Hermax: A Unified MaxSAT Library

Josep Maria Salvia Hornos
Department of Computer Engineering and Digital Design, University of Lleida, Spain
Department of IT and Systems, AI Division, Corporación Alimentaria Guissona S.A., Lleida, Spain

Cèsar Fernández Camón
Department of Computer Engineering and Digital Design, University of Lleida, Spain

Carles Mateu Piñol
Department of Computer Engineering and Digital Design, University of Lleida, Spain

Abstract

Despite the utility of Maximum Satisfiability (MaxSAT) in discrete optimization, developing iterative workflows remains cumbersome due to fragmented, low-level solver APIs. We present Hermax, a unified Python library and modelling compiler for MaxSAT. Hermax provides an IPAMIR interface that exposes incremental solving, assumptions, and weight updates through a single API across many incremental and non-incremental backends. Furthermore, it introduces a compiler with Constraint Programming primitives that translates high-level models directly into optimized CNF/WCNF through eager evaluation. This compiler allows automatic optimizations like integer ladder graph encoding that bypasses Pseudo-Boolean formulation when possible. Together, these features enable rapid prototyping and production grade optimization directly from Python across major platforms and hardware architectures.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Constraint and logic programming; Mathematics of computing → Discrete optimization; Software and its engineering → Software libraries and repositories

Keywords and phrases MaxSAT, Incremental Solving, IPAMIR, Python, Constraint modelling

Digital Object Identifier 10.4230/LIPIcs.SAT.2026.41

Category Tool Paper

Supplementary Material *Software (Source Code)*: `https://github.com/josalhor/hermax` [63]
Software (Distribution): `https://pypi.org/project/hermax` [62]
Text (Documentation): `https://hermax.readthedocs.io/` [61]

Funding Josep Maria Salvia Hornos reports funding from Generalitat de Catalunya AGAUR DI-2024-00020. Cèsar Fernández reports funding by the Spanish Ministry of Science and Innovation MCIN/AEI/10.13039/501100011033/ FEDER, UE within the RevTour project (Ref: PID2022-138564OA-I00).

Acknowledgements We want to thank Alexey Ignatiev and Joao Marques-Silva for participation in tangential MaxSAT research and Elena Barrachina for helping review library documentation.

1 Introduction

In the last twenty years, the efficiency of SAT solvers has enjoyed great success [4, 26], driven by optimized incremental Conflict-Driven Clause Learning (CDCL) engines such as Glucose, COMiniSatPS, and CaDiCaL [10, 53, 17]. Maximum Satisfiability [18] builds upon this success to find assignments that satisfy all hard constraints while maximizing the weight of satisfied soft constraints. However, the development of applications utilizing these engines remains hindered by the low-level programmatic interface that must be used.

© Josep Maria Salvia Hornos, Cèsar Fernández Camón, and Carles Mateu Piñol;
licensed under Creative Commons License CC-BY 4.0

Simultaneously, Python has emerged as one of the most preferred programming languages for rapid development of applications because of its straightforward syntax. While the SAT community has seen successful integration of Python libraries, MaxSAT prototyping has remained dispersed [64].

1.1 Contributions

In this paper, we present Hermax, a unified library that brings together decades of advances in MaxSAT solving in an accessible Python package. Our contributions with Hermax in this paper are:

- **Python IPAMIR API.** Hermax abstracts the differences over numerous incremental and non-incremental MaxSAT engines. This is possible by adopting an IPAMIR interface as designed in [52]. Adopting this interface allows users to swap backends without altering application logic and do so efficiently through a Python API. Bindings or reimplementations of Cardinality, Pseudo-Boolean (PB) and PB(AMO) constraints are also incorporated for low level manipulation.
- **CP to MaxSAT Compiler.** We provide a modelling compiler that translates Constraint Programming (CP) primitives directly into optimized CNF/WCNF. We document efficient integer equation encoding by graph implication of the ladder encoding, which allows us to bypass PB encoders for many affine constraints. The compiler supports incremental queries, floating and negative objectives, and lexicographic strategies.
- **Distribution, Examples, and Documentation.** We distribute precompiled wheels via PyPI for Windows, Linux, and macOS across both x86 and ARM architectures. This required solver specific patching and extensive testing. Moreover, we provide online documentation on the internals and over 30 optimization examples.

2 Related Works

High-level Python interfaces are ubiquitous across optimization, supported natively by solvers for MILP (CPLEX, Gurobi, HiGHS [31], PuLP [60, 36], SCIP [2, 44]), CP (CP-SAT [57], PyChoco [43]), and SMT (Z3 [23], cvc5 [13]), alongside modelling layers like MiniZinc [50], Pyomo [28], CVXPY [24], and CPMpy [27].

Among these systems, CPMpy is the closest point of comparison to Hermax because it offers a Python CP modelling layer with a MaxSAT target. However, CPMpy exhibits limitations: its MaxSAT capability is restricted to non-incremental RC2 and it relies on a generic decomposition pipeline where integer arithmetic is linearized via one-hot encodings.

In the Boolean domain, several libraries exist. PySAT [32] provides a toolkit for SAT prototyping with bindings for solvers and cardinality encodings. OptiLog [8] provides a Python framework for developing SAT-based systems, allowing dynamic loading of incremental SAT solvers and modelling of non-CNF formats [3] and PB support. RustSAT [37] is a modern library with Rust and Python Bindings, which, similar to PySAT, provides access to state of the art SAT solvers and includes various cardinality and PB encodings and utilities. However, all three libraries operate at the propositional solver level.

Despite two decades of Maximum Satisfiability research and the annual MaxSAT Evaluation [9, 16], the Python ecosystem lacks a high performance library dedicated to MaxSAT. Most prominently, the well known RC2 solver [33] has been available through PySAT, but an abstraction capable of calling various engines has remained missing.

Furthermore, the formalization of the IPAMIR [52] Interface (akin to IPASIR [12] for MaxSAT) has provided a common base for stateful MaxSAT, driving applications across machine learning, argumentation, and formal verification [30, 38, 40, 67, 65].

3 The Hermax Library

Taking its name from Hermes, the messenger in Greek mythology who moves between the realms, Hermax (Hermes + MaxSAT) is an emissary between worlds: the Pythonic world and the low-level reality of high performance solvers.

3.1 Design Principles and API

Hermax is designed to provide a unified, solver agnostic Python API that adheres to the IPAMIR interface. IPAMIR enables users to incrementally construct and modify formulas with irremovable clauses and update the weights of soft literals. Furthermore, it supports optimization under temporary assumptions, and allows for the optional registration of a callback. Hermax maps the IPAMIR C interface directly to an abstract Python base class, `IPAMIRSolver`. Figure 1 illustrates this equivalence.

IPAMIR C API

```c
void* ipamir_init();

void ipamir_add_hard(void* solver,
                     int32_t lit_or_zero);

void ipamir_add_soft_lit(void* solver,
                         int32_t lit,
                         uint64_t weight);

void ipamir_assume(void* solver, int32_t lit);
int ipamir_solve(void* solver);

// Status is returned directly by ipamir_solve

uint64_t ipamir_val_obj(void* solver);

int32_t ipamir_val_lit(void* solver, int32_t lit);
```

Hermax Python API

```python
class IPAMIRSolver:
  def __init__(self):

  def add_clause(self, clause: list[int]):

  def set_soft(self,
               lit: int,
               weight: int):

  # Assumptions are passed directly to solve
  def solve(self,
            assumptions: list[int]=None) -> bool:

  def get_status(self) -> SolveStatus:

  def get_cost(self) -> int:

  def val(self, lit: int) -> int:
```

Figure 1 Mapping between the IPAMIR interface and the Hermax base class.

Hermax also introduces abstractions that remain fully compliant with the IPAMIR contract. For example, while IPAMIR expects unit soft literals, Hermax provides an `add_soft_relaxed` helper that handles the introduction of relaxation variables and the hardening process automatically as well as a `get_model` for collecting entire solutions.

3.2 Supported Solvers

Hermax categorizes its MaxSAT engines into two families: native incremental solvers that internally implement the IPAMIR standard, and non-incremental adapters that emulate this interface through a wrapper layer. At the lowest level, lightweight `pybind11` wrappers expose the C IPAMIR as a Python object. This allows routing formulas in-memory and avoids disk I/O and parsing bottlenecks. Above these abstractions, Hermax implements high-level solver classes inheriting from `IPAMIRSolver`.

The `hermax.incremental` module provides direct access to the solvers available through the MaxSAT incremental competition, primarily: UWrMaxSAT [59], featuring both an updated Github 1.8 and the 1.4 competition version, as well as EvalMaxSAT [11] and iMaxHS [51].

The `hermax.non_incremental` module groups wrapper adapters used when a backend is invoked, emulating an incremental API. These adapters allow solvers like CASHWMaxSAT [55], RC2 [33], MaxHS [21, 22], CGSS [34], WMaxCDCL [20], OpenWBO [46] (with OLL [5, 48] and MSU3 support [45]) to be used in the same manner as their incremental counterparts. Hermax also supports process isolated incomplete MaxSAT solvers that are run natively thanks to the `hermax.non_incremental.incomplete` module, with OpenWBOInc [42], TT-OpenWBO [49], SPB-MaxSAT-c-FPS [68], Loandra [15], and NuWLS-c-IBR [41]. For incomplete solvers engineered for one-shot solving, Hermax utilizes processes to safely interrupt execution, extract the best known solution, and reclaim memory resources.

3.3 Incremental MaxSAT Example

To demonstrate Hermax's incremental capabilities, we model a dynamic WiFi frequency assignment problem where routers in a network graph $G = (R, E)$ must be assigned an available frequency $f \in F$. Using a variable $x_{r,f}$ to indicate if router r is assigned frequency f; we ensure exactly one frequency per router ($\forall r \in R, \bigvee_{f \in F} x_{r,f}$ and $\bigwedge_{f \neq f'} \neg(x_{r,f} \wedge x_{r,f'})$) and prevent adjacent routers from sharing the same active frequency ($\forall (u, v) \in E, \forall f \in F, \neg(x_{u,f} \wedge x_{v,f})$). To minimize network congestion, we introduce soft unit clauses ($\neg x_{r,f}$) assigned with penalty weights $W_{r,f}$, emulating exogenous frequency overlap. Astute readers will realize that this is akin to the NP-hard graph coloring problem, with many variations studied in the literature [54].

```python
from hermax.incremental import UWrMaxSAT
from itertools import combinations, product

E, R, F, W = ...
solver = UWrMaxSAT()

x = {(r, f): i for i, (r, f) in enumerate(product(R, F), 1)}

for (u, v), f in product(E, F):
    solver.add_clause([-x[u, f], -x[v, f]])

for r in R:
    solver.add_clause([x[r, f] for f in F])
    for f1, f2 in combinations(F, 2):
        solver.add_clause([-x[r, f1], -x[r, f2]])
    for f in F:
        solver.set_soft(-x[r, f], weight=W[r][f])

solver.solve()
print(f"Cost: {solver.get_cost()} | Model: {solver.get_model()}")
solver.solve(assumptions=[x[2, 1]])
print(f"Cost with R2 on freq 1: {solver.get_cost()}")
solver.set_soft(-x[3, 1], weight=500)
solver.solve()
print(f"Cost after congestion: {solver.get_cost()}")
```

Figure 2 Incremental frequency assignment modelled with Hermax and UWrMaxSAT.

Figure 2 illustrates the implementation of this problem using `UWrMaxSAT`. The script first initializes the solver and allocates the Boolean variables. Hard clauses ensure each router is assigned exactly one frequency and to prevent adjacent routers from sharing the

same one. Soft clauses are introduced in accordance with $W_{r,f}$ to minimize congestion. The incrementality of Hermax is applied after the initial solve: the model is re-evaluated under a temporary assumption (forcing router 2 to use frequency 1) without rebuilding the formula. Finally, the script modifies the weight for router 3 on frequency 1 to 500, simulating a congestion spike, and resolves the model while reusing the solver's learned clauses and state.

4 A MaxSAT Compiler

Despite the impressive performance of modern MaxSAT solvers, the ecosystem has lacked a high-level optimization language that targets exportable CNF and integrates with the plethora of existing solvers. To fix this gap, we introduce a SAT/MaxSAT modelling compiler with CP primitives. Written in pure Python, it provides the ergonomic abstractions of CP while guaranteeing immediate resolution to CNF and WCNF. We commit to the order/ladder encoding [7, 66] for integer variables, which allows the compiler to unlock optimizations unavailable in other SAT compilation pipelines.

Most compiler languages operate by constructing an Abstract Syntax Tree (AST) representing the entire problem, which is later flattened and optimized by a staged compilation pipeline. Our approach discards this paradigm in favor of *eager evaluation*. The compiler data flow can be seen in Figure 3.

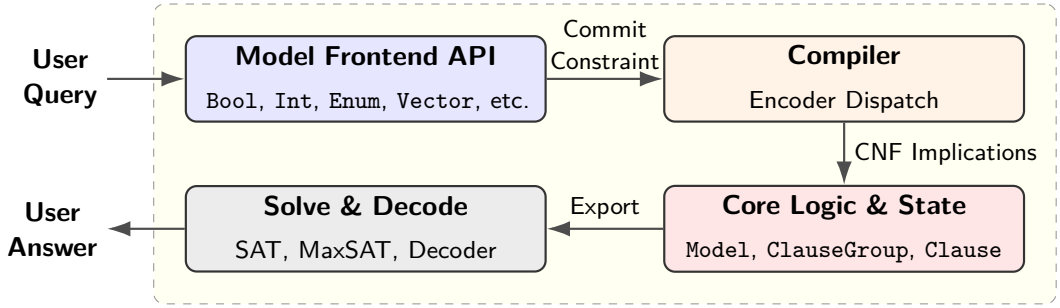

Figure 3 Data flow pipeline for the eager evaluation compiler architecture.

Unlike CP-SAT, constraint expressions are quickly resolved into clauses. Furthermore, our compiler is *CNF oriented*. We deliberately avoid emitting arbitrary logical circuits (e.g. those requiring Tseitin or DNF unrolling). This decision ensures that the compiled query resembles the user-intended model as closely as possible.

4.1 A Constraint Programming Inspired API

The library provides an extensive suite of modelling primitives, including Boolean variables, order encoded integers, optionally nullable categoricals, and interval variables for scheduling. These primitives can be organized into vectors and matrices collections that support NumPy style slicing, flattening, and row/column extraction. At the modelling level, users have access to various constraints, such as extensional tables, cardinalities (e.g., AMO, ALO, exact-K), boundary aggregations (max, min), and optimized `AllDifferent` backends. Additionally, the API supports algebraic operations, including vector multiplexing, dynamic index resolution ($A = V[B]$), floor division ($X//c$), piecewise functions, among many other functions.

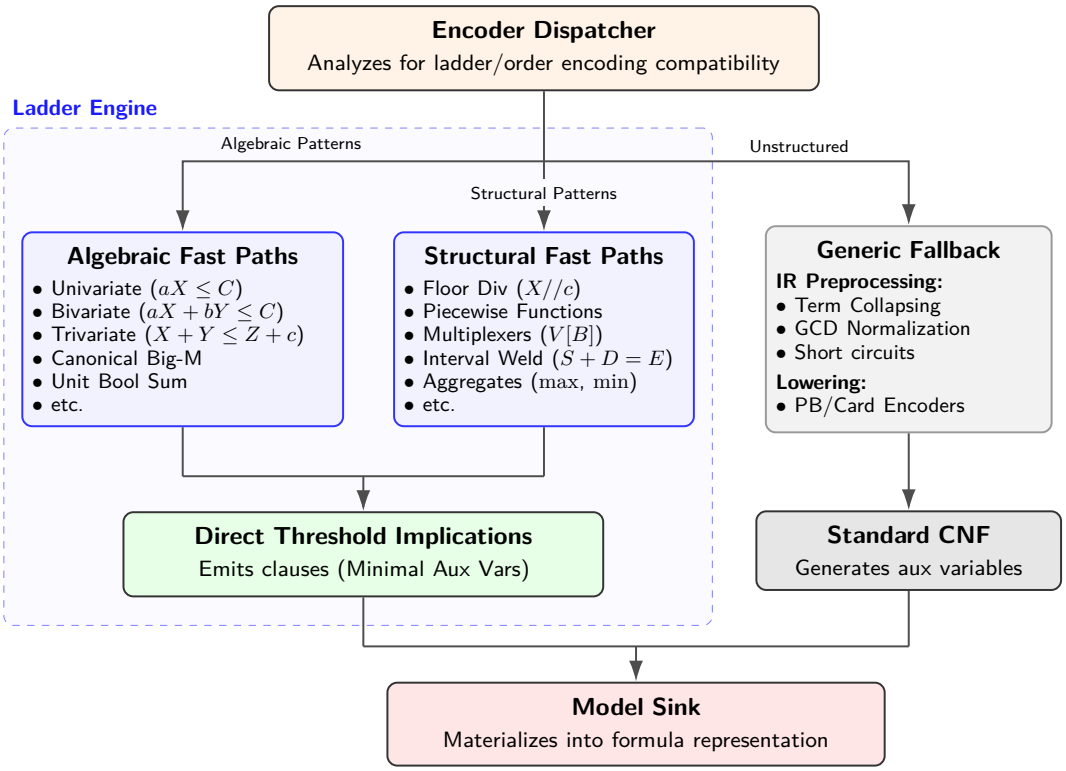

Figure 4 Compiler architecture mapping DSL constructs to optimized CNF representations.

4.2 Compiler Optimizations

As illustrated in Figure 4, the dispatcher can intercept algebraic and structural operations to bypass PB/Card encoders and use clause encodings over threshold literals of the ladder encoding. The dispatcher automatically normalizes constraints into these canonical forms to maximize pattern matching. Key strategies include:

- **Algebraic Fast Paths:** Linear forms such as $aX \leq C$, $aX+bY \leq C$, $X+Y \leq Z+c$, and some Big-M constraints ($aX+w\cdot b \leq C$) as well as bounded Boolean sums ($\sum x_i \leq k+M\cdot y$) are lowered, when possible, into encodings without auxiliary variables, emitting binary or ternary clauses.
- **Structural Operations:** Other high-level constructs also avoid arithmetic translations. Aggregates (max / min) are efficiently synthesized via threshold; array multiplexing ($V[B]$) is unrolled into gated branches, and fixed duration intervals enforce $S + D = E$ also via bitwise threshold. Inequalities ($X \neq Y$ or $|X - Y| \leq D$) exploit the ladder exact value representation to generate exclusion.
- **PB/Card Fallback:** Unstructured constraints use Card/PB encoders and go through an intermediate simplification pass. Term collapsing, constant folding, short circuiting, and GCD normalization [25] are applied. To encode these constraints, Hermax integrates the latest PySAT `card` package, developed bindings to the latest version of `pblib` [58], and reimplemented PB encodings over At-Most-One (PB(AMO)) constraints [19].

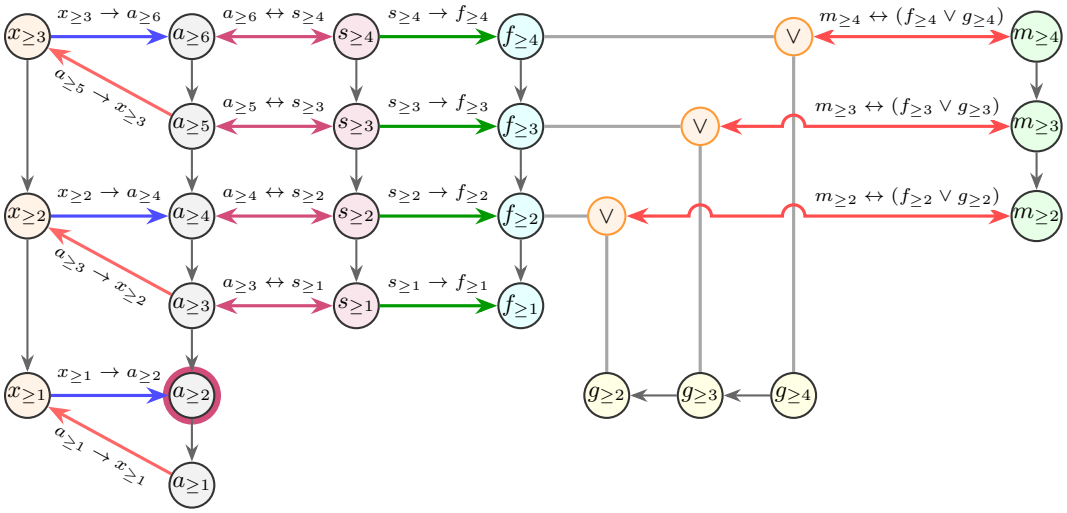

Figure 5 Variable implication graph showing $(2X = A) \wedge (A = S+2) \wedge (S \leq F) \wedge (M = \max(F, G))$. With $X \in [0, 3]; A \in [0, 6]; S \in [0, 4]; F \in [0, 4]; G \in [1, 4]; M \in [1, 4]$.

Typical SAT compilations of affine constraints rely on one-hot encodings (e.g., $\sum_i i \cdot a_{=i} = \sum_j 2j \cdot x_{=j}$), which generate large PBs. Instead, Hermax uses the transitivity of graph ladder encodings to compile these relations directly. Figure 5 illustrates four operations compiled without auxiliary variables:

1. **Scaling ($A = 2X$):** Generates upward implications ($x_{\geq k} \to a_{\geq 2k}$) and bounds domain gaps downward ($a_{\geq 2k-1} \to x_{\geq k}$).
2. **Shifting ($A = S + 2$):** Mapped as an index shift via exact equivalences ($a_{\geq k} \leftrightarrow s_{\geq k-2}$). The double-bordered node $a_{\geq 2}$ marks a forced threshold literal (unit clause).
3. **Precedence ($S \leq F$):** Enforced via monotone implications ($s_{\geq k} \to f_{\geq k}$).
4. **Maximum ($M = \max(F, G)$):** Evaluated across thresholds using binary and ternary clauses ($m_{\geq k} \leftrightarrow f_{\geq k} \vee g_{\geq k}$).

For reference, compiling $A = 2X$ requires 59 variables and 141 clauses in CPMpy (which uses one-hot PB encodings); Hermax compiles the same constraint using zero auxiliary variables, requiring only the 9 original ladder variables and 13 clauses. Consequently, combinatorial models involving interval variables (e.g. scheduling, routing) avoid expansion overhead.

This threshold representation extends to constraints such as `AllDifferent`. Rather than enforcing $\mathcal{O}(N^2)$ pairwise inequalities, Hermax materializes exact value literals ($x_{=k} \leftrightarrow x_{\geq k} \wedge \neg x_{\geq k+1}$) and applies At-Most-One constraints across the domains.

4.3 Model Compiler Example

To illustrate the expressivity of the supported language, we model the Capacitated Vehicle Routing Problem (CVRP). In CVRP, vehicles with uniform capacity must service nodes with specific demands, starting and ending at a central depot, while minimizing the travel distance. We will use the Miller-Tucker-Zemlin (MTZ) formulation [47]. We define a Boolean matrix for edges and a ladder encoded integer vector to track the accumulated load of the vehicle. The objective function must minimize the travel distance. We also demonstrate how distance matrices can be dynamically updated to simulate real-time traffic conditions.

```python
from hermax.model import Model

m = Model()
N, num_vehicles, capacity, demands, distances = ...
edges = m.bool_matrix("edge", N, N)
loads = m.int_vector("load", N, lb=0, ub=capacity)

m &= sum(edges.row(0)) == num_vehicles
m &= sum(edges.col(0)) == num_vehicles
m &= loads[0] == 0

for i in range(N):
  m &= ~edges[i][i]
  if i > 0:
    m &= edges.row(i).exactly_one()
    m &= edges.col(i).exactly_one()
    m &= loads[i] >= demands[i]
  for j in range(1, N):
    if i != j:
      m &= (loads[i] + demands[j] <= loads[j]).only_if(edges[i][j])

def route_cost(distance_matrix):
  return sum(edges[i][j] * distance_matrix[i][j] for i in range(N) for j in range(N))

m.obj = route_cost(distances)
res1 = m.solve()
print(f"Initial Optimal Cost: {res1.cost}")

res2 = m.solve(assumptions=[edges[1][3]])
print(f"Cost forcing route 1->3: {res2.cost}")

distances[0][1] = 999
m.obj = route_cost(distances)
res3 = m.solve()
print(f"Cost after traffic spike: {res3.cost}")
print(f"Is route 0->1 used? {res3[edges[0][1]]}")
```

Figure 6 Implementation of the CVRP using an incremental objective and assumptions.

This example illustrates how Hermax bridges modelling and low-level SAT so users build constraints using idiomatic Python. In the example, `sum()` natively lowers to cardinality constraints, and matrix slices (e.g., `edges.row(i)`) integrate with these encoders. Furthermore, the compiler evaluates arithmetic to minimize the formula size. Unary bounds against constants (`loads[i] >= demands[i]`) are folded at compile time directly into a single literal from the ladder encoding, introducing zero overhead.

For more complex operations, the engine intercepts the expressions to bypass PB encoding. The MTZ constraint $(E_{i,j} \implies (L_i + d_j \leq L_j))$ exemplifies this: instead of falling back on PB, the dispatcher recognizes the bivariate relation and applies gating at the clause level. This lowers the MTZ constraint into a $\mathcal{O}(|L|)$ number of ladder implications with binary clauses, and with ternary clauses introduced for the conditional.

The Hermax compiler also abstracts the state of incremental MaxSAT solving. When the model changes and the objective function is reassigned, the compiler automatically differentiates the new objective expression against the old one. It then issues the delta IPAMIR weight modifications required to synchronize the solver's state. Users can inject

temporary assumptions and query the solver's assignment using the exact same Python objects used during formulation. In terms of modelling of the objective functions, multiple abstractions and strategies are supported:

- **Floating point weights:** Users specify an integer precision parameter, allowing the compiler to automatically scale floating point values into integers.
- **Negative weights:** The compiler applies the equality $-w \cdot x = w \cdot \neg x - w$. It translates a negative penalty on a literal into a positive penalty on its negation, maintaining the negative offset of the reported objective.
- **Lexicographic optimization:** Hermax provides two optimization strategies. By default, it issues multiple consecutive solver calls to iteratively solve and harden each level. Alternatively, it performs an automatic transformation that scales and aggregates multi-level weights into a single objective level for a one-shot call.

5 Hermax Availability

Hermax is available under the Apache 2.0 open source license. Hermax officially supports Windows (with MinGW), Linux, and macOS across both x86 and ARM architectures with precompiled wheels, which required extensive patching of the integrated solvers.

> **Installation:** `pip install hermax`
> **Available on PyPI:** `https://pypi.org/project/hermax/`
> **Documentation:** `https://hermax.readthedocs.io/`

While this library lowers the barrier to entry, the reliability of the system remains dependent on the underlying C++ MaxSAT engines, which may exhibit implementation bugs [56]. This is important to consider on a tool that will be used by non-SAT experts. Our test suite uncovered undefined behaviour that only presented itself on some architectures, which allowed us to patch bugs at the individual solver level that have not manifested themselves on Linux x86.

6 Conclusions & Future Work

Hermax connects high-level Python prototyping with high-performance MaxSAT solving via an IPAMIR interface and an eager CNF/WCNF compiler. Our main contribution is practical unification: users can model with CP primitives, update objectives and assumptions incrementally, and target a broad set of MaxSAT engines from the same Python API.

In terms of solver support, we want to broaden Hermax by integrating modern [35] and historical solvers [6, 29, 39]. We also want to expand queries to multi-objective optimization and solution enumeration, potentially using backends like RustSAT.

As for the compiler, we want to work on evaluating the trade-off between full clause materialization and the high-level reasoning of native CP solvers, and bring insight into when each approach is most effective. To support this, we plan to explore alternative encoding strategies [1] and implement MaxSAT-specific preprocessing techniques [14] before solving.

We also want to explore portfolio strategies across different supported solvers. Specifically, making use of new 2026 MaxSAT Evaluation lower-bound reporting while solving to merge the capabilities of complete and incomplete solvers in a single call.

References

1 Ignasi Abío, Valentin Mayer-Eichberger, and Peter J Stuckey. Encoding linear constraints with implication chains to cnf. In *Proc. of the Int. Conf. on Princ. and Pract. of Constraint Programming (CP)*, pages 3–11, 2015. doi:10.1007/978-3-319-23219-5_1.

2 Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. doi:10.1007/S12532-008-0001-1.

3 Josep Alos, Carlos Ansótegui, Josep M Salvia, and Eduard Torres. Optilog v2: model, solve, tune and run. In *25th Proc. of the Int. Conf. on Theory and Appl. of Satisfiability Testing (SAT)*, pages 25–1, 2022. doi:10.4230/LIPIcs.SAT.2022.25.

4 Sahel Alouneh, Sa'ed Abed, Mohammad H Al Shayeji, and Raed Mesleh. A comprehensive study and analysis on sat-solvers: advances, usages and achievements. *Artificial Intelligence Review*, 52(4):2575–2601, 2019. doi:10.1007/S10462-018-9628-0.

5 Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *Tech. Commun. of the 28th Int. Conf. on Logic Programming (ICLP)*, pages 212–221, 2012. doi:10.4230/LIPIcs.ICLP.2012.211.

6 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Sat-based maxsat algorithms. *Artificial Intelligence*, 196:77–105, 2013. doi:10.1016/J.ARTINT.2013.01.002.

7 Carlos Ansótegui and Felip Manya. Mapping problems with finite-domain variables to problems with boolean variables. In *Proc. of the Int. Conf. on Theory and Appl. of Satisfiability Testing (SAT)*, pages 1–15, 2004. doi:10.1007/11527695_1.

8 Carlos Ansótegui, Jesus Ojeda, António Pacheco, Josep Pon, Josep M Salvia, and Eduard Torres. Optilog: A framework for sat-based systems. In *Proc. of the Int. Conf. on Theory and Appl. of Satisfiability Testing (SAT)*, pages 1–10, 2021. doi:10.1007/978-3-030-80223-3_1.

9 Josep Argelich, Chu-Min Li, Felip Manya, and Jordi Planes. The first and second max-sat evaluations. *Journal on Satisfiability*, 4(2-4):251–278, 2008. doi:10.3233/SAT190047.

10 Gilles Audemard and Laurent Simon. On the glucose sat solver. *International Journal on Artificial Intelligence Tools*, 27(01):1840001, 2018. doi:10.1142/S0218213018400018.

11 Florent Avellaneda. Evalmaxsat 2023. *MaxSAT Evaluation*, 2023:12, 2023.

12 Tomáš Balyo, Armin Biere, Markus Iser, Carsten Sinz, and Allen Van Gelder. Sat race 2015. *Artificial Intelligence*, 241:45–65, 2016. doi:10.1016/J.ARTINT.2016.08.007.

13 Haniel Barbosa, Clark W. Barrett, et al. cvc5: A versatile and industrial-strength SMT solver. In *Proc. of the 28th Int. Conf. on TACAS*, volume 13243 of *LNS in Computer Science*, pages 415–442, 2022. doi:10.1007/978-3-030-99524-9_24.

14 Anton Belov, António Morgado, and Joao Marques-Silva. Sat-based preprocessing for maxsat. In *Proc. of the Int. Conf. on Logic for Program., Art. Intell. and Reason. (LPAR)*, pages 96–111, 2013. doi:10.1007/978-3-642-45221-5_7.

15 Jeremias Berg, Emir Demirović, and Peter J Stuckey. Core-boosted linear search for incomplete maxsat. In *Proc. of the Int. Conf. on Integr. of Constraint Program., Art. Intell., and Oper. Res. (CPAIOR)*, pages 39–56, 2019. doi:10.1007/978-3-030-19212-9_3.

16 Jeremias Berg, Matti Järvisalo, Ruben Martins, Andreas Niskanen, and Tobias Paxian. Maxsat evaluation 2024: Solver and benchmark descriptions, 2024.

17 Armin Biere, Tobias Faller, Katalin Fazekas, et al. Cadical 2.0. In *International Conference on Computer Aided Verification*, pages 133–152, 2024. doi:10.1007/978-3-031-65627-9_7.

18 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Maxsat, hard and soft constraints. *11.12. 1. Autarkies and qualitative matrix analysis*, page 613, 2009. doi:10.3233/978-1-58603-929-5-613.

19 Miquel Bofill, Jordi Coll, Peter Nightingale, Josep Suy, Felix Ulrich-Oltean, and Mateu Villaret. Sat encodings for pseudo-boolean constraints together with at-most-one constraints. *Artificial Intelligence*, 302:103604, 2022. doi:10.1016/J.ARTINT.2021.103604.

20 Jordi Coll, Chu-Min Li, Shuolin Li, Djamal Habet, and Felip Manyà. Solving weighted maximum satisfiability with branch and bound and clause learning. *Computers & Operations Research*, page 107195, 2025. doi:10.1016/J.COR.2025.107195.

21 Jessica Davies and Fahiem Bacchus. Solving maxsat by solving a sequence of simpler sat instances. In *Proc. of the Int. Conf. on Princ. and Pract. of Constraint Programming (CP)*, pages 225–239, 2011. doi:10.1007/978-3-642-23786-7_19.

22 Jessica Davies and Fahiem Bacchus. Exploiting the power of mip solvers in maxsat. In *Proc. of the Int. Conf. on Theory and Appl. of Satisfiability Testing (SAT)*, pages 166–181, 2013. doi:10.1007/978-3-642-39071-5_13.

23 Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proc. of the Int. Conf. on Tools and Algorith. for the Constr. and Anal. of Syst. (TACAS)*, pages 337–340, 2008. doi:10.1007/978-3-540-78800-3_24.

24 Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016. URL: https://jmlr.org/papers/v17/15-408.html.

25 Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modelling and Computation*, 2(1-4):1–26, 2006. doi:10.3233/SAT190014.

26 Johannes K Fichte, Daniel Le Berre, Markus Hecher, and Stefan Szeider. The silent (r) evolution of sat. *Communications of the ACM*, 66(6):64–72, 2023. doi:10.1145/3560469.

27 Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proc. of the Workshop on Constraint Model. and Reformulation (ModRef)*, volume 19, 2019.

28 William E Hart, Carl D Laird, Jean-Paul Watson, David L Woodruff, et al. *Pyomo-optimization modeling in python*, volume 67. Springer, 2017.

29 Federico Heras, Javier Larrosa, and Albert Oliveras. Minimaxsat: An efficient weighted max-sat solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008. doi:10.1613/JAIR.2347.

30 Hao Hu, Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. Learning optimal decision trees with maxsat and its integration in adaboost. In *Proc. of the 29th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1170–1176, 2020. doi:10.24963/IJCAI.2020/163.

31 QHJAJ Huangfu and JA Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. doi:10.1007/S12532-017-0130-5.

32 Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. Pysat: A python toolkit for prototyping with sat oracles. In *Proc. of the Int. Conf. on Theory and Appl. of Satisfiability Testing (SAT)*, pages 428–437, 2018. doi:10.1007/978-3-319-94144-8_26.

33 Alexey Ignatiev, António Morgado, and João Marques-Silva. Rc2: an efficient maxsat solver. *Journal on Satisfiability, Boolean Modelling and Computation*, 11(1):53–64, 2019. doi:10.3233/SAT190116.

34 Hannes Ihalainen, Jeremias Berg, et al. Refined core relaxation for core-guided maxsat solving. In *27th Proc. of the Int. Conf. on Princ. and Pract. of Constraint Programming (CP)*, pages 28–1, 2021. doi:10.4230/LIPIcs.CP.2021.28.

35 Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. Unifying sat-based approaches to maximum satisfiability solving. *Journal of Artificial Intelligence Research*, 80:931–976, 2024. doi:10.1613/JAIR.1.15986.

36 J. S. Roy, Stuart A. Mitchell, and PuLP contributors. PuLP, 2025. Python LP and MILP modeling library.

37 Christoph Jabs. RustSAT: A library for SAT solving in rust. In *28th Proc. of the Int. Conf. on Theory and Appl. of Satisfiability Testing (SAT)*, volume 341, pages 15:1–15:13, 2025. doi:10.4230/LIPIcs.SAT.2025.15.

38 Christoph Jabs, Jeremias Berg, Andreas Niskanen, and Matti Järvisalo. Maxsat-based bi-objective boolean optimization. In *Proc. of the Int. Conf. on Theory and Applications of Satisfiability Testing (SAT)*, pages 12:1–12:18, 2022. doi:10.4230/LIPIcs.SAT.2022.12.

39 Mikoláš Janota. Mifumax—a literate maxsat solver: System description. *Journal on Satisfiability, Boolean Modelling and Computation*, 9(1):83–88, 2014. doi:10.3233/SAT190103.

40 Mikoláš Janota, António Morgado, José Fragoso Santos, and Vasco Manquinho. The seesaw algorithm: Function optimization using implicit hitting sets. In *Proc. of the Int. Conf. on Princ. and Pract. of Constraint Programming (CP)*, pages 31:1–31:16, 2021. doi:10.4230/LIPIcs.CP.2021.31.

41 Menghua Jiang and Yin Chen. NuWLS-c-IBR: solver description. *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions*, page 22, 2024.

42 Saurabh Joshi, Prateek Kumar, Sukrut Rao, and Ruben Martins. Open-wbo-inc: Approximation strategies for incomplete weighted maxsat. *Journal on Satisfiability, Boolean Modelling and Computation*, 11(1):73–97, 2019. doi:10.3233/SAT190118.

43 Dimitri Justeau-Allaire and Charles Prud'Homme. pychoco: all-inclusive python bindings for the choco-solver constraint programming library. *Journal of Open Source Software*, 10(113):8847, 2025.

44 Stephen Maher, Matthias Miltenberger, et al. PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In *Mathematical Software*, pages 301–307. Springer, 2016. doi:10.1007/978-3-319-42432-3_37.

45 Joao Marques-Silva and Jordi Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Proc. of the Conf. on Design, Autom. and Test in Eur.*, pages 408–413, 2008. doi:10.1109/DATE.2008.4484715.

46 Ruben Martins, Vasco Manquinho, et al. Open-wbo: A modular maxsat solver. In *Proc. of the Int. Conf. on Theory and Appl. of Satisfiability Testing (SAT)*, pages 438–445, 2014. doi:10.1007/978-3-319-09284-3_33.

47 Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960. doi:10.1145/321043.321046.

48 António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided maxsat with soft cardinality constraints. In *Proc. of the Int. Conf. on Princ. and Pract. of Constraint Programming (CP)*, pages 564–573, 2014. doi:10.1007/978-3-319-10428-7_41.

49 Alexander Nadel. Tt-open-wbo-inc: an efficient anytime maxsat solver. *Journal on Satisfiability, Boolean Modelling and Computation*, 15(1):1–7, 2024. doi:10.3233/SAT-231504.

50 Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *Proc. of the Int. Conf. on Princ. and Pract. of Constraint Programming (CP)*, pages 529–543, 2007. doi:10.1007/978-3-540-74970-7_38.

51 Andreas Niskanen, Jeremias Berg, and Matti Järvisalo. Enabling incrementality in the implicit hitting set approach to maxsat under changing weights. In *27th Proc. of the Int. Conf. on Princ. and Pract. of Constraint Programming (CP)*, pages 44–1, 2021. doi:10.4230/LIPIcs.CP.2021.44.

52 Andreas Niskanen, Jeremias Berg, and Matti Järvisalo. Incremental maximum satisfiability. In *25th Proc. of the Int. Conf. on Theory and Appl. of Satisfiability Testing (SAT)*, pages 14–1, 2022. doi:10.4230/LIPIcs.SAT.2022.14.

53 Chanseok Oh. COMiniSatPS Pulsar and GHackCOMSPS. In *Proceedings of SAT Competition 2017*, volume B-2017-1, pages 12–13. University of Helsinki, 2017.

54 David Orden, José Manuel Giménez, Ivan Marsa-Maestre, and Enrique de la Hoz. Spectrum graph coloring and applications to wi-fi channel assignment. *Symmetry*, 10(3):65, 2018. doi:10.3390/SYM10030065.

55 Shiwei Pan, Yiyuan Wang, et al. An efficient core-guided solver for weighted partial maxsat. In *Proc. of the Int. Joint Conf. on Art. Intell. (IJCAI)*, pages 2647–2656, 2025. doi:10.24963/IJCAI.2025/295.

56 Tobias Paxian and Armin Biere. Maxsat fuzzing and delta debugging. *Journal of Artificial Intelligence Research*, 85, 2026. doi:10.1613/JAIR.1.19716.

57 Laurent Perron, Frédéric Didier, and Steven Gay. The cp-sat-lp solver (invited talk). In *Proc. of the Int. Conf. on Princ. and Pract. of Constraint Programming (CP)*, pages 3–1, 2023. doi:10.4230/LIPIcs.CP.2023.3.

58 Tobias Philipp and Peter Steinke. Pblib – a library for encoding pseudo-boolean constraints into cnf. In *Proc. of the Int. Conf. on Theory and Appl. of Satisfiability Testing (SAT)*, pages 9–16, 2015. doi:10.1007/978-3-319-24318-4_2.

59 Marek Piotrów. Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems. In *Proc. of the 32nd Int. Conf. on Tools with Art. Intell. (ICTAI)*, pages 132–136. IEEE, 2020. doi:10.1109/ICTAI50040.2020.00031.

60 Matthew J Saltzman. Coin-or: an open-source library for optimization. In *Programming languages and systems in computational economics and finance*, pages 3–32. Springer, 2002.

61 Josep Maria Salvia Hornos, Cèsar Fernández Camón, and Carles Mateu Piñol. Hermax: MaxSAT Optimization for Python Documentation. Text (visited on 2026-06-29). URL: https://hermax.readthedocs.io/, doi:10.4230/artifacts.26925.

62 Josep Maria Salvia Hornos, Cèsar Fernández Camón, and Carles Mateu Piñol. Hermax: MaxSAT Optimization for Python Pypi Distribution. Software (visited on 2026-06-29). URL: https://pypi.org/project/hermax, doi:10.4230/artifacts.26924.

63 Josep Maria Salvia Hornos, Cèsar Fernández Camón, and Carles Mateu Piñol. josalhor/hermax. Software (visited on 2026-05-26). URL: https://github.com/josalhor/hermax, doi:10.4230/artifacts.25966.

64 Josep Maria Salvia Hornos, Cèsar Fernández Camón, Carles Mateu Piñol, Joao Marques-Silva, Ivet Rafegas Fonoll, and Josep Cirera Balcells. Modelling and optimising switching on a physical binary sorter. *Annals of Operations Research*, pages 1–22, 2026. doi:10.1007/S10479-025-06903-2.

65 Tobias Seufert, Felix Winterer, et al. Everything you always wanted to know about generalization of proof obligations in bit-level pdr. In *MBMV. Workshop*, pages 144–147, 2024. doi:10.1109/TCAD.2022.3198260.

66 Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear csp into sat. *Constraints*, 14(2):254–272, 2009. doi:10.1007/S10601-008-9061-0.

67 Johannes P Wallner, Andreas Niskanen, and Matti Järvisalo. Complexity results and algorithms for extension enforcement in abstract argumentation. *Journal of Artificial Intelligence Research*, 60:1–40, 2017. doi:10.1613/JAIR.5415.

68 Jiongzhi Zheng, Zhuo Chen, Chu-Min Li, and Kun He. Rethinking the soft conflict pseudo boolean constraint on maxsat local search solvers. In *Proc. of the Int. Joint Conf. on Art. Intell. (IJCAI)*, 2024. URL: https://www.ijcai.org/proceedings/2024/220.