NLIPSat: Satisfiability-Based Nonlinear Integer Programming Encoding Toolkit

Zhengling Yangli
Yunnan University, China

Zhifei Zheng
Gaoling School of AI, Renmin University of China, Beijing, China

Sami Cherif
MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

Rui Sá Shibasaki
MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

Chu-Min Li
MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

Abstract

While Maximum Satisfiability (MaxSAT) has been successfully applied to a wide range of combinatorial optimization problems, the encoding of Nonlinear Integer Programming (NLIP) with polynomial functions into MaxSAT has so far only been studied at a theoretical level. In this paper, we introduce NLIPSat, the first tool capable of encoding bounded polynomial NLIP instances directly into Maximum Satisfiability. Building upon recent MaxSAT formulations for polynomial NLIP proposed in [31], NLIPSat enables the encoding of polynomial nonlinear objective functions as weighted soft clauses and also supports the encoding of hard non-linear polynomial constraints within a polynomial setting. Extensive experiments on different benchmarks show that NLIPSat outperforms the state-of-the-art SMT solver Z3 by a wide margin.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Mathematics of computing → Combinatorial optimization

Keywords and phrases Maximum Satisfiability, Nonlinear Integer Programming, Encodings, Tool

Digital Object Identifier 10.4230/LIPIcs.SAT.2026.43

Category Tool Paper

Supplementary Material *Software (Source Code)*: `https://github.com/ZhenglingYangli/NLIPSat-Toolkit` [27]; archived at `swh:1:dir:1567209c46a147ae48868eca1e8a5b44f4fceed3`

Funding This work is partially supported by the project ANR-24-CE23-6126 (BforSAT) and the Chair ANR-19-CHIA-0013 (Massal'IA), co-funded by the French national research agency and the French electricity distribution network operator Enedis. It was also granted access to HPC resources of "Plateforme MatriCS" within University of Picardie Jules Verne, co-funded by the European Union with the European Regional Development Fund (FEDER) and the Hauts-De-France Regional Council.

1 Introduction

The Boolean Satisfiability problem (SAT) [4] asks whether a Conjunctive Normal Form (CNF) formula admits a satisfying assignment. As a foundational NP-complete problem, SAT has matured to the point where modern solvers routinely handle industrial instances with millions of variables [16]. Maximum Satisfiability (MaxSAT) [2, 21] extends SAT from decision to optimization by introducing *soft* clauses alongside mandatory *hard* clauses, where the goal shifts to satisfying the maximum number of soft clauses while satisfying the hard

constraints. It has been applied to many domains including scheduling [9], assembly-line balancing [29], and biological network reasoning [15], and its solvers have been enhanced by integer linear programming (ILP) techniques [28]. In particular, pseudo-Boolean (PB) constraints enable MaxSAT to encode ILP by expressing linear objective functions as weighted soft clauses. However, extending this connection to *nonlinear* integer programming (NLIP) with polynomial functions remains largely unexplored. NLIP arises in portfolio selection [24], network design [12], and the quadratic assignment problem [25]. Even optimizing a degree-4 polynomial over bounded integers is NP-hard [23], while integer programming with unbounded quadratic constraints is undecidable [19].

Recent theoretical work [31] has demonstrated the feasibility of encoding NLIP into MaxSAT. Three integer-to-Boolean encoding strategies were introduced namely one-hot (OH), unary (UNA), and binary (BIN), lifting the traditional integer encodings in [1]. Each encoding transforms a polynomial term into weighted soft clauses whose total satisfied weight equals the term value. An order decomposition technique further controls the exponential growth of high-order terms. The key insight is that a polynomial objective $f(\mathbf{X}) = \sum_j T_j(\mathbf{X})$ can be decomposed term-by-term, with each monomial encoded independently, exposing its cost structure to a conflict-driven MaxSAT solver. However, this work remained purely theoretical, and no practical tool implementing these formulations has been developed.

Current satisfiability-based baselines for bounded polynomial NLIP relies on SMT solvers such as Z3 [11] and cvc5 [5], which combine the CDCL(T) with arithmetic reasoning, incremental linearization [8], and reductions to bit-blasting [20]. While powerful for satisfiability, they handle optimization through iterative bound refinement and do not expose the term-level cost structure to the conflict-driven search. A MaxSAT-based encoding addresses this by decomposing the polynomial objective into per-term soft clauses, enabling direct reasoning about each monomial's contribution. As shown in our experiments, the MaxSAT approach significantly outperforms Z3 across all benchmarks, establishing it as a competitive alternative. The practical challenge lies in bridging the theoretical formulations with real-world input formats, constraint handling, and the interaction between encoding strategy and solver paradigm.

In this paper, we present NLIPSat, the first open-source tool that bridges polynomial NLIP and Weighted Partial MaxSAT. NLIPSat fulfills three complementary roles: as a *solving approach*, it provides a MaxSAT-based paradigm that significantly outperforms the state-of-the-art SMT solver Z3; as a *generic encoding platform*, it exports standard WCNF files so that any off-the-shelf MaxSAT solver can tackle NLIP problems without modification; and as a *reusable library* for encoding nonlinear polynomial constraints into CNF, it can be integrated into arbitrary encoding or algorithmic frameworks. Furthermore, we extend the theoretical framework to polynomial constraints via an adaptive pseudo-Boolean compilation strategy, and address practical challenges.

The remainder of this paper is organized as follows. Section 2 provides background on MaxSAT, NLIP, and integer-to-Boolean encodings. Section 3 recalls the encoding framework from [31]. Section 4 describes the architecture and implementation of NLIPSat. Section 5 presents the experimental evaluation, and Section 6 concludes and discusses future work.

2 Preliminaries

Let X be a set of Boolean variables taking values in $\{0, 1\}$. A *literal* l is either a variable $x \in X$ or its negation $\neg x$. A *clause* C is a disjunction of literals and can be represented as a set of literals. It is *satisfied* by an assignment α if at least one of its literals is assigned

to 1. A formula φ in *Conjunctive Normal Form* (CNF) is a conjunction of clauses. The *Satisfiability* (SAT) problem [4] consists in determining whether there exists an assignment satisfying a given CNF formula.

▶ **Definition 1** (Weighted Partial MaxSAT [2, 21]). *A Weighted Partial MaxSAT instance is a bipartite formula $\varphi = H \cup S$, where H is a set of* hard *clauses that must be satisfied, and S is a set of* soft *clauses, each associated with a positive integer weight w. Equivalently, each soft clause is represented as a pair (C, w), where C is a clause and w is its positive integer weight. The objective is to find an assignment α^* that satisfies all hard clauses while minimizing the total weight of falsified soft clauses: $\alpha^* = \arg\min_{\alpha \models H} \sum_{(C,w)\in S,\, \alpha \not\models C} w$.*

We refer to this generic variant simply as MaxSAT throughout the rest of the paper. We also use pseudo-Boolean (PB) constraints of the form $\sum_j w_j \cdot l_j \bowtie c$, where $w_j \in \mathbb{N}$, $c \in \mathbb{N}$, l_j is a literal, and $\bowtie \in \{=, \geq, \leq\}$, which can be efficiently compiled into CNF [13].

The class of polynomial optimization problems handled by NLIPSat is formalized as follows.

▶ **Definition 2** (Polynomial Term and Order). *A* polynomial term *over integer variables $\mathbf{X} = (X_1, \ldots, X_n)$ is defined as:*

$$T(\mathbf{X}) = c \cdot \prod_{q\in Q} X_q^{k_q}, \quad \text{where } c \in \mathbb{Z},\ k_q \in \mathbb{N}\ \ \forall q \in Q,\ \text{and } Q \subseteq \{1, \ldots, n\}. \tag{1}$$

A term with $k \geq 2$ is called a high-order term.

▶ **Definition 3** (Nonlinear Integer Programming). *A* nonlinear integer program *(NLIP) takes the form:*

$$\max\ f(\mathbf{X}) = \textstyle\sum_j T_j(\mathbf{X}) \quad s.t. \quad g_i(\mathbf{X}) \leq b_i,\ X \in \mathbb{X},\ \mathbb{X} \subset \{0, \cdots, ub\}^n \tag{2}$$

where f and each g_i are polynomial functions and at least one of them is nonlinear.

The bounded integer domains ensure that every NLIP instance has a finite search space, making possible the encoding into a finite Boolean formula. Note that a prerequisite for any MaxSAT-based approach to integer problems is the mapping of bounded integer variables to Boolean variables. Given $X \in \{0, \ldots, ub\}$, three well-known representations [1] are commonly used, each introducing a series of Boolean variables x_r to encode X: a one-hot (OH) encoding introduces $ub{+}1$ Boolean variables with an exactly-one constraint, i.e., $x_r = 1$ iff $X = r$; a unary (UNA) encoding introduces $ub{+}1$ Boolean variables with chain constraints $x_r \Rightarrow x_{r-1}$ for $r \geq 1$, encoding $X = \sum_{r=1}^{ub} x_r$; and a Binary (BIN) encoding introduces $\lfloor \log_2 ub \rfloor + 1$ Boolean variables, encoding $X = \sum_{r=0}^{\lfloor \log_2 ub \rfloor} 2^r x_r$, which is exponentially more compact than OH or UNA. These integer representations can be lifted to perform a direct transformation of NLIP into MaxSAT [31], as will be detailed in Section 3.

3 Theoretical Framework

This section recalls the theoretical foundations from [31] for encoding a polynomial NLIP instance (Definition 3) into Weighted Partial MaxSAT. At a high level, the encoding replaces bounded integer variables with Boolean indicators and uses weighted soft clauses to make the MaxSAT objective reproduce polynomial-term values. We present the framework bottom-up: we first describe how individual polynomial terms are encoded and optimized, and then show how these building blocks are assembled to represent the complete objective and constraints.

One-Hot (OH)

r_1	r_2	w
1	1	1
1	2	2
2	1	2
2	2	4

Unary (UNA)

r_1	r_2	$\tilde{w}$
1	1	1
1	2	1
2	1	1
2	2	1

Binary (BIN)

i	j	w
0	0	1
0	1	2
1	0	2
1	1	4

Figure 1 Encoding $X_1 \cdot X_2$ ($ub = 2$) from Example 4. OH: 3×3 value pairs (zero-weight rows omitted). UNA: all weights collapse to 1 via finite differences. BIN: $\lfloor \log_2 3 \rfloor + 1 = 2$ bits per variable, weights 2^{i+j}.

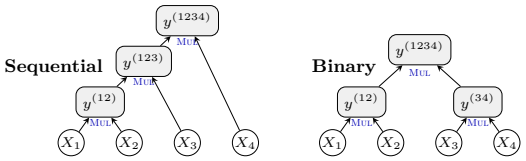

Figure 2 Two decomposition strategies for $X_1X_2X_3X_4$. Both reduce BIN auxiliary variables from $(\log ub)^4$ to $O((\log ub)^2)$ per MUL.

Atomic Term Encoding. The fundamental building block is the encoding of a single polynomial term $T(\mathbf{X}) = c \cdot \prod_{q=1}^{m} X_q^{k_q}$ (order $k = \sum_q k_q$, m distinct variables, all bounded in $\{0, \ldots, ub\}$). For each term, auxiliary Boolean variables z are introduced and linked to the integer-encoding variables x via hard clauses. Weighted soft clauses then capture the term's contribution, ensuring the total satisfied soft weight equals $T(\mathbf{X})$ under any integer assignment. The lifting of the three integer-to-Boolean representations in Section 2 yield three atomic formulations with distinct trade-offs.

The one-hot formulation (**OH**) introduces auxiliary variables $z_{r_1 \ldots r_m}$ for each value combination $(r_1, \ldots, r_m) \in \{0, \ldots, ub\}^m$. Hard clauses enforce $z_{r_1 \ldots r_m} \Leftrightarrow \bigwedge_{q=1}^{m} x_{r_q}^{(q)}$. Soft clauses $(z_{r_1 \ldots r_m},\ c \cdot \prod_{q=1}^{m} r_q^{k_q})$ directly map each value combination to its contribution. The unary formulation (**UNA**) applies the *finite difference* operator [31] to decompose T into incremental contributions: $T(\mathbf{X}) = \sum_{r_1=1}^{X_1} \cdots \sum_{r_m=1}^{X_m} \tilde{T}(r_1, \ldots, r_m)$. By exploiting the ordered structure of unary variables, UNA retains the same variable count as OH but *reduces the maximum weight from* $O(ub^k)$ *to* $O(ub^{k-1})$. The binary formulation (**BIN**) rewrites $T(\mathbf{X}) = c \cdot \prod_{q=1}^{k} X_{\Omega(q)}$ by expanding exponents into repeated factors, where Ω is a mapping such that each original variable X_q appears exactly k_q times in the expanded product. Then, $X_q = \sum_{r=0}^{\lfloor \log_2 ub \rfloor} 2^r x_r^{(q)}$. This yields only $O((\log ub)^k)$ auxiliary variables, exponentially more compact than OH or UNA, though clause weights reach $O(ub^k)$. Intuitively, BIN's compact representation becomes essential as domains grow, while UNA's reduced maximum weight could lower cost for handling cores in core-guided and branch and bound solvers. However, OH's atomic structure where each value combination maps to a single auxiliary variable may also produce small, focused unsatisfiable cores that could benefit core-guided search. Section 5 examines these encoding–solver interactions empirically.

Order decomposition. Under BIN, a term of order k generates $O((\log ub)^k)$ clauses, growing exponentially in k. *Order decomposition* [31] addresses this by recursively factoring a high-order product into a tree of pairwise multiplications. The key operation is $\textsc{Mul}(A, B)$, which encodes the product $A \cdot B$ using binary variables $y^{(ab)}$. It links auxiliary indicators $z_{r_a r_b}^{(ab)} \Leftrightarrow (x_{r_a}^{(a)} \wedge x_{r_b}^{(b)})$, then enforces a PB equality $\sum_{r=0}^{\lfloor \log_2 (ub_a \cdot ub_b) \rfloor} 2^r y_r^{(ab)} = \sum_{r_a}^{\lfloor \log_2 ub_a \rfloor} \sum_{r_b}^{\lfloor \log_2 ub_b \rfloor} 2^{r_a + r_b} z_{r_a r_b}^{(ab)}$. Figure 2 illustrates two partition strategies: *sequential decomposition* (left), and *binary decomposition* (right), which additionally enables reuse of common sub-products across terms.

Complete instance formulation. With the per-term encoding in hand, an entire NLIP instance is constructed by aggregating these blocks. For an *objective* $f(\mathbf{X}) = \sum_j T_j(\mathbf{X})$, each polynomial term T_j is encoded independently to produce a set of weighted soft clauses S_j. The complete objective is captured by the union $S = \bigcup_j S_j$; minimizing the falsified

weight of S equivalently maximizes $f(\mathbf{X})$. For each *constraint* $g_i(\mathbf{X}) \leq b_i$, its terms are similarly encoded to produce auxiliary variables z_ℓ with coefficients w_ℓ. The constraint reduces to a pseudo-Boolean inequality $\sum_\ell w_\ell\, z_\ell \leq b_i$, which is compiled into a set of hard clauses H_{con}. The final Weighted Partial MaxSAT instance is $\varphi = H \cup S$, where the hard clauses $H = H_{\text{var}} \cup H_{\text{link}} \cup H_{\text{con}}$ enforce valid integer representations, term-variable linkages, and polynomial constraints. This reduction from the full instance to a union of per-term encodings is the theoretical basis of NLIPSat.

▶ **Example 4.** Consider max $X_1 \cdot X_2$ s.t. $X_1 + X_2 \leq 3$, $X_1, X_2 \in \{0, 1, 2\}$ (optimum $f^* = 2$). Tracing the bottom-up encoding: Figure 1 shows how the objective term $X_1 \cdot X_2$ (m=2, k=2) is atomically encoded into soft clauses S. The linear constraint $X_1 + X_2 \leq 3$ becomes a PB inequality over the encoding variables and is compiled into hard clauses H_{con}. The complete MaxSAT instance $\varphi = H \cup S$ combines these clauses, and its optimum is $\mathbf{X}^* = (1, 2)$ (or $(2, 1)$) with $f^* = 2$.

The encoding framework shares a surface similarity with bit-blasting approaches for nonlinear integer arithmetic [20], since both ultimately translate arithmetic reasoning over bounded integers into propositional logic. In the SMT setting, bit-blasting encodes each bounded integer as a fixed-width bit-vector and expands arithmetic operations into Boolean circuits, producing a plain SAT instance whose sole semantics are feasibility. Mainstream solvers such as cvc5 [5] apply this as one incomplete tactic within a broader CDCL(T) framework, and Borralleras et al. [8] extend the idea to nonlinear integer optimization by iteratively reducing the problem to linear arithmetic and invoking a Max-SMT oracle at each step. In all these approaches, the polynomial objective is handled *outside* the Boolean core: optimization proceeds through a sequence of feasibility queries with tightening bounds, so the propositional solver never directly reasons about the cost contribution of individual polynomial terms.

The fundamental difference of our approach lies in the semantics of the Boolean encoding itself. Rather than producing a feasibility instance and iterating externally, NLIPSat compiles the entire polynomial objective directly into weighted soft clauses, so that the total satisfied soft weight under any feasible assignment is equal to the value of the objective. This optimization-centered formulation, established in [31], allows MaxSAT conflict analysis to operate at the granularity of individual monomials rather than on a monolithic feasibility formula. In addition, NLIPSat offers three structurally different integer-to-Boolean liftings (OH, UNA, and BIN), each with distinct trade-offs in clause count, maximum soft weight, and solver affinity, rather than committing to a single bit-vector layout. For high-order monomials, order decomposition further rewrites each product as a tree of pairwise multiplications linked by pseudo-Boolean equalities, reducing clause growth from $O((\log ub)^k)$ to $O((\log ub)^2)$ per multiplication step, a transformation that has no direct counterpart in classical bit-blasting. Together with its multi-format parsers, WCNF export, pluggable MaxSAT backends, and solution verification, NLIPSat constitutes a complete optimization-aware toolchain rather than a straightforward reduction of nonlinear arithmetic to SAT.

4 The NLIPSat Tool

The encoding framework of Section 3 is stated at an abstract level and relies on idealized assumptions. To apply it to real NLIP instances from standard benchmarks, a tool must handle input heterogeneity, control formula growth, and ensure end-to-end correctness. NLIPSat addresses these challenges while keeping the formulation modular: encoding choices (OH/UNA/BIN, with optional decomposition) are independent of the back-end solver, and the resulting WCNF can be exported to any external MaxSAT solver.

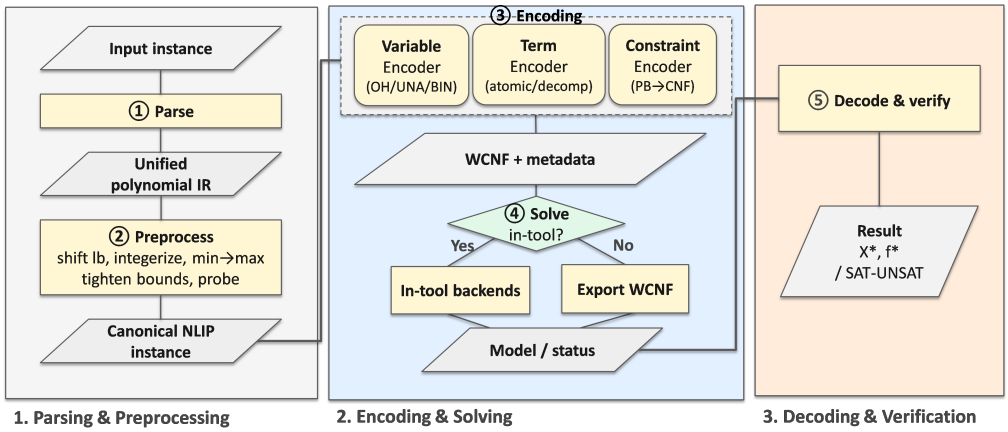

Figure 3 Architecture of NLIPSat. These modules can be used either through the command-line driver or through the `nlipsat` Python API, where users can parse or construct an NLIP instance, encode it into WCNF, and then solve it with a selected backend or export it for an external MaxSAT solver.

4.1 Architecture at a Glance

Figure 3 summarizes NLIPSat as a five-stage pipeline (parsing, preprocessing, encoding, solving, and decoding with verification) with three dedicated encoding components: *Variable Encoder*, *Term Encoder*, and *Constraint Encoder*. The key design decision is to decouple encoding from solving, enabling systematic comparisons across encodings and solvers and supporting NLIPSat's three complementary roles as a solving approach, a generic encoding platform, and a reusable library.

4.2 Design Goals and Key Mechanisms

We organize the implementation around four design goals. Each goal motivates a small set of mechanisms in Figure 3.

Robustness to real inputs. Real instances arrive in heterogeneous formats and rarely satisfy the canonical assumptions of the encodings. NLIPSat implements dedicated parsers for QPLIB [14] and SMT-LIB (QF_NIA[1]) [6], translating both into a unified internal polynomial representation. It then normalizes instances into canonical form by (i) shifting variables with $lb_q \neq 0$ via $X_q \leftarrow X_q - lb_q$ (propagated through all terms), (ii) converting rational coefficients to integers using the least common denominator, and (iii) negating minimization objectives. For SMT-LIB decision instances (no objective), all constraints become hard clauses, and the solver acts as a SAT oracle.

Controlling formula size and weight scale. NLIPSat aligns its encoding implementation with the three components in Figure 3. The *Variable Encoder* maps each bounded integer variable X_q to Boolean variables using a global choice among OH/UNA/BIN (Section 2). The *Term Encoder* applies the term formulations of Section 3 to every term, regardless of whether it appears in the objective or in a constraint. Two practical extensions keep

[1] `https://smt-lib.org/logics-all.shtml#QF_NIA`

```
>>> from nlipsat import load_problem, EncodingConfig, \
...                     build_wcnf, solve, verify_solution
>>>
>>> problem = load_problem("example4.json")
>>> cfg = EncodingConfig()
>>> wcnf, name2idx, vpool = build_wcnf(problem, "BIN", cfg)
>>> print(len(wcnf.hard), len(wcnf.soft))
18 4
>>> print(wcnf.wght)
[1, 2, 2, 4]
>>> print(wcnf.topw)
10
>>>
>>> wcnf.to_file("example.wcnf")
>>> result = solve(problem, encoding="BIN", config=cfg, solver="RC2")
>>> ok, report = verify_solution(problem, result)
>>> print(result["objective_value"], ok)
2 True
```

Figure 4 Using the `nlipsat` API on Example 4: load the instance, construct the WCNF formula, inspect its fields, export, solve, and verify.

encodings compact and solver-friendly: zero-weight clauses are skipped, and negative UNA weights are normalized by literal negation with an objective offset tracked for decoding. Order decomposition (Section 3) is optionally enabled under BIN to mitigate exponential growth on high-order terms. The *Constraint Encoder* aggregates encoded terms into PB inequalities and compiles them into CNF hard clauses using an adaptive strategy: BDD-based encoding [13] by default, switching to adder-based encoding when coefficients exceed a configurable threshold.

Modularity and reproducibility. Encoding and solving are configured independently, enabling systematic evaluation of encoding/solver interaction. NLIPSat can export the produced WCNF for any external solver, or solve internally using the state-of-the-art solvers RC2 [18], MaxHS [10], WMaxCDCL [22], and CaDiCaL [3]. The implementation is written in Python and uses PySAT [17] for CNF construction and solver interfaces, and the toolkit is exposed both as a command-line driver and as the `nlipsat` Python package. Tools such as PBLib [26] or SAT4j [7] operate at the lower level of PB/SAT encodings, while NLIPSat provides the higher-level reduction from complete NLIP instances to MaxSAT.

End-to-end trustworthiness. To guard against encoding bugs and solver errors, NLIPSat performs runtime verification after solving. It decodes the Boolean assignment back to integers, re-evaluates every constraint $g_i(\mathbf{X}) \leq b_i$, and checks that the reconstructed objective value matches the MaxSAT cost (including the offset introduced by negative-weight normalization). Any discrepancy is flagged, providing an end-to-end correctness guarantee.

4.3 Typical Usage

The `nlipsat` Python package is designed to provide a simple way to construct, inspect, and solve MaxSAT encodings of bounded polynomial NLIP instances. The main object manipulated by the package is a PySAT `WCNF` formula, where hard clauses enforce feasibility and weighted soft clauses encode the polynomial objective.An NLIP instance can be loaded from QPLIB, SMT-LIB, Diverse SAT, or JSON files via `load_problem`, or constructed as a Python dictionary with keys `variables`, `objective`, and `constraints`. In Figure 4, the loaded instance corresponds to Example 4: $\max X_1X_2$ subject to $X_1 + X_2 \leq 3$, $X_1, X_2 \in \{0, 1, 2\}$.

Given a problem, `build_wcnf` creates the WCNF formula for a chosen encoding (`"OH"`, `"UNA"`, or `"BIN"`) and an optional `EncodingConfig` that controls order decomposition and weight normalization. It returns the WCNF object, a variable-name map (`name2idx`), and the identifier pool (`vpool`) for decoding solver assignments back to integer values. The returned object is a standard PySAT `WCNF` formula and stores hard clauses in `hard`, soft clauses in `soft`, weights in `wght`, and the top weight in `topw`. The `solve` function returns a result dictionary with the decoded assignment and objective value, which `verify_solution` checks against the original constraints.

For benchmark-scale runs, the same functionality is exposed by the command-line driver `python3 codes/main.py`, including encoding selection (`-e`), backend-solver selection (`-s`), order decomposition (`-decomp`), solution verification (`-verify`), and WCNF export (`-s NONE -o out.wcnf`).

5 Experiments

All experiments were conducted on the MatriCS platform[2], with a CentOS 8.6 system equipped with an Intel Xeon E5-2680 v4 processor operating at a base frequency of 2.40 GHz with Turbo Boost capability up to 3.30 GHz. The cutoff time set for all solvers is 3600 seconds and the cutoff memory is 16 GB per instance. All solvers ran on the same cluster node under identical settings, and our preliminary experiments confirmed reproducibility.

We evaluate on three benchmark suites covering optimization, combinatorial diversity, and satisfiability: (i) **QPLIB** [14]: 137 optimization instances with bounded integer variables, featuring quadratic terms in objectives, constraints, or both; (ii) **Diverse SAT** [30]: 108 instances derived from the diverse satisfiability problem, which we encode as polynomial NLIP instances; (iii) **SMT-LIB** (QF_NIA) [6]: 150 satisfiability instances from the quantifier-free nonlinear integer arithmetic category, with variable domains restricted to $[0, 10]$. An overview of instance features is given in Table 1.

For QPLIB and Diverse SAT, we evaluate three encoding strategies (OH, UNA, BIN) each paired with three MaxSAT solvers (RC2, MaxHS, WMaxCDCL). For SMT-LIB, we evaluate four encodings (OH, UNA, BIN, BIN+DECOMP) with CaDiCaL. As a baseline, we use the SMT solver **Z3** [11], the recent winner of the SMT-COMP QF_NonLinearIntArith track, invoked in optimization mode for QPLIB and Diverse SAT and in QF_NIA mode for SMT-LIB. Z3 is the natural front-end baseline because it supports both satisfiability and optimization, providing a single uniform comparison point across the three benchmark families. By contrast, SAT, MaxSAT, and PB engines operate one abstraction level below and serve as the backend technologies that NLIPSat builds upon, rather than as alternative end-to-end NLIP solvers.

Table 2 gives an overview of all configurations. For QPLIB and Diverse SAT the numbers are given for optimally solved instances. Two observations stand out. First, NLIPSat dramatically outperforms the state-of-the-art SMT solver Z3 on every benchmark by a wide margin: it solves up to 25 times more instances on QPLIB (25 vs. 1), 7 times more instances on Diverse SAT (91 vs. 13), and decides 149 vs. 138 on SMT-LIB while uniquely deciding 11 instances where Z3 returns UNKNOWN. Second, the best encoding–solver pairing is dataset-dependent (OH_RC2, BIN_WMaxCDCL, BIN+DECOMP_CaDiCaL), confirming the value of NLIPSat's modular design.

[2] `https://www.matrics.u-picardie.fr/`

Table 1 Benchmark features and generated (W)CNF sizes (median / max). For the original problems we report variables, constraints, and the highest polynomial degree. For each encoding we report the number of Boolean variables (*Vars*) and clauses (*Cls*) in the produced (W)CNF instances.

		Original problem			Generated (W)CNF size			
Benchmark	#Inst	Vars	Constr.	Deg.	OH	UNA	BIN	DECOMP
QPLIB	137	220 (10 000)	50 (13 090)	2	*Vars*: 5 150 (88 017) *Cls*: 20 100 (592 088)	5 150 (88 017) 20 000 (591 812)	4 542 (87 741) 19 900 (591 536)	– –
Diverse SAT	108	475 (83 908)	1 951 (276 106)	2	*Vars*: 11 400 (2 013 792) *Cls*: 20 652 (3 321 125)	11 400 (2 013 792) 19 862 (3 153 303)	7 600 (1 342 528) 20 652 (3 321 178)	– –
SMT-LIB	150	152 (985)	143 (985)	4–18	*Vars*: 5 467 358 (6 262 911) *Cls*: 33 700 234 (39 342 109)	6 921 415 (11 322 288) 20 091 666 (30 861 991)	789 153 (2 431 045) 2 589 314 (12 769 602)	208 074 (1 348 188) 616 737 (6 426 084)

Table 2 Solved instances across all benchmarks. Best results are highlighted in bold. Blank entries indicate solver–benchmark incompatibility. CaDiCaL is a SAT solver while QPLIB and Diverse SAT require MaxSAT solvers. BIN+DECOMP is reported only for SMT-LIB, where degree ≥ 3 terms occur.

	QPLIB (137)			Diverse SAT (108)			SMT-LIB (150)			
Solver	OH	UNA	BIN	OH	UNA	BIN	OH	UNA	BIN	BIN+D
RC2	7	6	5	**91**	87	86	–	–	–	–
MaxHS	22	20	20	85	66	80	–	–	–	–
WMaxCDCL	23	24	**25**	85	64	64	–	–	–	–
CaDiCaL	–	–	–	–	–	–	33	53	140	**149**
Z3		1			13			138		

QPLIB. NLIPSat dramatically outperforms Z3 on this benchmark: the best configuration (BIN_WMaxCDCL) solves 25 instances optimally, compared to only 1 for Z3. On medium-size instances (2 580–10 286 vars), WMaxCDCL solves 9/11, demonstrating strong scalability. RC2 struggles because quadratic terms produce large clause weights that increase core-guided iteration cost.

Diverse SAT. All 9 NLIPSat configurations dramatically outperform Z3 on this benchmark: the best configuration (OH_RC2) solves 91 out of 108 instances optimally, whereas Z3 solves only 13. Even the weakest NLIPSat configuration (64 optimal) solves nearly 5 times more instances than Z3. OH encoding consistently yields the highest solve count across all three solvers (91, 85, 85 for RC2, MaxHS, WMaxCDCL). We therefore use solved instances as the primary comparison metric for Diverse SAT.

SMT-LIB. BIN decides 140 instances, BIN+DECOMP decides 149 out of 150 instances, surpassing Z3 (138 instances) by a wide margin. Overall, BIN and BIN+DECOMP are clearly more robust, with BIN+DECOMP deciding 149 instances overall and 11 instances on which Z3 returns UNKNOWN.

Figure 5 plots the cumulative solved instances over time. On the combined QPLIB + Diverse SAT benchmark (245 instances), OH_RC2 rises steeply due to its dominance on Diverse SAT, while WMaxCDCL-based curves ultimately reach the highest counts thanks to strong QPLIB performance. RC2 curves lag because quadratic terms produce large clause weights that increase core-guided iteration cost. On SMT-LIB, BIN and BIN+DECOMP dominate while OH and UNA stall early due to memory exhaustion. These patterns confirm that BIN's $O((\log ub)^k)$ compactness is critical at wider domains, while OH's atomic structure benefits core-guided solvers on smaller domains.

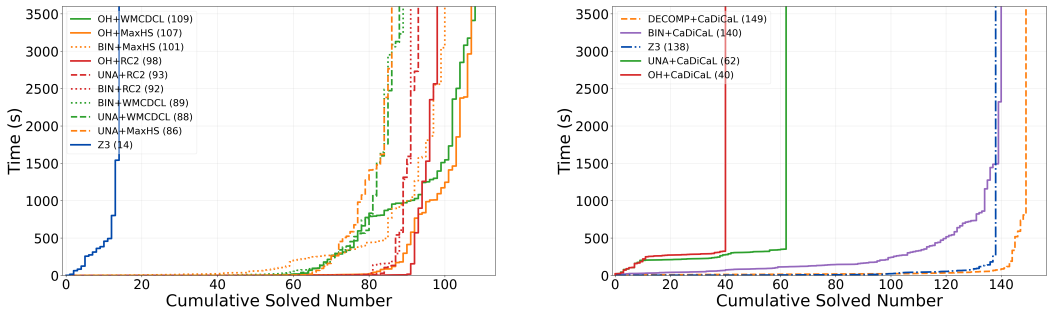

Figure 5 Accumulated solved instances with respect to solving time for QPLIB + Diverse SAT (left, 245 instances) and SMT-LIB (right, 150 instances).

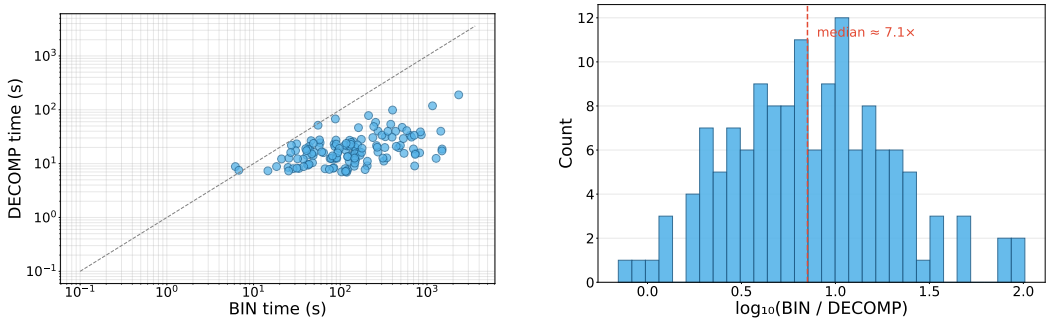

Figure 6 BIN vs. BIN+DECOMP on SMT-LIB with per-instance runtime (left) and speedup distribution (right, median≈7×).

We also examine the effect of order decomposition on the 150 SMT-LIB instances (Figure 6). BIN+DECOMP decides 9 additional instances over BIN (149 vs. 140) while losing none. Among the 10 instances that BIN fails to decide, 6 exhaust the 16 GB memory limit during encoding (within 250 s), whereas DECOMP decides them all within 1.6–3.0 GB; the remaining 4 instances reach the time limit of 3600 s. DECOMP decides 3 of them in 335–591 s, while 1 remains undecided by both configurations. Among the 140 commonly decided instances, DECOMP is faster on 138 (98.6%), with a median BIN/DECOMP runtime ratio of ≈7×, confirming that decomposition effectively reduces clause explosion for high-order terms.

6 Conclusion

We have presented NLIPSat, a modular open-source tool and encoding library that bridges Nonlinear Integer Programming with polynomial functions and Maximum Satisfiability. NLIPSat fulfills three complementary roles. As a competitive satisfiability-based solving approach, it provides an approach that dramatically outperforms the state-of-the-art SMT solver Z3 by a wide margin across different benchmarks. As a generic encoding platform, it exports standard WCNF files so that any off-the-shelf MaxSAT solver can tackle NLIP problems without modification, making it straightforward to evaluate new solvers on nonlinear benchmarks. As a reusable library, it provides modular encoding primitives for nonlinear polynomial constraints into CNF, which can be integrated into arbitrary encoding or algorithmic frameworks.

NLIPSat is available at `https://github.com/ZhenglingYangli/NLIPSat-Toolkit` and several promising directions remain for future work: (i) adaptive per-term encoding selection based on monomial structure, (ii) hybrid approaches combining MaxSAT with SMT-based bound inference, (iii) extending the framework to handle polynomial constraints with real-valued relaxations, and (iv) comparing NLIPSat with mathematical-programming solvers for nonlinear integer optimization, which follow a different continuous-relaxation and branch-and-bound paradigm.

References

1 Carlos Ansótegui and Felip Manyà. Mapping problems with finite-domain variables into problems with boolean variables. In *SAT 2004 – The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*, 2004. URL: `http://www.satisfiability.org/SAT04/programme/53.pdf`.

2 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 929–991. IOS Press, 2021. `doi:10.3233/FAIA201008`.

3 Tomáš Balyo, Nils Froleyks, Marijn JH Heule, Markus Iser, Matti Järvisalo, and Martin Suda. Proceedings of sat competition 2020: Solver and benchmark descriptions. *Department of Computer Science Series of Publications B ; Report B-2020-1 ; B-2020-1*, 2020.

4 Tomás Balyo, Marijn J. H. Heule, and Matti Järvisalo. SAT competition 2016: Recent developments. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 5061–5063. AAAI Press, 2017. `doi:10.1609/AAAI.V31I1.10641`.

5 Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, Lecture Notes in Computer Science, pages 415–442. Springer, 2022. `doi:10.1007/978-3-030-99524-9_24`.

6 Clark Barrett, Aaron Stump, Cesare Tinelli, et al. The smt-lib standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK)*, volume 13, page 14, 2010.

7 Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *J. Satisf. Boolean Model. Comput.*, 7(2-3):59–6, 2010. `doi:10.3233/SAT190075`.

8 Cristina Borralleras, Daniel Larraz, Enric Rodríguez-Carbonell, Albert Oliveras, and Albert Rubio. Incomplete SMT techniques for solving non-linear formulas over the integers. *ACM Trans. Comput. Log.*, 20(4):25:1–25:36, 2019. `doi:10.1145/3340923`.

9 Sami Cherif, Heythem Sattoutah, Chu-Min Li, Corinne Lucet, and Laure Brisoux Devendeville. Minimizing working-group conflicts in conference session scheduling through maximum satisfiability (short paper). In Paul Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming, CP 2024, Girona, Spain, September 2-6, 2024*, volume 307 of *LIPIcs*, pages 34:1–34:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. `doi:10.4230/LIPIcs.CP.2024.34`.

10 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming – CP 2011 – 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011. `doi:10.1007/978-3-642-23786-7_19`.

11 Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. doi:10.1007/978-3-540-78800-3_24.

12 Marco A. Duran and Ignacio E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Math. Program.*, 39(3):337, 1987. doi:10.1007/BF02592081.

13 Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *J. Satisf. Boolean Model. Comput.*, 2(1-4):1–26, 2006. doi:10.3233/SAT190014.

14 Fabio Furini, Emiliano Traversi, Pietro Belotti, Antonio Frangioni, Ambros M. Gleixner, Nick Gould, Leo Liberti, Andrea Lodi, Ruth Misener, Hans D. Mittelmann, Nikolaos V. Sahinidis, Stefan Vigerske, and Angelika Wiegele. QPLIB: a library of quadratic programming instances. *Math. Program. Comput.*, 11(2):237–265, 2019. doi:10.1007/S12532-018-0147-4.

15 João Guerra and Inês Lynce. Reasoning over biological networks using maximum satisfiability. In Michela Milano, editor, *Principles and Practice of Constraint Programming – 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 941–956. Springer, 2012. doi:10.1007/978-3-642-33558-7_67.

16 Aarti Gupta, Malay K. Ganai, and Chao Wang. Sat-based verification methods and applications in hardware verification. In Marco Bernardo and Alessandro Cimatti, editors, *Formal Methods for Hardware Verification, 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2006, Bertinoro, Italy, May 22-27, 2006, Advanced Lectures*, volume 3965 of *Lecture Notes in Computer Science*, pages 108–143. Springer, 2006. doi:10.1007/11757283_5.

17 Alexey Ignatiev, António Morgado, and João Marques-Silva. Pysat: A python toolkit for prototyping with SAT oracles. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing – SAT 2018 – 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2018. doi:10.1007/978-3-319-94144-8_26.

18 Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019. doi:10.3233/SAT190116.

19 Robert G. Jeroslow. There cannot be any algorithm for integer programming with quadratic constraints. *Oper. Res.*, 21(1):221–224, 1973. doi:10.1287/OPRE.21.1.221.

20 Fuqi Jia, Rui Han, Pei Huang, Minghao Liu, Feifei Ma, and Jian Zhang. Improving bit-blasting for nonlinear integer constraints. In René Just and Gordon Fraser, editors, *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Seattle, WA, USA, July 17-21, 2023*, pages 14–25. ACM, 2023. doi:10.1145/3597926.3598034.

21 Chu Min Li and Felip Manyà. Maxsat, hard and soft constraints. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 903–927. IOS Press, 2021. doi:10.3233/FAIA201007.

22 Shuolin Li, Chu-Min Li, Jordi Coll, Djamal Habet, and Felip Manyà. Improving the lower bound in branch-and-bound algorithms for maxsat. In Toby Walsh, Julie Shah, and Zico Kolter, editors, *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 – March 4, 2025, Philadelphia, PA, USA*, pages 11272–11281. AAAI Press, 2025. doi:10.1609/AAAI.V39I11.33226.

23 Jesús A. De Loera, Raymond Hemmecke, Matthias Köppe, and Robert Weismantel. Integer polynomial optimization in fixed dimension. *Math. Oper. Res.*, 31(1):147–153, 2006. doi:10.1287/MOOR.1050.0169.

24 Harry M Markowitz. *Portfolio selection: efficient diversification of investments*. Yale university press, 2008.

25 Panos M. Pardalos, Franz Rendl, and Henry Wolkowicz. The quadratic assignment problem: A survey and recent developments. In Panos M. Pardalos and Henry Wolkowicz, editors, *Quadratic Assignment and Related Problems, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-21, 1993*, volume 16 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–42. DIMACS/AMS, 1993. doi:10.1090/DIMACS/016/01.

26 Tobias Philipp and Peter Steinke. PBLib – A library for encoding pseudo-boolean constraints into CNF. In Marijn Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015 – 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, Lecture Notes in Computer Science, pages 9–16. Springer, 2015. doi:10.1007/978-3-319-24318-4_2.

27 Zhengling Yangli, Zhifei Zheng, Sami Cherif, Rui Sá Shibasaki, and Chu-Min Li. NLIPSat. Software, swhId: swh:1:dir:1567209c46a147ae48868eca1e8a5b44f4fceed3 (visited on 2026-06-29). URL: https://github.com/ZhenglingYangli/NLIPSat-Toolkit, doi:10.4230/artifacts.26928.

28 Jialu Zhang, Chu Min Li, Sami Cherif, Shuolin Li, and Zhifei Zheng. Integer linear programming preprocessing for maximum satisfiability. In *37th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2025, Athens, Greece, November 3-5, 2025*, pages 408–414. IEEE, 2025. doi:10.1109/ICTAI66417.2025.00061.

29 Zhifei Zheng, Sami Cherif, and Rui Sá Shibasaki. Optimizing power peaks in simple assembly line balancing through maximum satisfiability. In *36th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2024, Herndon, VA, USA, October 28-30, 2024*, pages 363–370. IEEE, 2024. doi:10.1109/ICTAI62512.2024.00060.

30 Zhifei Zheng, Sami Cherif, Rui Sá Shibasaki, Chu Min Li, and Jialu Zhang. Exact approaches for the diverse satisfiability problem. In Giovanni Casini, Besik Dundua, and Temur Kutsia, editors, *Logics in Artificial Intelligence – 19th European Conference, JELIA 2025, Kutaisi, Georgia, September 1-4, 2025, Proceedings, Part II*, volume 16094 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 2025. doi:10.1007/978-3-032-04590-4_15.

31 Zhifei Zheng, Sami Cherif, Rui Sá Shibasaki, Chu Min Li, and Jialu Zhang. Maximum satisfiability formulations for nonlinear integer programming. In Giovanni Casini, Besik Dundua, and Temur Kutsia, editors, *Logics in Artificial Intelligence – 19th European Conference, JELIA 2025, Kutaisi, Georgia, September 1-4, 2025, Proceedings, Part II*, volume 16094 of *Lecture Notes in Computer Science*, pages 190–206. Springer, 2025. doi:10.1007/978-3-032-04590-4_14.