MILP Alternatives to MaxSAT for Synthesizing Pareto-Optimal Interpretations of Black-Box Models

Aryan Katiyar
Department of Computer Science
Indian Institute of Technology, Bombay
22b1205@iitb.ac.in

Himanshu Gangwal
Department of Computer Science
Indian Institute of Technology, Bombay
himanshu16gangwal@iitb.ac.in

Code Repository: The code used in this work is available at CODE

1 Introduction

Interpretable machine learning aims to provide human-understandable models that capture the behaviour of complex black-box predictors. The framework proposed by [2] formulates this as the problem of synthesizing a bounded decision diagram that simultaneously maximizes two quantities: the correctness score $\Delta_{\mathcal{C}}$ (agreement with the black-box) and the explainability score $\Delta_{\mathcal{E}}$ (favouring compact structures and preferred predicates). Their approach encodes the synthesis of a single interpretation as a weighted MaxSAT instance

$$\varphi_{E,S,\Delta_{\mathcal{C}},\Delta_{\mathcal{E}}} \;=\; \varphi_E \wedge \varphi_S \wedge \varphi_{\Delta_{\mathcal{C}}} \wedge \varphi_{\Delta_{\mathcal{E}}},$$

and repeatedly solves variations of this instance to enumerate the full Pareto frontier of explanations.

While this method provides strong guarantees and rich interpretability trade-offs, the cost of solving a single MaxSAT instance is substantial, especially when only the *first* Pareto point—maximizing $\Delta_{\mathcal{C}} + \Delta_{\mathcal{E}}$ is required. This first solution already gives a high-quality interpretation combining both fidelity and simplicity, and is often sufficient in interactive or exploratory workflows. However, obtaining it still requires solving the full weighted MaxSAT encoding.

The goal of this project is to explore alternatives to the original MaxSAT encoding that reduce the time needed to obtain this first interpretation. Rather than altering the decision-diagram template or the semantics of the problem, we preserve the original formulation and focus on back-end optimization encodings. Specifically, we:

- Reproduce the original MaxSAT encoding and analyse structural behaviours such as predicate repetition and the collapse to single-node diagrams under the standard weighting scheme.
- Investigate modified explainability weights and structural constraints to prevent redundant nodes and restore meaningful model-size trade-offs.
- Develop and evaluate several alternative MILP/IP formulations for the same synthesis problem, including a naive slack formulation, an indicator-based (extended) formulation, a pseudo-Boolean strengthened formulation, and a quadratic formulation linearized via McCormick envelopes.

2 Overview of the MaxSAT Encoding

This section summarises the MaxSAT encoding used in the original framework, up to and including the explainability block $\varphi_{\Delta_\mathcal{E}}$. We also introduce the notation used throughout the report.

2.1 Template and Notation

We assume:

- a fixed number of internal nodes $N = \{1, \ldots, k\}$ (node 1 is the root);
- a set of leaves L corresponding to output labels;
- a set of discretized predicates P, each predicate p spawning a finite set of branch indices $B(p)$;
- a finite set of samples S, each with an input and the black-box output label.

The MaxSAT encoding introduces the following Boolean variables:

Predicate assignment. For each node i and predicate p:

$$\lambda_{i,p} = \begin{cases} 1 & \text{if node } i \text{ tests predicate } p, \\ 0 & \text{otherwise.} \end{cases}$$

Transitions. For each internal node i, branch index c, and successor j:

$$\tau_{i,c,j} = 1 \text{ iff node } i \text{ has an outgoing edge on branch } c \text{ to successor } j.$$

Sample matching. For each node or leaf i and sample s:

$$m_{i,s} = \begin{cases} 1 & \text{if sample } s \text{ reaches node/leaf } i \text{ following the diagram,} \\ 0 & \text{otherwise.} \end{cases}$$

Node reachability. Each node has a usage indicator:

$$u_i = \begin{cases} 1 & \text{if node } i \text{ is reachable from the root,} \\ 0 & \text{otherwise.} \end{cases}$$

With $u_1 = 1$, reachability is enforced via active transitions.

Predicate usage. Define

$$\lambda'_{i,p} \;\leftrightarrow\; (u_i \wedge \lambda_{i,p}),$$

so that predicates are rewarded only when placed on reachable nodes.

2.2 Hard Constraints: φ_E and φ_S

The hard clauses consist of **structural constraints** φ_E (enforcing a well-formed decision diagram: one predicate per node, valid transitions, and acyclicity) and **sample-consistency constraints** φ_S (propagating sample behaviour through the diagram). All clauses in these two blocks are hard.

2.3 Soft Clauses for Correctness: φ_{Δ_C}

Correctness is encoded using soft unit clauses ($m_{1,s}$) for each sample s, rewarding agreements with the black-box at the root.

2.4 Soft Clauses for Explainability: $\varphi_{\Delta_\mathcal{E}}$

Explainability is encoded via soft clauses rewarding predicate usage ($\lambda'_{i,p}$) with weight $w(p)$ and unused-node indicators (u_i) with weight W_{unused}.

In the original system, W_{unused} is set to be strictly larger than any predicate weight, ensuring that adding a node is always penalized unless it significantly improves correctness.

3 Datasets

We evaluate our approach on seven benchmark datasets commonly used in machine learning, program synthesis, and explainable AI research. Each dataset is discretized into a small number of interpretable numerical buckets, as shown below.

- **Adult Income** : Predict whether a person earns >50K per year based on demographic and employment features. After preprocessing, 3 attributes are used (originally 14), with discretizations: ⟨age: 4 buckets, hours_per_week: 4 buckets, workclass: 6 categories⟩. UCI Adult Dataset
- **Banknote Authentication**: Image statistics from genuine vs. forged banknotes. We use 3 features (originally 4), each bucketed into 4 intervals: ⟨variance: 4, skewness: 4, curtosis: 4⟩. UCI Banknote Dataset
- **California Housing Prices** : Tabular regression dataset predicting median house prices. We reduce to 2 key features (originally 10), with: ⟨population: 4:1 (fine:coarse merging), median_income: 2:1 discretization⟩. California Housing Dataset
- **AutoTaxi Simulator** : A synthetic reinforcement-learning environment simulating autonomous taxi navigation decisions. We use 3 state variables discretized as: ⟨clouds: 6, day_time: 4, init_pos: 4⟩. ICML AutoTaxi benchmark
- **Iris** : Classic classification dataset distinguishing three flower species using morphological measurements. We use 3 out of the original 4 attributes, each discretized into 4 buckets: ⟨sepal_length: 4, petal_length: 4, petal_width: 4⟩. UCI Iris Dataset
- **Loan Acquisition**: A financial-risk dataset predicting user loan behavior. We use 4 out of 6 attributes: ⟨age: 4, monthly_income: 2, dependents: 2, credit_score: 3⟩. Loan Acquisition Dataset
- **Theorem Prover Dataset**: Symbolic-reasoning dataset used for predicting outcomes in automated theorem proving tasks. We use 2 features: ⟨F1: 4, F10: 3⟩ out of the 14 present. Automated Theorem Proving Dataset

port

4 Repetition Constraints

In this section, we explore the repetition problem that emerges in our encoding approach. We then outline the strategy adopted in the paper to resolve this issue and analyze the side effects and trade-offs resulting from that solution.

4.1 Analysis of the Original Implementation

The core goal of the synthesis framework is to generate Decision Diagrams (DD) that are both accurate (Correctness $\Delta_{\mathcal{C}}$) and interpretable (Explainability $\Delta_{\mathcal{E}}$). Ideally, the explainability score is a weighted sum of predicate preference and compactness:

$$\Delta_{\mathcal{E}} = \sum_{p,i} w(p) \cdot \lambda'_{i,p} + \sum_{i} W_{\text{unused}} \cdot \neg u_i \tag{1}$$

The terms in this objective are formally defined based on the structural constraints of the decision diagram:

- **Node Usage (u_i):** A node i is considered "used" if it is reachable from the root. The root node (index 1) is always used ($u_1 = 1$), while subsequent nodes are used if there is a transition from an active parent node:

$$u_1 = 1, \quad \text{and for } i > 1, \quad u_i \leftrightarrow \bigvee_{j<i} \bigvee_{1 \le c \le c_{\max}} (\tau_{j,c,i} \wedge u_j) \tag{2}$$

  where $\tau_{j,c,i}$ represents a transition from node j to node i on condition c.

- **Augmented Predicate ($\lambda'_{i,p}$):** To ensure rewards are only given for predicates in the active part of the graph, $\lambda'_{i,p}$ is true only if node i is used (u_i) and assigned predicate p ($\lambda_{i,p}$):

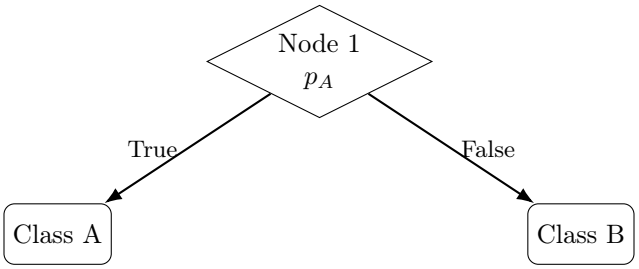
$$\lambda'_{i,p} \leftrightarrow u_i \wedge \lambda_{i,p} \tag{3}$$

4.1.1 Motivation: The Redundancy Loophole

Without specific safeguards, this formulation creates a perverse incentive. If the reward for using a preferred predicate, $w(p)$, is higher than the penalty for using an extra node (approximated by W_{unused}), the solver will repeat the same predicate multiple times along a path to inflate the score.

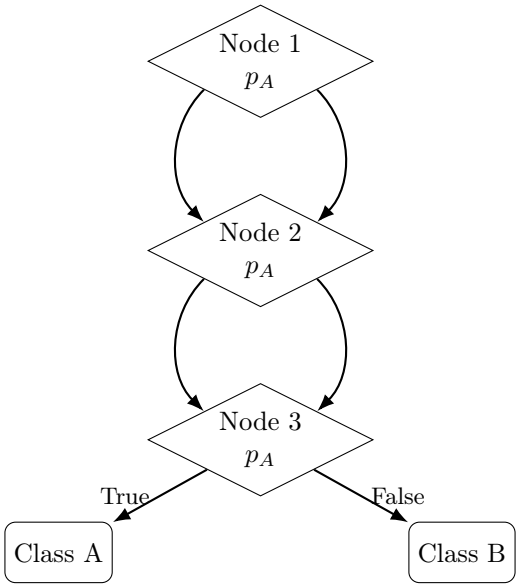
*Example: Artificially Constructing Redundancy Consider a classification on attribute **Age** where the predicate p_A ("Is Age < 20?") has a high preference weight $w(p_A) = 11$, and the reward for saving a node is low $W_{\text{unused}} = 10$. Max nodes $k = 5$.

Classifier 1 (The Compact Optimum): A single node performs the split.



Score: $(1 \times 11) + (4 \text{ unused} \times 10) = \mathbf{51}$.

Classifier 2 (The Redundant Flaw): Nodes 1 and 2 check p_A but pass all traffic forward. Node 3 performs the split.



Score: $(3 \times 11) + (2 \text{ unused} \times 10) = \mathbf{53}$.

Because $53 > 51$, the solver would incorrectly choose the redundant Classifier 2 and superficially boost the explainability score.

4.1.2 The Implemented Solution: Aggressive Weight Scaling

To prevent the issue described above without implementing computationally expensive hard constraints, the paper (and code) enforces a strict weight hierarchy:

$$W_{\text{unused}} = \max_{p}(w(p)) + 1 \tag{4}$$

By setting the reward for *not* using a node higher than the maximum possible reward for using a predicate, the "net profit" of adding a node ($w(p) - W_{\text{unused}}$) becomes negative (-1). This strictly penalizes depth and mathematically guarantees that Classifier 2 (score $300 - 30 = 270$) would lose to Classifier 1 (score $100 - 10 = 90$ relative penalty), thus "solving" the redundancy.

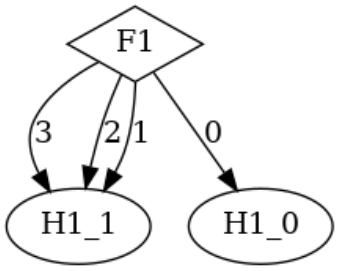
4.2 Critique: The Trivialization of Explainability

While the weight scaling described above successfully prevents redundancy, it introduces a severe side effect: Single Node Degeneracy.

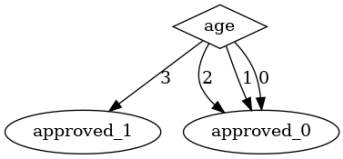
4.2.1 The Single-Node Degeneracy

The size penalty is so dominant that the reward for leaving nodes unused typically exceeds the accuracy improvements gained by introducing additional nodes. Consequently, the solver is mathematically driven to favor the smallest tree even if it has a low accuracy. This phenomenon is

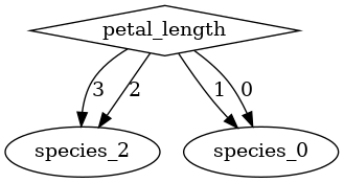
especially pronounced when feature weights are small (e.g., all equal to 1, resulting in an unused-node reward of 2). Empirically, this leads the solver to produce trees consisting of a single decision node (depth-1) in nearly all cases.



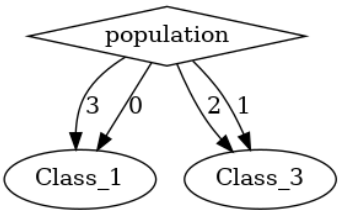
(a) Theorem Prover



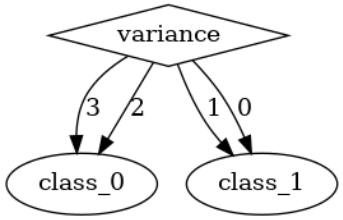
(b) Loan Acquisition



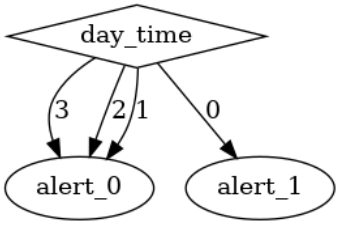
(c) Iris



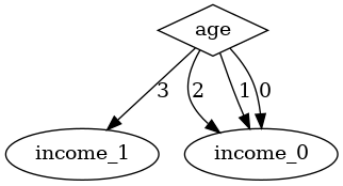
(d) California Housing



(e) Banknote



(f) Autotaxi



(g) Adult Income

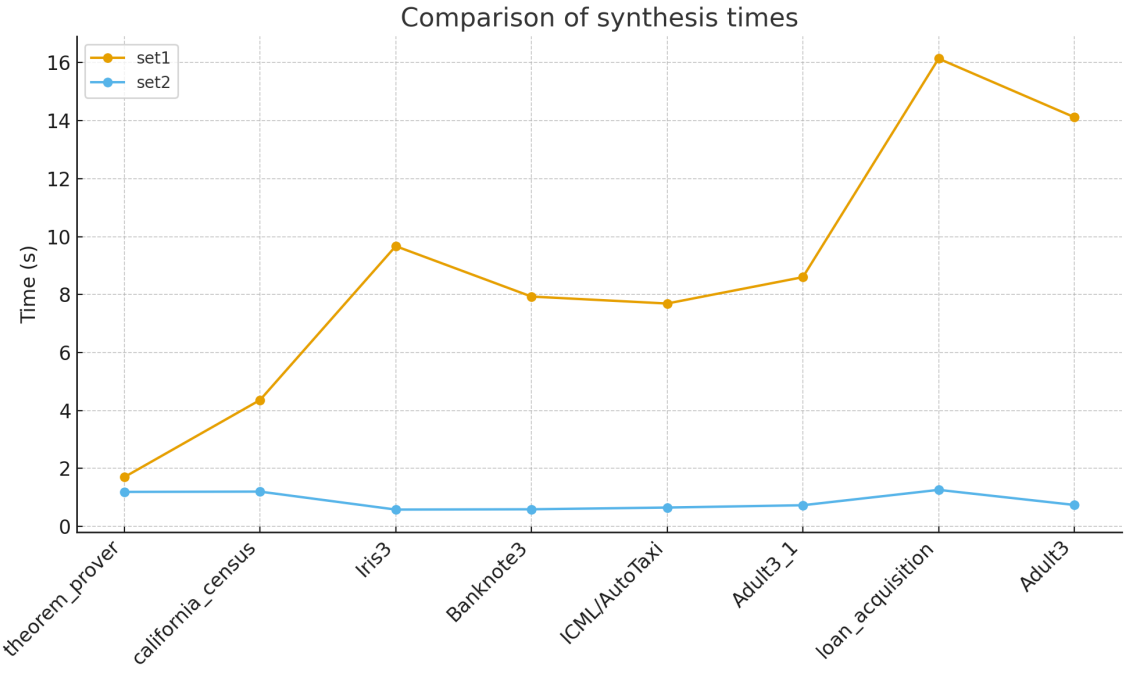
Figure 1: These plots show only the nodes for which $u_i = 1$ (used nodes). We can clearly see that this formulation typically drives the solver toward the Pareto point that employs only a single node, even though up to three nodes were allowed.

4.2.2 Inefficiency of MaxSAT for Trivial Solutions

If the Pareto-optimal solution is constrained to be a single node, the use of a sophisticated MaxSAT solver is computationally inefficient.

- **Observation:** If we only want the best single-node split, we can simply iterate through every feature $f \in P$, calculate its information gain or accuracy, and pick the feature which gives the maximum value of $\mathbf{e}+\mathbf{c}$ .This takes linear time $O(|P| \cdot |Data|)$. This is clearly evident in Figure 2.
- **Conclusion:** Using an NP-hard MaxSAT solver to find a solution that could be found via simple iteration is a suboptimal method. The power of MaxSAT lies in finding complex

combinations of features, which this weighting scheme suppresses.



set1	1.7	4.36	9.67	7.93	7.69	8.6	16.14	14.12
set2	1.19	1.2	0.58	0.59	0.65	0.73	1.26	0.74

Figure 2: set1 corresponds to the default max_weight+1 encoding being used for unused nodes with maxsat; set2 is just iteration over all features to find the best single feature to split on

4.3 Proposed Solution: Hard Reachability Constraints

To allow the solver to explore deeper, more meaningful tree structures without succumbing to redundancy, we must enforce structural integrity via logical constraints rather than weight penalties.

4.3.1 Formal Definition

We introduce a hard constraint $\Phi_{\text{Non-Repeat}}$ that explicitly forbids repeating a predicate on any path. This requires defining *transitive reachability* ($R_{i,j}$):

1. **Reachability ($R_{i,j}$):** True if path $i \to \cdots \to j$ exists.

$$R_{i,j} \iff \left(\bigvee_{c} \tau_{i,c,j} \right) \vee \left(\bigvee_{k} (R_{i,k} \wedge R_{k,j}) \right)$$

2. **Non-Repetition ($\Phi_{\text{Non-Repeat}}$):**

$$\bigwedge_{p \in P} \bigwedge_{i<j} \left((\lambda'_{i,p} \wedge R_{i,j}) \implies \neg \lambda'_{j,p} \right)$$

where the definition of $\lambda'_{i,p}$ can be seen in equation 3

4.3.2 Restoring the Weight Balance

With $\Phi_{\text{Non-Repeat}}$ ensuring structural validity, we no longer need the aggressive penalty. We can set the unused node weight to a balanced value, such as the mean of the predicate weights:

$$W_{\text{unused}} = \frac{1}{|P|} \sum_{p \in P} w(p) \tag{5}$$

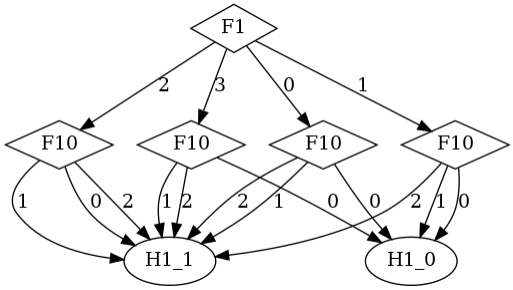
This allows the solver to grow the tree (sacrificing W_{unused}) if a deeper node uses a highly preferred predicate (high $w(p)$), effectively exploring the true trade-off between interpretability and size which can be clearly seen in 4. However doing this method also increased the runnig time

Dataset	**Default method [s]**	**Reachability constraint [s]**
theorem_prover	1.70	1.71
california_census	4.36	4.35
Iris3	9.67	9.47
Banknote3	8.06	17.11
ICML/AutoTaxi	7.69	600+(timeout)
Adult3_1	8.60	46.25
loan_acquisition	16.63	207.83

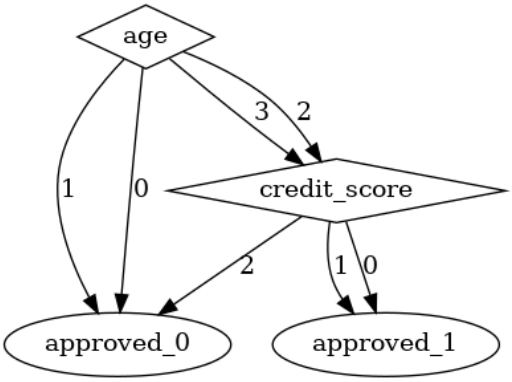
Table 1: Running time for maxsat in the default scenario and after introducing the reachability constraint. The increased computation time on the introduction of reachability constraints is more evident in the case of larger data sets, which have more features. This suggests an increase in computational complexity.

Dataset	**Default method [s]**	**Reachability constraint [s]**
theorem_prover	2.10	10.53
california_census	4.40	4.60
Iris3	9.77	10.67
Banknote3	8.30	10.58
ICML/AutoTaxi	8.55	26.18
Adult3_1	9.40	12.21
loan_acquisition	16.87	193.87

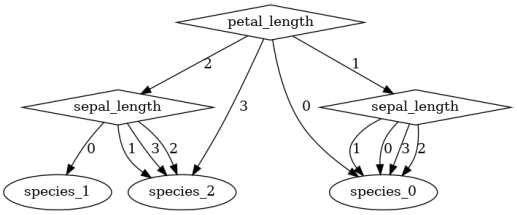
Table 2: Running time for Naive MILP in the default scenario and after introducing the reachability constraint. The increased computation time on the introduction of reachability constraints is more evident in the case of larger data sets, alsthough it is significantly less than that in the case of the default method.



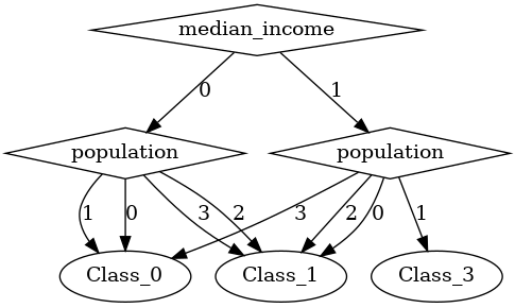
(a) Theorem Prover



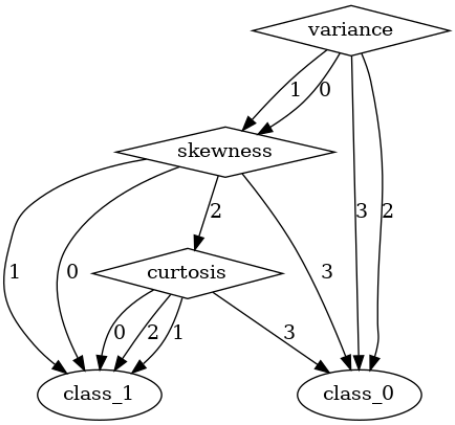
(b) Loan Acquisition



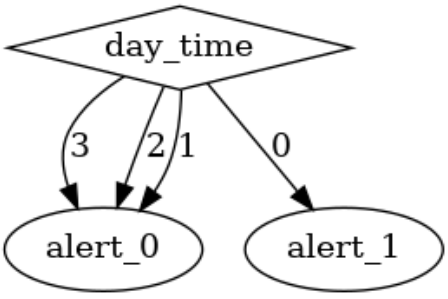
(c) Iris



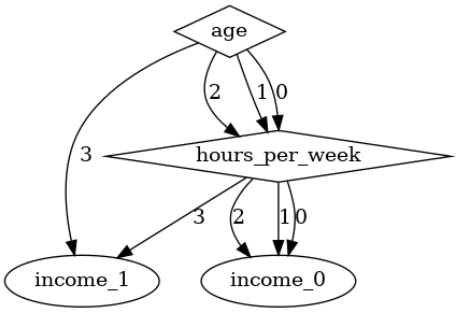
(d) California Housing



(e) Banknote



(f) Autotaxi



(g) Adult Income

Figure 3: Now we can clearly see that almost all methods use more nodes than before, except Autotaxi. However this methods also introduces some redundancy as seen in Figure 3c

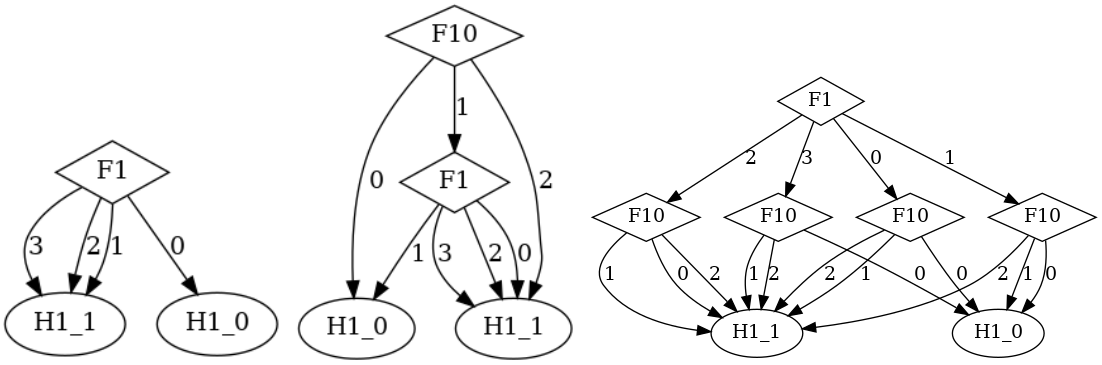
4.3.3 Shortcomings of this method

However, this alternative weighting scheme is not without its own pitfalls. As shown in Figure 3c, reducing the unused–node reward from max +1 to the mean feature weight can introduce other type of structural redundancy. In the illustrated decision diagram, the right `sepal_length` node is redundant: regardless of the bucket value, every branch ultimately leads to `species_0`. Although this node does not improve classification accuracy, it is still included because using it yields a small but positive explainability reward(as the weight for this feature is higher than the mean feature weight). Thus, while the mean-based weighting prevents the collapse to a single node, it may still incentivize the creation of unnecessary nodes.

A more fundamental issue with mean-based weighting is that it implicitly treats "using a node" and "not using a node" as nearly equivalent, which is not appropriate: from an explainability perspective, omitting a node is always strictly better than including one. Ideally, unused nodes should retain a positive but modest advantage. One natural approach is to assign them a weight of the form

$$w_{\text{unused}} = \max(\text{feature\_weights}) + \varepsilon \cdot \text{mean}(\text{feature\_weights}), \tag{6}$$

where ε is a small factor (e.g. $\varepsilon = 0.1$). This preserves a preference for simpler models while avoiding both extremes: the collapse to a single-node tree seen with max +1, and the redundancy introduced by pure mean-based weighting. This can be implemented by scaling the weights by a constant factor so that everything becomes an integer although this might cost us in terms of computation time.



(a) With max_weight+1 (b) using method 6 $\epsilon = 0.1$ (c) Using method 5

Figure 4: We can see that we reach a miidle ground with the *ϵmethod*.

Another approach we explored to counter this form of redundancy was to introduce additional constraints that explicitly rule out nodes whose use is pointless. In particular, we target nodes for which all outgoing transitions lead to the same successor, regardless of the chosen predicate bucket. Formally, for any node with $\lambda_{i,p} = 1$, we require that it is not the case that all branches lead to the same node.

$$\lambda'_{i,p} \implies \neg \left(\bigwedge_{c=1}^{max(B(p))} \tau_{i,c,j} \right) \qquad \text{for all } j.$$

This condition prevents cases in which every branch from node i deterministically transitions to the same child j, ensuring that such structurally redundant nodes are excluded from the search space. In practice, however, enforcing this constraint is extremely inefficient: the additional quantification over all j and all buckets c significantly enlarges the SAT encoding and makes the solver considerably slower, to the point that the method becomes impractical for real use.

4.4 Computational Cost and Justification

While the reachability hard constraint approach yields structurally superior explanations, it incurs a significant computational cost as evident in 1 and 2

4.4.1 Complexity of Transitive Closure

Defining the reachability variable $R_{i,j}$ requires computing the transitive closure of the graph. In SAT encodings, this typically requires $O(N^3)$ clauses (where N is the number of nodes).

- **Original Encoding:** Size complexity is roughly $O(N^2)$ (for transitions).
- **Proposed Encoding:** Size complexity grows to $O(N^3)$ due to reachability propagation.

This increase in complexity is the prime deterrent in employing reachability constraints.

5 Naive Slack-Based Encoding

A direct approach to solving MaxSAT using Integer Programming is to relax the strict constraints of SAT by introducing "slack" variables. In a standard SAT problem, a clause C_j consisting of a set of literals is satisfied if the sum of the binary values of its literals is at least 1. For a hard clause, this constraint is absolute:

$$\sum_{l \in C_j} x_l \geq 1 \tag{7}$$

However, for soft clauses in Weighted MaxSAT, we must allow for violation. The naive encoding introduces a binary decision variable $s_j \in \{0, 1\}$ for every soft clause j. This variable acts as a toggle:

$$\sum_{l \in C_j} x_l + s_j \geq 1 \tag{8}$$

The logic operates as follows: if the literals sum to 0 (the clause is false), the inequality forces s_j to be 1 to maintain feasibility. If the literals sum to 1 or more (the clause is true), s_j is free to be 0.

To find the optimal solution, we minimize the weighted sum of these slack variables:

$$\text{Minimize} \quad Z = \sum_{j \in \text{Soft}} w_j \cdot s_j \tag{9}$$

This formulation allows standard MILP solvers to process the logical formula, though it treats the violation mechanism as a simple arithmetic patch.

6 Extended Formulation (Indicator-Based) Encoding

The Extended Formulation improves upon the naive Big-M approach by explicitly modeling the satisfaction status of each clause using an auxiliary indicator variable. Rather than treating clause violation as a breakdown of a constraint, this method bounds the satisfaction variable from both above and below.

6.1 Mathematical Formulation

For each clause C_j with length $k_j = |C_j|$, we introduce a binary decision variable $y_j \in \{0, 1\}$, where $y_j = 1$ denotes that the clause is satisfied.

The relationship between the literals x_l and the clause satisfaction y_j is enforced via two linear constraints:

1. **Implication of Falsity (Upper Bound):**

$$y_j \leq \sum_{l \in C_j} x_l \tag{10}$$

   This constraint ensures that if all literals in the clause are false (the sum is 0), the satisfaction indicator y_j is forced to 0.

2. **Relaxation Tightening (Lower Bound):**

$$y_j \geq \frac{1}{k_j} \sum_{l \in C_j} x_l \tag{11}$$

   This constraint links the satisfaction variable to the "degree" of truth in the clause. While logically redundant for integer solutions (since any non-zero sum allows $y_j = 1$), in the continuous relaxation, this cuts off fractional solutions where y_j might otherwise float freely.

The objective function minimizes the weighted dissatisfaction:

$$\text{Minimize} \quad Z = \sum_{j \in \text{Soft}} w_j (1 - y_j) \tag{12}$$

6.2 Illustrative Example

Consider a clause $C_1 = (x_1 \vee x_2 \vee x_3)$ with weight w.

- **Variables:** Literals x_1, x_2, x_3 and indicator y_1.
- **Constraints:**

$$y_1 \leq x_1 + x_2 + x_3 \quad \text{(If sum is 0, } y_1 = 0) \tag{13}$$

$$y_1 \geq \frac{1}{3}(x_1 + x_2 + x_3) \tag{14}$$

- **LP Relaxation Effect:** If the solver sets $x_1 = 0.3, x_2 = 0, x_3 = 0$, the lower bound forces $y_1 \geq 0.1$. In the Big-M formulation, such fractional correlations are often lost; here, the "cost" of the variable y_1 is strictly coupled to the variables x_i.

7 Pseudo-Boolean Strengthened Encoding

The Pseudo-Boolean (PB) Strengthened encoding augments the standard Big-M formulation by explicitly bounding the cardinality of falsified literals. While the standard encoding places a lower bound on the sum of satisfied literals, this formulation introduces a complementary "cover inequality" that places an upper bound on the sum of unsatisfied literals. This redundancy, while logically equivalent in the integer domain, significantly tightens the feasible region of the Linear Programming relaxation.[1]

7.1 Mathematical Formulation

For a clause C_j of length $k_j = |C_j|$, let x_l be the binary variable representing the truth of literal l, and let $\bar{x}_l = 1 - x_l$ represent its falsity. We introduce a relaxation variable $r_j \in \{0, 1\}$.

The encoding imposes two simultaneous constraints per clause:

1. **Satisfaction Lower Bound (Standard PB):**

$$\sum_{l \in C_j} x_l + r_j \geq 1 \tag{15}$$

   This ensures that the sum of true literals is at least 1, unless the clause is relaxed ($r_j = 1$).

2. **Falsity Upper Bound (Cover Inequality):**

$$\sum_{l \in C_j} \bar{x}_l \leq (k_j - 1) + k_j \cdot r_j \tag{16}$$

   This constraint asserts that the number of falsified literals cannot equal the clause length k_j. If all k_j literals are false, the left-hand side becomes k_j, violating the limit of $k_j - 1$. This forces the relaxation term $k_j \cdot r_j$ to activate (forcing $r_j = 1$) to satisfy the inequality.

The objective function remains the minimization of weighted relaxation variables:

$$\text{Minimize} \quad Z = \sum_{j \in \text{Soft}} w_j \cdot r_j \tag{17}$$

7.2 Constraint Redundancy and Relaxation Tightness

Although the Falsity Upper Bound is theoretically redundant to the Satisfaction Lower Bound for binary values $x \in \{0, 1\}$, they define distinct hyperplanes in the continuous space $x \in [0, 1]$. By including both, we enforce a "sandwiching" effect on the fractional solutions.

Consider a clause $(x_1 \vee x_2 \vee x_3)$.

- The Lower Bound $\sum x_i \geq 1$ permits a fractional assignment like $x_1 = 0.5, x_2 = 0.5, x_3 = 0$.
- The Falsity Upper Bound requires $\sum(1 - x_i) \leq 2$. For the assignment above, the falsity sum is $0.5 + 0.5 + 1.0 = 2.0$, which barely passes.

The true tightening occurs because the intersection of the two half-spaces eliminates "corners" of the polytope that are far from integer solutions. This helps the LP relaxation approximate the convex hull of the clause more closely than the Big-M formulation alone.

7.3 Algorithmic Advantage: The Knapsack Form

The primary motivation for introducing the redundant cover inequality is not merely geometric, but algorithmic. Modern MIP solvers (like Gurobi or CPLEX) utilize specialized cut-generation routines based on constraint structures.

- **Structural Recognition:** The standard Big-M constraint $\sum x_l \geq 1 - r$ is often treated as a generic linear row. However, the cover inequality $\sum \bar{x}_l \leq B + M \cdot r$ is structurally identical to a **Knapsack Constraint** (or a Generalized Upper Bound with capacity).
- **Cut Generation:** Recognizing the Knapsack structure allows the solver to apply sophisticated separation algorithms, such as **Cover Cuts** and **Lifted Cover Inequalities**. These cuts are valid linear inequalities that "slice off" fractional solutions that satisfy the explicit constraints but violate the underlying integer logic.
- **Branching Power:** By exposing this structure, we allow the solver to infer relationships between literals across different clauses. If multiple clauses share variables, the aggregated Knapsack constraints allow the solver to deduce conflict clauses (cliques) during the Presolve phase, significantly pruning the search tree before the Branch-and-Bound process even begins.

Therefore, the redundant constraint acts as a hint to the solver engine, unlocking a library of polyhedral strengthening techniques that would otherwise remain dormant for a standard Big-M formulation.

8 Quadratic MILP Encoding with McCormick Linearization

The Quadratic encoding represents a departure from standard additive (Big-M) formulations by modeling the logical falsity of a clause directly. Instead of summing variables to reach a threshold, this method calculates the product of "unsatisfaction indicators." A clause is considered violated if and only if all its constituent literals evaluate to false.

8.1 Mathematical Formulation

Let C_j be a clause in the WCNF formula. We introduce a binary decision variable $y_j \in \{0, 1\}$ representing the satisfaction of clause C_j, where $y_j = 1$ implies satisfaction.

For each literal l in clause C_j, we define an *unsatisfaction indicator* variable $u_{j,l} \in \{0, 1\}$. This variable takes the value 1 if the literal l evaluates to false under the current assignment of variables x. The mapping is defined as:

$$u_{j,l} = \begin{cases} 1 - x_i & \text{if } l = x_i \text{ (positive literal)} \\ x_i & \text{if } l = \neg x_i \text{ (negative literal)} \end{cases} \tag{18}$$

The condition that clause C_j is violated corresponds to the logical AND of these indicators. Mathematically, this is represented by the product:

$$P_j = \prod_{l \in C_j} u_{j,l} \tag{19}$$

Since $P_j = 1$ implies the clause is violated (all literals are false), the satisfaction variable y_j is constrained by:

$$y_j + P_j = 1 \tag{20}$$

8.2 McCormick Linearization

To handle the non-linear product term P_j within a Mixed-Integer Linear Program, we employ McCormick envelopes. The product is decomposed into a sequence of pairwise multiplications. For any binary product $z = a \cdot b$, we enforce the following linear constraints:

$$z \leq a \tag{21}$$
$$z \leq b \tag{22}$$
$$z \geq a + b - 1 \tag{23}$$

For a clause with k literals, this results in $k - 1$ auxiliary variables and $3(k - 1)$ linear constraints to compute the final product P_j.

8.3 Illustrative Example: 3-Literal Clause

Consider a clause $C_1 = (x_1 \vee \neg x_2 \vee x_3)$. The encoding proceeds as follows:

Step 1: Define Unsatisfaction Indicators

We generate indicators u_1, u_2, u_3 corresponding to the falsity of each literal:

$$\begin{aligned} u_1 &= 1 - x_1 \quad \text{(False if } x_1 \text{ is True)} \\ u_2 &= x_2 \qquad \text{(False if } \neg x_2 \text{ is True, i.e., } x_2 = 0) \\ u_3 &= 1 - x_3 \end{aligned}$$

Step 2: Linearize the Product

We require the product $P = u_1 \cdot u_2 \cdot u_3$. This is computed sequentially. First, we compute the intermediate product $z_1 = u_1 \cdot u_2$ using McCormick constraints:

$$z_1 \leq u_1 \tag{24}$$
$$z_1 \leq u_2 \tag{25}$$
$$z_1 \geq u_1 + u_2 - 1 \tag{26}$$

Next, we compute the final product P by multiplying the intermediate result z_1 with the remaining indicator u_3 (i.e., $P = z_1 \cdot u_3$):

$$P \leq z_1 \tag{27}$$
$$P \leq u_3 \tag{28}$$
$$P \geq z_1 + u_3 - 1 \tag{29}$$

Step 3: Link to Satisfaction

Finally, the clause satisfaction variable y_1 is linked to the violation product P:

$$y_1 + P = 1 \tag{30}$$

If $x_1 = 0, x_2 = 1, x_3 = 0$ (all literals false), then $u_1 = 1, u_2 = 1, u_3 = 1$. The constraints force $z_1 = 1$ and subsequently $P = 1$, forcing $y_1 = 0$. If any variable satisfies the clause, at least one u becomes 0, forcing $P = 0$ and allowing $y_1 = 1$.

9 Results

Dataset	Default	Naive MILP	Extended	PB-Strength.	Quadratic
theorem_prover	1.71	10.53	11.26	12.07	10.64
california_census	4.35	4.60	4.52	4.50	4.54
Iris3	9.47	10.67	10.79	10.91	10.72
Banknote3	17.11	10.58	10.53	10.52	10.53
ICML/AutoTaxi	600.00 (timeout)	26.18	24.32	24.02	24.10
Adult3_1	46.25	12.21	12.48	13.24	13.04
loan_acquisition	207.83	193.87	204.58	174.62	178.33

Table 3: Runtime (seconds) for selected solver variants (default MaxSAT, naive MILP, extended, PB-strengthened, quadratic) on each dataset for the reachability + mean-weight encoding (results2). Default MaxSAT for AutoTaxi timed out at 600 seconds.

Across all datasets, MaxSAT remains extremely efficient under the original encoding but degrades sharply when reachability constraints are added. In particular, the AutoTaxi instance times out at 600 seconds, demonstrating that MaxSAT is highly sensitive to additional structural constraints. In contrast, the various MILP encodings although slower on very small benchmarks, scale much more consistently. Among them, the PB-strengthened and quadratic encodings achieve the best overall performance, outperforming the naive MILP formulation and significantly reducing solve times on the larger datasets (Adult and Loan). These results suggest that MILP-based approaches offer a more robust alternative when richer structural constraints are introduced.

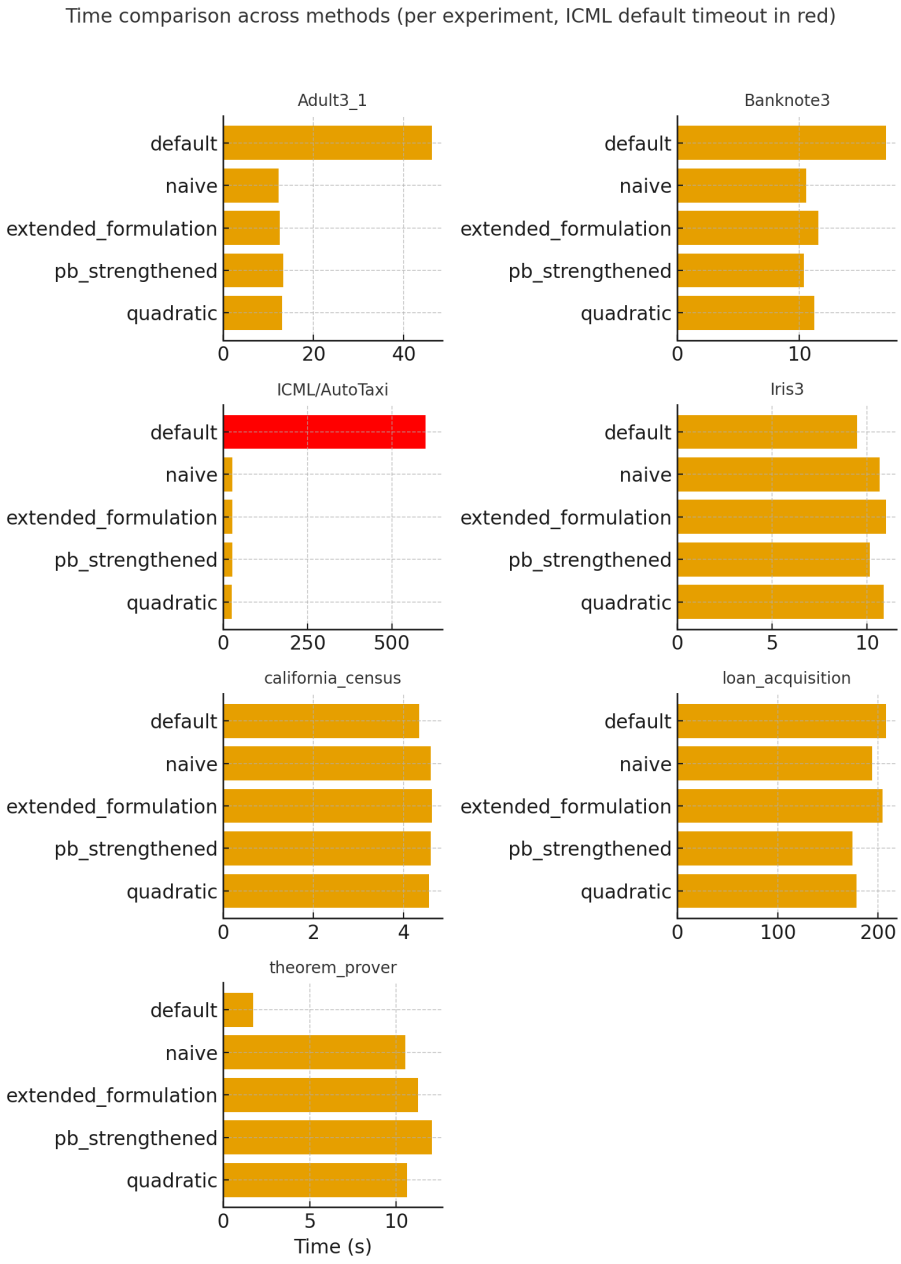


Figure 5: Theorem Prover

References

[1] John Hooker. *Logic-Based Methods for Optimization*. 01 2006.

[2] Hazem Torfah, Shetal Shah, Supratik Chakraborty, S. Akshay, and Sanjit A. Seshia. Synthesizing pareto-optimal interpretations for black-box models. *CoRR*, abs/2108.07307, 2021.