, \ldots, b'_{j-1}, 1, b'_{j+1}, \ldots, b'_n \rangle$. By assumption, we have $C_1(\bar{b}') = 1$. From Lemma 1, the interpretation of C_1 is a convex function. Now assume that either $C_1(\bar{b}'_0) \neq 1$ or $C_1(\bar{b}'_1) \neq 1$. Then there is a convex combination of $C_1(\bar{b}'_0)$ and $C_1(\bar{b}'_1)$ that is strictly below $C_1(\bar{b}')$, a contradiction with the convexity fact. We conclude that $C_1(\bar{b}'_0) = C_1(\bar{b}'_1) = 1$. An analogous argument holds for the remaining clauses $C_2, \ldots, C_l$. This means that we can set either $\bar{b}_{j+1} := \bar{b}'_0$ or $\bar{b}_{j+1} := \bar{b}'_1$ and we will indeed have $C_i(\bar{b}_{j+1}) = 1$ for each $1 \leq i \leq l$.

Theorem 3. MaxSAT-OPT *is complete for* $\mathbf{FP}^{\mathbf{NP}}[O(\log m)]$ *under metric reductions.*

Proof. Containment was obtained in Corollary 2 and the discussion preceding it. We prove hardness. We claim that the metric reduction of **Classical-MaxSAT-OPT** to **MaxSAT-OPT** is provided by a pair of *identity functions*. Take an arbitrary instance of **Classical-MaxSAT-OPT** problem, namely a multiset $\langle C_1, \ldots, C_m \rangle$ of Boolean clauses in variables $\{x_1, \ldots, x_n\}$, and interpret it as a multiset of clauses in $\mathcal{L}(\text{Ł})$ (no change in notation is needed, see above). By Lemma 1, the interpretation of each C_i for $i = 1, \ldots, m$ in $[0,1]_{\text{Ł}}$ is a convex function. The convexity of the interpretation is not violated by rewriting each C_i in the basic connectives of $\mathcal{L}(\text{Ł})$; this yields formulas $\langle C_1^*, \ldots, C_m^* \rangle$. Feed this m-tuple to the algorithm solving **MaxSAT-OPT**. The output is a natural number $k \leq m$ which indicates the maximal number among $\langle C_1^*, \ldots, C_m^* \rangle$ that are simultaneously satisfiable by an assignment in $[0,1]_{\text{Ł}}$. We assume without loss of generality that the first k formulas in the list are satisfied by some assignment; hence so are the first k among $\langle C_1, \ldots, C_m \rangle$. By Lemma 2, the same clauses (hence, the same number of clauses) are also simultaneously satisfiable by a *Boolean* assignment. This gives a lower bound on the number of simultaneously satisfiable clauses among $\langle C_1, \ldots, C_m \rangle$ in $\{0,1\}$. At the same time, the two-element Boolean algebra is a subalgebra of $[0,1]_{\text{Ł}}$, so any assignment in $\{0,1\}^n$ is also an assignment in $[0,1]^n$: therefore, considering that k was the answer of the algorithm solving **MaxSAT-OPT**, no more than k clauses among $\langle C_1, \ldots, C_m \rangle$ can be simultaneously satisfiable in $\{0,1\}$, because otherwise k would not be optimal for **MaxSAT-OPT**. Therefore k is the optimal value.

The binary search algorithm always makes a logarithmic number of oracle calls, no matter what the instance is. Also, the complexity analysis as given does not take into account the efficiency of the computations executed by the oracle; all that is known about the oracle is that it correctly decides a particular **NP**-complete problem. Considering the experience obtained in Boolean MaxSAT solvers based on Boolean SAT solvers, there might be alternatives to binary search that might turn out to be more efficient in practice, where one departs from the paradigm that emphasizes worst-case complexity. A typical Boolean MaxSAT solver does a *linear* search, either from unsatisfiable to satisfiable (core-guided approach), or from satisfiable to unsatisfiable (model-guided approach) [8,18]. The solvers heavily exploit the fact that the formulas in the multiset are Boolean clauses (i.e., a *normal form* is assumed) and that a SAT solver also returns a satisfying assignment or an unsatisfiable core; moreover, the calls to the SAT solver need not be its independent runs. These parallels invite an openness of mind when implementing MaxSAT solvers for Łukasiewicz logic.

4 Tableau-Like Method

4.1 Satisfiability

We give first a decision method for the **SAT** problem, combining several approaches that might be termed *analytic*. **SAT** and its complexity have been investigated in depth [1,2,6,7,9,12,14,16,21,23,26]. In particular, tableau calculi have been proposed in [12,24]. Presenting our decision method for **SAT** has several goals. It outlines our approach to a simpler problem than **MaxSAT-OPT**, to be modified in Subsect. 4.2. Our method for **SAT** can then be used as a lower bound on the complexity of the method for **MaxSAT-OPT** in Subsect. 4.2. Furthermore, the method, in its variant generating a tree with an exponential number of branches, provides a simple proof for **SAT** in **NP** and an upper bound on the runtime of a deterministic algorithm for **SAT**.

The method operates in two subsequent stages. The first one is a variant of Tseitin transformation of an arbitrary formula to a formula in *normal form* [31]; in classical logic, the target normal form is a CNF, while in our case, the target normal form is a system of equations in the language $\mathcal{L}(\text{Ł})$. The transformation preserves satisfiability, involves only a polynomial increase in size, and adds new variables. A variant of the transformation was used for testing **SAT** in [9].

Let $||\varphi||$ denote the number of pairwise distinct subformulas in φ.[2] Recall at this point the formula $\rho_{1/k}$ from Sect. 3 and its subformula $(k-1)y$. If brackets in this subformula nest to the right (or to the left), then $||(k-1)y||$ is proportional to $|(k-1)y|$. But if $(k-1)y$ is bracketed as a balanced binary tree, then $||(k-1)y||$ is proportional to $\log_2(|(k-1)y|)$.

[2] φ is viewed as a string, any subformula is a substring, and subformulas are the same if and only if the strings are. Thus $x \oplus (x \oplus x)$ is distinct from $(x \oplus x) \oplus x$. Per convention $\neg\neg\psi$ does not occur as subformula for any ψ, since $\neg\neg\psi \leftrightarrow \psi$ in Ł.

The second stage is a tableau-like procedure that utilizes the system of equations obtained in the first stage as labels for nodes in a rooted linear tree, and expands the nodes using simple rules that translate these equations of $\mathcal{L}(Ł)$ into linear equations in the reals. Subsequently, each branch is evaluated for solvability in the reals, analogously to [12,24].

The algorithm for **SAT** is given below. The presentation is informal.

Decision method $TŁ_{SAT}$. Let $\varphi(x_1, \ldots, x_n)$ be an input formula.

1. **List subformulas.** Let **L** be the list of all pairwise distinct subformulas occurring in φ, including φ and all its variables. Let l be the number of items in **L**. If φ does not contain any double negations, we have $l = ||\varphi||$. We assume that if α is a subformula of β, then α occurs before β in **L**.
2. **Name subformulas.** Introduce new pairwise distinct variables z_i for the i-th formula in **L** with $1 \leq i \leq l$. These will be called "z-variables". It is assumed that the z variables are also distinct from each x_j for $1 \leq j \leq n$.[3]
3. **Equations on names.** Let **S** be the list of equations in the language $\{\neg, \oplus\}$ obtained by initializing **S** as empty and taking the following step for each item in the list **L**:
   - if x is a propositional variable in φ and $1 \leq i \leq l$ and z_i is the variable for x, include in **S** the equation
     $$x = z_i;$$
   - if $\neg\alpha$ is a subformula of φ and $1 \leq i, j \leq l$ and z_i is the variable for α and z_j is the variable for $\neg\alpha$, include in **S** the equation
     $$z_j = \neg z_i;$$
   - if $\alpha \oplus \beta$ is a subformula of φ and $1 \leq i, j, k \leq l$ and z_i, z_j, z_k are the variables for α, β, $\alpha \oplus \beta$ respectively, include in **S** the equation
     $$z_i \oplus z_j = z_k.$$

   Having each item of **L** processed, **S** contains equations in the language $\mathcal{L}(Ł)$. The number of equations in **S** is l.
4. **Initialize tree.** Initialize a rooted tree **T**, linear at this stage, with l nodes. From the root down, label each node of **T** with one equations from **S**. Start with equations containing the x-variables and mark them *final*. Then process those containing $\neg$ and subsequently those containing $\oplus$ and mark each as *active*.[4]
5. **Boundary constraints.** Append before the root $2l$ new nodes labelled $0 \leq z_i \leq 1$ for each $i = 1, \ldots, l$. Mark each as *final*.
6. **Target constraint.** Append as new root of the tree a node labelled $z_l = 1$ for z_l the variable introduced for φ. (By convention taken in step 1, z_l is assigned to φ.) Mark *final*.

[3] This is a convention in favour of clarity of presentation. Avoiding introduction of new variables for atoms $x_1, \ldots, x_n$ would save n new variables.

[4] The structure of **T** will be linear up to a certain point and binary from there on. This is the case because a) the equations with the x-variables are not expanded, and b) all the equations with $\neg$ are expanded before any of the equations with $\oplus$, and the expansion rule for $\neg$ does not lead to branching. Cf. Example 1.

7. **Expand tree.** From the root of **T** towards the leaves, process each node N:
   - If the label of N is marked *final* (i.e, does not contain $\neg$ or $\oplus$), leave it intact and proceed to the next node.
   - If the label of N is marked *active* (contains $\neg$ or $\oplus$), mark it *passive*, and below each leaf of **T**, append a new subtree with labelled nodes using the following **expansion rules** (one new node per each constraint), marking each new label *final*:

$$\frac{z_i \oplus z_j = z_k}{\begin{array}{l|l} z_i + z_j \leq 1 & z_i + z_j \geq 1 \\ z_i + z_j = z_k & z_k = 1 \end{array}} \qquad \frac{z_i = \neg z_j}{z_i = 1 - z_j}$$

   An application of the rule on the left involves branching below each leaf of **T**. The labels in the conclusions of these rules are linear constraints in real numbers. The mark *final* indicates the algorithm leaves them intact. Having processed all nodes of **T**, each branch of **T** defines a system of linear constraints marked *final* in an unambiguous way.
8. **Solve systems.** From the leftmost branch to the right, test the system of constraints on the branch for solvability in $\mathbb{R}$[5] until a branch is found whose system of constraints is solvable. In such a case, return **'yes'** and exit.
9. **Default.** Return **'no'** and exit.

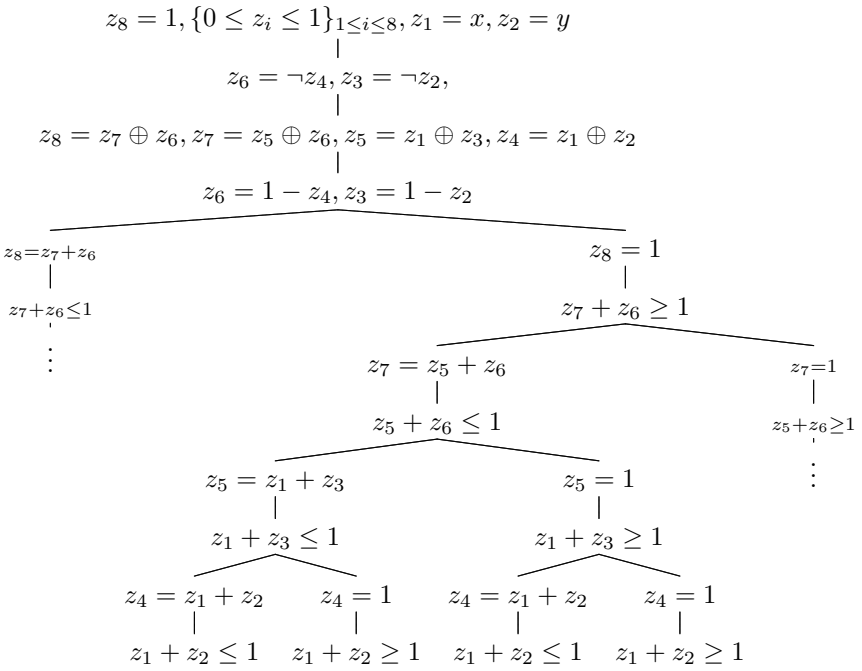
Typically in an analytic tableau method (cf. eg. Hähnle [12]), one starts with a given formula φ and decomposes it, taking one occurrence of a connective in each step and expanding the tableau using the given tableau rules. If a subformula of φ occurs multiple times in φ, it is processed multiple times and each time, new variables are introduced with it: cf. e.g. [12, section 5.1] where new variables i_1 and i_2 are introduced for each occurrence of an implication. This is a feature of the analytic method. With creating the set of subformulas first, we avoid this and have potentially less new variables. (Cf. also the introduction in [24], where our method might therefore not qualify as purely analytic.)

Example 1. A simple example will illustrate the generation of the tree and the resulting systems of constraints. Consider the formula $((x \oplus \neg y) \oplus \neg(x \oplus y)) \oplus \neg(x \oplus y)$. A list of its subformulas is the following:

$$\langle x, y, \neg y, x \oplus y, x \oplus \neg y, \neg(x \oplus y), (x \oplus \neg y) \oplus (\neg(x \oplus y)), ((x \oplus \neg y) \oplus \neg(x \oplus y)) \oplus \neg(x \oplus y) \rangle$$

In order to present the example in a compact way, we write three initial nodes only: the first, with the boundary, target and ground equations; the second, with the equations from **S** with symbol $\neg$, and the third, with the equations from **S** with symbol $\oplus$. Below this, we expand the tree as described by the algorithm. We omit marks (active, passive, final). We use vertical dots to indicate the tree that would be included in their place is a copy of the one depicted at its side.

[5] The testing procedure is in **P**. For the purpose of testing, one can render each equality $\mathbf{ax} = \mathbf{b}$ as two inequalities $\mathbf{ax} \leq \mathbf{b}$ and $-\mathbf{ax} \leq -\mathbf{b}$.



Lemma 3. *The expansion rules in step 7 of* $\mathrm{T}\text{Ł}_{\mathrm{SAT}}$ *preserve the following invariant: for any assignment* v *of values in* $[0,1]$ *to all* z*-variables,* v *satisfies the equation in the premise in the algebra* $[0,1]_{\text{Ł}}$ *if and only if* v *satisfies all constraints in at least one branch in the conclusions of the rule in the algebra* $\mathbb{R}$.

Proof. Notice that the expansion rules work as a switch between the signature of $\mathcal{L}(\text{Ł})$ and language of real closed fields. (Where by slight abuse of language, we only differentiate between the two sets of the operation symbols, but not the relation symbols.) In both cases the statement is a straightforward consequence of the semantics of the connectives $\neg$ and $\oplus$ in $[0,1]_{\text{Ł}}$. We prove the case for $\oplus$. Top-to-bottom: let v be an assignment of values in $[0,1]$ to z-variables introduced in step 2, and consider z_i, z_j, z_k s.t. $v(z_i) \oplus v(z_j) = v(z_k)$ is true in $[0,1]_{\text{Ł}}$. Then it must be the case that either $v(z_i)+v(z_j) \le 1$ and $v(z_i)+v(z_j) = v(z_k)$ holds in $\mathbb{R}$, or $v(z_i)+v(z_j) \ge 1$ in which case we also have $v(z_k) = 1$ in $\mathbb{R}$. Bottom to top: again let v be an assignment of values in $[0,1]$ to z-variables. If $v(z_i)+v(z_j) \le 1$ and $v(z_i) + v(z_j) = v(z_k)$ both hold in $\mathbb{R}$, we have $v(z_i) \oplus v(z_j) = v(z_k)$ is true in $[0,1]_{\text{Ł}}$. If $v(z_i) + v(z_j) \ge 1$ and $v(z_k) = 1$ in $\mathbb{R}$, we have $v(z_i) \oplus v(z_j) = v(z_k)$ is true in $[0,1]_{\text{Ł}}$. This exhausts possible cases.

Theorem 4. *The method* $\mathrm{T}\text{Ł}_{\mathrm{SAT}}$ *is sound and complete for* **SAT**.

Proof. The **soundness** claim states that whenever the method answers 'yes' on input φ, then there is an assignment v to $x_1, \ldots, x_n$ such that $v(\varphi) = 1$. So assume that there is a branch B of $\mathbf{T}$ such that the system of constraints

given by B is solvable, under some assignment v to variables on B, and fix v. In particular, for $i = 1, \ldots, n$, the variable x_i gets value $v(x_i)$ (notice each x_i occurs on every branch). The assignment v extends to φ in a unique way and one shows by induction on the structure of φ, using Lemma 3, that for any subformula ψ of φ, we have $v(\psi) = v(z_j)$ for z_j with $j \in \{1, \ldots, l\}$ being the z-variable assigned to ψ in step 2. In particular, $v(\varphi) = 1$.

The **completeness** claim states that if $v(\varphi) = 1$ for some assignment v, then the method yields 'yes' on input φ. So fix v s.t. $v(\varphi) = 1$. We claim there is a branch of $\mathbf{T}$ with a solvable system of equations. First produce the full tree $\mathbf{T}$. Then assign values to all z-variables, starting from those that are names for $x_1, \ldots, x_n$, and then inductively on the structure of φ using again that $v(\psi) = v(z_j)$ for a z_j assigned to ψ in step 2. This is consistent with equations obtained in step 3. By abuse of language, call this assignment v. The assignment v makes it possible to travel downward from the root of $\mathbf{T}$ via labelled nodes, using Lemma 3 to show that v satisfies each label: in particular if $\mathbf{T}$ branches due to a node with label $z_i \oplus z_j = z_k$, then (assuming the label in the premise is satisfied by v), Lemma 3 guarantees that there is at least one branch on which the new (and hence, all) labels are satisfied by v. Finally a leaf L of $\mathbf{T}$ is reached: since Lemma 3 was applied at each expansion, and since the boundary and the final constraint clearly hold under v, all final constraints on the branch determined by L hold under v.

Lemma 4. *The problem* **SAT** *on instance φ can be solved deterministically by constructing the tree* $\mathbf{T}$ *and testing the solvability of systems of linear constraints in* $\mathbb{R}$ *on no more than* $2^{||\varphi||}$ *branches. Each branch has at most* $4||\varphi|| + 1$ *constraints and* $||\varphi|| + n$ *variables.*

Proof. Branching of the tree takes place at each occurrence of $\oplus$ in $\mathbf{S}$; the number of such occurrences is bounded by $||\varphi||$. Each branch has at most $2||\varphi||$ constraints for subformulas, plus $2||\varphi||$ boundary constraints, plus a target constraint. (Here we do not consider the possibility of replacing each equation with two inequalities.) Each branch of the tree uses all the variables: n input variables $x_1, \ldots, x_n$ and $||\varphi||$ z-variables.

Corollary 3. *The problem* **SAT** *is in* **NP**, *in particular, a formula is satisfiable if and only if there is a polynomial-size witness consisting of a tableau branch of the method* $\mathrm{T}\text{Ł}_{\mathrm{SAT}}$ *and matching system of constraints solvable in* $\mathbb{R}$.

Proof. Since the method $\mathrm{T}\text{Ł}_{\mathrm{SAT}}$ is sound and complete for **SAT** by Theorem 4, any satisfiable formula has the following polynomial-size certificate of its own satisfiability in $[0,1]_{\text{Ł}}$: the system of equations in z-variables constructed in step 3, and a branch of the tree $\mathbf{T}$, defined by a list of instructions specifying which branch to take upon each application of $\oplus$-rule, combined with a system C of constraints that matches the indicated branchings (in the sense that the equations with $\oplus$ have been expanded according to the specified branch) and such that C is solvable in $\mathbb{R}$. On the other hand, the soundness and completeness theorem also says that an unsatisfiable formula *cannot* have such a certificate.

Furthermore, any decision tree obtained from the above procedure can be linearized, using the methods of [12]. In particular, any instance of the application of the branching rule introduced in step 7 can be replaced by an instance of an application of the following lemma (observing the condition that distinct Boolean variables will be used for distinct instances):

Lemma 5. (Cf. [12, Sect. 5.1], [13, Lemma 6.2.19]) *Assume* $a_1, a_2, a_3 \in [0,1]$. *Then* $a_1 \oplus a_2 = a_3$ *holds in* $[0,1]_{\text{Ł}}$ *if and only if there is an* $y \in \{0,1\}$ *such that all of the following constraints hold in* $\mathbb{R}$*:*

(i) $a_1 + a_2 \leq 1 + y$
(ii) $y \leq a_1 + a_2$
(iii) $a_3 \leq a_1 + a_2$
(iv) $a_1 + a_2 \leq a_3 + y$
(v) $y \leq a_3$.

Proof. Assume $a_1 \oplus a_2 = a_3$ holds in $[0,1]_{\text{Ł}}$. Case 1: $a_1 + a_2 \leq 1$, then from the assumption we have $a_1 + a_2 = a_3$. We set $y \coloneqq 0$. The fact that $a_1, a_2, a_3 \in [0,1]$ implies (ii) and (v); the remaining constraints in the Lemma follow from $a_1 + a_2 = a_3$. Case 2: $a_1 + a_2 > 1$. The assumption implies $a_3 = 1$; we set $y \coloneqq 1$, we get (v). The fact that $a_1, a_2, a_3 \in [0,1]$ implies (i) and (iv). From $a_1 + a_2 > 1$ we get (ii) and (iii).
Now assume there is an $y \in \{0,1\}$ such that all constraints listed hold in $\mathbb{R}$. Case 1: $y = 0$. We have (i) $a_1 + a_2 \leq 1$ and (iii,iv) $a_3 \leq a_1 + a_2 \leq a_3$. Hence $a_1 \oplus a_2 = a_3$. Case 2: $y = 1$. We have (v) $1 \leq a_3$ and (ii) $1 \leq a_1 + a_2$. Hence $a_1 \oplus a_2 = a_3$.

This modification eventually yields, in step 8, a single MIP problem — one of the extant competitive ways to address the **SAT** problem. A major advantage of using a MIP solver is the advanced possibility of applying heuristics, whereas in the simple version above, the only optimization considered is aborting the computation upon finding a branch with a solvable system.[6] That is: by design, the algorithm TŁ_{SAT} needs to generate and perhaps eventually test exponentially many systems of equations. However, from the viewpoint of the worst-case deterministic complexity, the MIP method does not differ substantially from testing the (possibly exponentially many) branches.

4.2 Maximum Satisfiability

In this Subsection we adapt the previous method to the **MaxSAT-OPT** problem from Sect. 2. It is easily observed that usual methods for **SAT**, the method from the previous Subsection among them (even if it easily adapts to test joint satisfiability of a list of formulas), are not applicable for **MaxSAT-OPT**; cf. [19] for a discussion. One problem is that they yield a Boolean value. Taking any satisfiable formula α and considering the m-element list $\langle \alpha, \ldots, \alpha \rangle$, for any $m > 1$,

[6] One might optimize by testing immediately on every generated branch and exiting the computation upon finding one with a solvable system. In our exposition though, we prefer to consider the size of the full decision tree.

clearly a complete method needs to produce the answer m on this input. The tableau approaches of [12,24] uses MIP solvers on branches, also returning a Boolean value. Another feature of the method from the previous Subsection is that it considers distinct subformulas as a set; thus any repetition of the same formula in the list on input would be obliterated.

These considerations invite the approach of preserving the Tseitin-like procedure of listing equations obtained from the subformulas, but combining it with:

– updating the target constraint for a multiset of formulas on input, and
– updating the query about the system of constraints obtained on each branch.

The following algorithm updates the decision method $\mathrm{T\L}_{\mathrm{SAT}}$ from the Subsect. 4.1. To highlight the differences, each step only gives the information that has changed compared to the previous case.

Optimization method $\mathrm{T\L}_{\mathrm{MaxSAT}}$ for computing **MaxSAT-OPT**.
Let $\langle \varphi_1, \ldots, \varphi_m \rangle$ be a list of formulas in variables $x_1, \ldots, x_n$.

1. **List subformulas.** Let **L** be the list of all pairwise distinct subformulas occurring in $\varphi_1, \ldots, \varphi_m$, including each formula $\varphi_1, \ldots, \varphi_m$ with $1 \leq i \leq m$ and all variables $x_1, \ldots, x_n$. Let l be the number of items in **L**. Conventions as in step 1 of $\mathrm{T\L}_{\mathrm{SAT}}$.
2. **Name subformulas.** As before.
3. **Equations on names.** As before.
4. **Initialize tree.** As before.
5. **Boundary constraints.** As before.
6. **Mark hard constraints.** For each node in **T** up to this point, mark all constraints as **hard constraints**.
7. **Target constraints.** Append before the root of **T** a new chain with labels $z_{j_i} \geq 1$ for z_{j_i} the variable introduced for φ_i, with $i = 1, \ldots, m$, preserving the multiplicity of φ_i in the input list. Mark these constraints as **soft constraints**.
8. **Expand tree.** As before, preserving in the expansion that a hard constraint produces hard constraints.
9. **Solve systems.** From the leftmost branch to the right, taking one branch at a time. Each branch defines, via the *final* label, a system of linear constraints in $\mathbb{R}$, with the target constraints from step 7 marked *soft* and all other constraints marked *hard*. Thus each branch defines an instance of the Max-FS problem with hard and soft constraints. Obtain the solution (i.e., a natural number, possibly 0) to the instance on each branch.[7]
10. **Maximize.** Return the maximum of satisfied soft constraints among the constraint systems over all the branches, and exit.

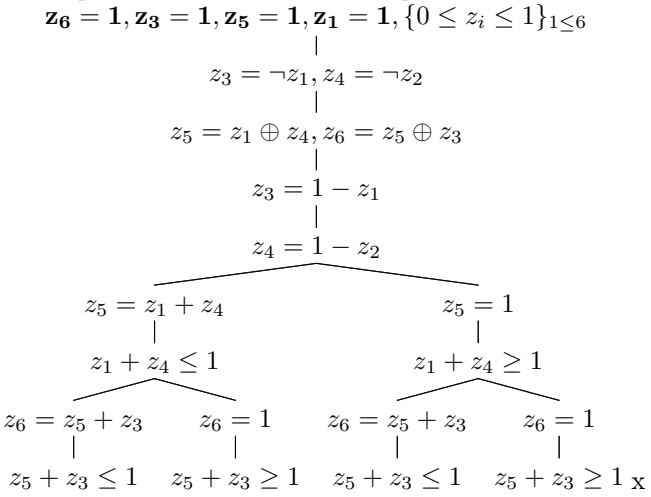
---

[7] Since all equalities are marked hard, any feasible solution to the **Max-FS** task will need to satisfy all of them. More generally, see [5, Concluding remarks] for handling soft constraints that are equalities.

Example 2. Let us consider the list of formulas $\langle (x \oplus \neg y) \oplus \neg x, \neg x, x \oplus \neg y, x \rangle$. A list of its subformulas (according to the definition in step 1) is the following:

$$\langle x, y, \neg x, \neg y, x \oplus \neg y, (x \oplus \neg y) \oplus \neg x \rangle$$

In order to depict the example in a compact way we use the same conventions as in Example 1. Furthermore, we will print in bold the soft constraints.



Theorem 5. *The method* $\mathrm{T}\text{Ł}_{\mathrm{MaxSAT}}$ *is sound and complete for* **MaxSAT-OPT**.

Proof. The **soundness** claim states that whenever the method returns $k \in \mathbb{N}$ on input $\langle \varphi_1, \ldots, \varphi_m \rangle$, then there is an assignment v to variables $x_1, \ldots, x_n$ that satisfies k formulas among $\langle \varphi_1, \ldots, \varphi_m \rangle$. If $\mathrm{T}\text{Ł}_{\mathrm{MaxSAT}}$ returns k, that means the tree $\mathbf{T}$ was constructed with a branch B and a system of constraints given by B that yielded k upon solving the Max-FS problem with hard and soft constraints, and that this was the maximum solution among all branches. Fix such a v and notice that v defines values for $x_1, \ldots, x_n$. Using Lemma 3, all hard constraints from the system, in particular, all constraints from steps 3, 5 and 8 are satisfied by v, and so are k of the target constraints. If ψ is a subformula of some φ_i with $i \in \{1, \ldots, m\}$, we have $v(\psi) = v(z_j)$ whenever z_j is the z-variable assigned to ψ, by induction. In particular, from step 7 we have that there are k formulas φ_i among $\langle \varphi_1, \ldots, \varphi_m \rangle$ such that $v(\varphi_i) = 1$.

The **completeness** claim states that if, for some assignment v, there are k items φ_i on the list $\langle \varphi_1, \ldots, \varphi_m \rangle$ such that $v(\varphi_i) = 1$, then the method $\mathrm{T}\text{Ł}_{\mathrm{MaxSAT}}$ yields at least k on that instance. So assume that $v(\varphi_i) = 1$ for at least k such items and fix v. We claim there is a branch B of $\mathbf{T}$ with a system of constraints that yields at least k upon solving its instance of Max-FS problem. First construct the tree $\mathbf{T}$. From v, we get values for $x_1, \ldots, x_n$, the z-variables that are their names, and using equations from step 3 for the remaining z-variables. The assignment v indicates a leaf of $\mathbf{T}$ that defines a branch B via a series of (possibly

non-unique) choices on the hard constraints. If ψ is a subformula of some φ_i with $i \in \{1, \ldots, m\}$, also $v(\psi) = v(z_j)$ whenever z_j is the z-variable assigned to ψ, all the hard constraints and at least k soft constraints are satisfied on B under v. Since k formulas on input are satisfied by v, also k soft constraints are satisfied. Thus the method $\mathrm{T}\L_{\mathrm{MaxSAT}}$, which returns a maximum over all branches, will yield a value no less than k.

To put side by side the efficiency of the method $\mathrm{T}\L_{\mathrm{SAT}}$ from Subsect. 4.1 with the method $\mathrm{T}\L_{\mathrm{MaxSAT}}$ above, we assume a modification of $\mathrm{T}\L_{\mathrm{SAT}}$ that takes as input a finite list of arbitrary formulas $\langle \varphi_1, \ldots, \varphi_m \rangle$ and tests their joint satisfiability. Then we obtain comparable trees from both methods, the main difference being in the target constraints. Each branch of the tree obtained from $\mathrm{T}\L_{\mathrm{SAT}}$ defines a set of constraints the solvability of which is in **P**. It is typically not necessary to test solvability on all the branches. On the other hand, if $\langle \varphi_1, \ldots, \varphi_m \rangle$ is an input to $\mathrm{T}\L_{\mathrm{MaxSAT}}$, then on each branch of the generated tree, it is indeed necessary to solve the Max-FS problem with hard and soft constraints that the branch defines, because the method eventually takes a maximum over *all* the branches. Moreover, the problem on each branch is **NP**-hard [4]. In this sense, the complexity of the method $\mathrm{T}\L_{\mathrm{SAT}}$ is a *lower bound* on the complexity of the method $\mathrm{T}\L_{\mathrm{MaxSAT}}$ as presented above.

One can conceive optimizing the method $\mathrm{T}\L_{\mathrm{MaxSAT}}$ by observing that, firstly, the multiset of soft constraints remains the same over all the branches, and secondly, if any subset S' of a set S of hard constraints is unsolvable, then so is S. We refrain from pursuing these considerations here, since they are addressed by the methods used in MIP solvers. The following lemma comes in useful.

Lemma 6. *The tree obtained from the* $\mathrm{T}\L_{\mathrm{MaxSAT}}$ *method can be linearized at the cost of adding at most* $||\varphi||$ *Boolean variables. The linearization method does not affect the soft constraints.*

Proof. Any branching in step 8 of the algorithm can be replaced by expanding the tree with new nodes (without branching) using Lemma 5. The constraints obtained from the Lemma are all marked *hard*. This step therefore does not impact the set of possible solutions to the hard constraints in the system. The soft constraints are the same on all the branches, therefore the soft constraints in the linearization are well defined.

An extension of the Max-FS problem with Boolean variables among the set of hard constraints can also be rendered as a MIP problem with hard and soft constraints, with the Boolean variables not occurring in the soft constraints. Section 3 gives as benchmark for **MaxSAT-OPT** $\log m$ calls to a MIP solver for **SAT** with inputs of size $O(\Sigma_{i=1}^{m}|\varphi_i| + m^2)$.

5 Concluding Remarks and Future Work

Envisaged work on this material will consider finite-valued reductions of the **SAT** problem via upper bounds on denominators [1–3] to obtain a comparison

with variants of $\mathrm{T\L}_{\mathrm{SAT}}$ for deterministic worst-case complexity for arbitrary formulas. Also, it remains to be seen whether upper bounds on denominators (a "small-model theorem", cf., e.g., [11]) can be used to classify the decision version of the above Max-FS problem with Boolean variables among its hard constraints within $\mathbf{FP}^{\mathbf{NP}}$ for a conclusive comparison with the canonical approach. Another line of possible work stems from a generalized notion of satisfiability, considering, instead of the MaxSAT family of problems, their MaxSAT_r version, for a rational $r \in (0, 1]$, asking for the maximum number of formulas that are assigned a value greater than or equal to r by a single assignment.

Acknowledgements. We thank three anonymous reviewers for their useful and inspiring comments. Haniková was supported by the long-term strategic development financing of the ICS (RVO:67985807) and by mobility grant no. CSIC-20–12 of the Czech Academy of Sciences. Manyà was supported by grants PID2019-111544GB-C21, PID2022-139835NB-C21 and TED2021-129319B-I00 funded by MCIN/AEI/10.13039/501100011033. Vidal was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 101027914.

References

1. Aguzzoli, S.: An asymptotically tight bound on countermodels for Łukasiewicz logic. Int. J. Approximate Reasoning **43**(1), 76–89 (2006)
2. Aguzzoli, S., Ciabattoni, A.: Finiteness in infinite-valued Łukasiewicz logic. J. Logic Lang. Inf. **9**(1), 5–29 (2000)
3. Aguzzoli, S., Gerla, B.: Finite-valued reductions of infinite-valued logics. Archive Math. Logic **41**(4), 361–399 (2002)
4. Amaldi, E., Kann, V.: The complexity and approximability of finding maximum feasible subsystems of linear relations. Theor. Comput. Sci. **147**, 181–210 (1995)
5. Amaldi, E., Pfetsch, M.E., Leslie, E., Trotter, J.: On the maximum feasible subsystem problem, IISs and IIS-hypergraphs. Math. Program. Ser. A **95**, 533–554 (2003)
6. Ansótegui, C., Bofill, M., Manyà, F., Villaret, M.: Building automated theorem provers for infinitely-valued logics with satisfiability modulo theory solvers. In: Proceedings, 42nd International Symposium on Multiple-Valued Logics (ISMVL), Victoria, BC, Canada, pp. 25–30. IEEE CS Press (2012)
7. Ansótegui, C., Bofill, M., Manyà, F., Villaret, M.: Automated theorem provers for multiple-valued logics with satisfiability modulo theory solvers. Fuzzy Sets Syst. **292**, 32–48 (2016)
8. Bacchus, F., Järvisalo, M., Ruben, M.: Maximum satisfiability. In: Handbook of Satisfiability, second edition, pp. 929–991. IOS Press (2021)
9. Bofill, M., Manyà, F., Vidal, A., Villaret, M.: New complexity results for Łukasiewicz logic. Soft. Comput. **23**, 2187–2197 (2019)
10. Chang, C.C.: A new proof of the completeness of the Łukasiewicz axioms. Trans. Am. Math. Soc. **93**(1), 74–80 (1959)
11. Fagin, R., Halpern, J.Y., Megiddo, N.: A logic for reasoning about probabilities. Inf. Comput. **87**(1–2), 78–128 (1990)
12. Hähnle, R.: Many-valued logic and mixed integer programming. Ann. Math. Artif. Intell. **12**(3–4), 231–264 (1994)

13. Hájek, P.: Metamathematics of Fuzzy Logic, Trends in Logic, vol. 4. Kluwer, Dordrecht (1998)
14. Haniková, Z.: Computational complexity of propositional fuzzy logics. In: Cintula, P., Hájek, P., Noguera, C. (eds.) Handbook of Mathematical Fuzzy Logic, vol. 2, pp. 793–851. College Publications (2011)
15. Haniková, Z.: On the complexity of validity degrees in Łukasiewicz logic. In: Anselmo, M., Della Vedova, G., Manea, F., Pauly, A. (eds.) CiE 2020. LNCS, pp. 175–188. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-51466-2_15
16. Haniková, Z., Savický, P.: Term satisfiability in FL_{ew}-algebras. Theor. Comput. Sci. **631**, 1–15 (2016)
17. Krentel, M.W.: The complexity of optimization problems. J. Comput. Syst. Sci. **36**, 490–509 (1988)
18. Li, C.M., Manyà, F.: MaxSAT, hard and soft constraints. In: Handbook of Satisfiability, second edition, pp. 903–927. IOS Press (2021)
19. Li, C.M., Manyà, F., Vidal, A.: Tableaux for maximum satisfiability in Łukasiewicz logic. In: IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL), pp. 243–248. IEEE Computer Society, Miyazaki (2020)
20. Mundici, D.: Mapping abelian ℓ-groups with strong unit one-one into MV-algebras. J. Algebra **98**(1), 76–81 (1986)
21. Mundici, D.: Satisfiability in many-valued sentential logic is NP-complete. Theor. Comput. Sci. **52**(1–2), 145–153 (1987)
22. Mundici, D.: Ulam game, the logic of MaxSAT, and many-valued partitions. In: Dubois, D., Klement, E.P., Prade, H. (eds.) Fuzzy Sets, Logics and Reasoning about Knowledge, pp. 121–137. Kluwer (1999)
23. Mundici, D., Olivetti, N.: Resolution and model building in the infinite-valued calculus of Łukasiewicz. Theor. Comput. Sci. **200**, 335–366 (1998)
24. Olivetti, N.: Tableaux for Łukasiewicz Infinite-valued Logic. Stud. Logica. **73**, 81–111 (2003)
25. Preto, S., Manyà, F., Finger, M.: Linking Łukasiewicz Logic and Boolean Maximum Satisfiability, ISMVL, pp. 164–169. IEEE Computer Society, Miyazaki (2023)
26. Schockaert, S., Janssen, J., Vermeir, D.: Satisfiability checking in Łukasiewicz logic as finite constraint satisfaction. J. Autom. Reasoning **49**, 493–550 (2012)
27. Schockaert, S., Janssen, J., Vermeir, D., De Cock, M.: Finite satisfiability in infinite-valued Łukasiewicz logic. In: Godo, L., Pugliese, A. (eds.) SUM 2009. LNCS (LNAI), vol. 5785, pp. 240–254. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04388-8_19
28. Schrijver, A.: Theory of Linear and Integral Programming. Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Chichester (1998)
29. Stockmeyer, L.J.: Computational Complexity. In: Coffman, E.G., et al. (eds.) Handbooks in OR & MS, Vol. 3, pp. 455–517. Elsevier Science Publishers (1992)
30. Torrens, A.: Cyclic elements in MV-algebras and post algebras. Math. Logic Q. **40**(4), 431–444 (1994)
31. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Slisenko, A.O. (ed.) Studies in mathematics and mathematical logic, Part II, pp. 115–125. Steklov Mathematical Institute (1968)
32. Vidal, A.: MNiBLoS: a SMT-based solver for continuous t-norm based logics and some of their modal expansions. Inf. Sci. **372**, 709–730 (2016)