VIPR Certificate Construction from Black-Box ILP Solvers

Stefan Szeider
TU Wien, Vienna, Austria

Abstract

We propose an oracle-guided approach to construct VIPR certificates that certify the optimality of integer linear programming (ILP) solutions. Our approach treats the ILP solver as a black-box oracle and translates its floating-point answers into exact rational derivations. This translation is based on the notion of cascaded rationalization, a sequence of continued-fraction approximations at increasing precision levels. Our approach does not require an exact LP solver and comes with self-contained, independently verifiable VIPR certificates. We implement this approach and evaluate it empirically against SCIP's exact mode on generated benchmarks across four problem classes and on MIPLIB 2017 instances. On generated benchmarks, the oracle approach produces substantially more compact certificates on nearly all instances. On MIPLIB, each approach solves instances that the other cannot handle. The two approaches to certified optimality are complementary.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Theory of computation → Integer programming

Keywords and phrases Integer linear programming, VIPR certificates, LP duality, branch and bound, solver verification

Digital Object Identifier 10.4230/LIPIcs.CP.2026.52

Supplementary Material *Software (Source Code)*: `https://github.com/szeider/vorc`
Dataset (Experimental Data): `https://doi.org/10.5281/zenodo.20057193`

Funding Supported by the Austrian Science Fund (FWF), projects 10.55776/COE12 and 10.55776/P36420.

1 Introduction

Mixed-integer linear programming (MILP), and its all-integer special case ILP, is a fundamental framework for combinatorial optimization, with applications ranging from logistics and scheduling to circuit design and computational biology [15, 17]. State-of-the-art solvers such as Gurobi and CPLEX routinely solve large-scale instances. However, these solvers provide no guarantee of correctness. They rely on floating-point arithmetic, apply aggressive numerical heuristics, and employ millions of lines of code that inevitably contain bugs. Incorrect results have been reported in practice [4, 16], and a recent study confirms that numerical errors occur even in mature open-source solvers [11].

Independent verification of solver results is therefore a pressing concern, particularly in safety-critical applications and in computational proofs that feed into mathematical arguments [1]. The Verified Integer Programming Result (VIPR) format [3] provides a proof system for this purpose. A VIPR certificate encodes a sequence of inference steps that an independent checker verifies in exact rational arithmetic. If the checker accepts the certificate, the claimed optimum is correct, regardless of what the solver did internally.

To date, no solver-independent approach to VIPR certificate generation has been proposed. The primary method is SCIP's *exact mode* [4, 5], which records proof steps during its own branch-and-bound search and uses exact LP solvers (SoPlex or QSopt_ex) to compute the rational multipliers required for each derivation step. The advantage of this approach is

direct access to the solver's full machinery. The downside is tight coupling to a single solver, a dependency on exact LP solvers, and certificates whose size reflects the solver's internal search strategy.

Black-box Certification

We present an alternative approach that decouples the certificate generation from the solver. Any LP solver exposing dual multipliers and Farkas certificates can serve as oracle, including closed-source commercial solvers such as Gurobi. No exact LP solver is needed. The resulting certificates are self-contained. Every derivation step is verified before emission, and no post-processing is required.

Our approach utilizes the notion of a *limited-precision LP oracle* formalized by Gleixner and Steffy [9]. Such an oracle is a black-box solver that returns approximate primal-dual solutions but is never trusted to be exact. From such an oracle one can recover exact LP solutions via continued-fraction approximation. We exploit this oracle model for a different purpose. Instead of solving LPs exactly, we use the oracle to construct VIPR certificates that prove ILP optimality. Our certificate constructor runs its own branch-and-bound search, translating the oracle's floating-point answers into exact rational derivations through *cascaded rationalization* which applies continued-fraction approximation at increasing precision levels, each verified in exact arithmetic before acceptance.

We implement this approach as the prototype tool Vorc (*VIPR Oracle Construction*), written in Rust with Gurobi as the LP oracle. We evaluate Vorc against Scip running in exact mode on generated benchmark instances across four problem classes and on all 223 instances from the MIPLIB 2017 benchmark set [8].

Our results show that on generated benchmarks, the oracle approach produces certificates that are 14.4× smaller than Scip's on average (geometric mean over 69 instances), with smaller certificates on 97% of instances (Section 5, Table 1). On MIPLIB instances, Vorc solves 14 instances that Scip cannot handle within a 1-hour time limit, including one infeasibility certificate, while Scip solves 26 instances that Vorc cannot. The two approaches are complementary (Table 4). The certificate size advantage is roughly split between fewer derivation steps and more compact individual steps. However, this advantage decreases as constraint density increases (Tables 2 and 3).

Related Work

Scip's exact mode has been extended to include safe Gomory cuts [6] and certified constraint propagation [2]. On the verification side, the idea behind the work by Wood et al. [19] is to encode VIPR inference rules as SMT formulas; this uncovered ambiguities in the specification. However, this is orthogonal to our approach of generating certificates. To the best of our knowledge, no approaches for generating VIPR certificates outside the solver have been reported in the literature. Hoen and Gleixner [11] analyze the numerical correctness of branch-and-bound decisions a posteriori and outline a *grey-box* path that completes partial certificates by re-exploring incorrectly discarded sub-problems. Our approach does not require access to the solver internals and comes with its own branch-and-bound search. The key feature is that the resulting certificates do not require post-processing.

Proof logging has attracted a lot of attention in Boolean satisfiability, where solvers routinely emit DRAT proofs. Such proofs can then be independently verified using certified tools [14, 18]. A similar approach has been successfully applied for pseudo-Boolean optimization through VeriPB [7], MaxSAT preprocessing [12], and constraint programming [13].

Exact proof generation is harder for MIP than for satisfiability. In particular, this involves floating-point arithmetic, LP duality, and the integrality gap. In this paper, we extend this line of research to the MIP setting.

2 Preliminaries

2.1 Mixed-Integer Linear Programming

A *mixed-integer linear program* (MILP) takes the form

$$z^* \;=\; \min\{\, c^\top x \;\mid\; Ax \geq b,\; x_j \in \mathbb{Z} \text{ for } j \in I \,\},$$

where $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, and $I \subseteq \{1,\ldots,n\}$ is the set of integer-constrained variables. When $I = \{1,\ldots,n\}$, the problem is a pure *integer linear program* (ILP). The *LP relaxation* drops the integrality requirement: $z^*_{\text{LP}} = \min\{\, c^\top x \mid Ax \geq b \,\}$. Since the LP relaxation minimizes over a larger feasible set, we always have $z^*_{\text{LP}} \leq z^*$. The difference $z^* - z^*_{\text{LP}}$ is called the *integrality gap*.

2.2 The Verification Problem

Given a MILP instance and a claimed optimal value z^*, the verification problem is to confirm that z^* is indeed optimal. This requires establishing two bounds: the *upper bound*, a feasible solution x^* with $c^\top x^* = z^*$, which shows that z^* is attainable; and the *lower bound*, a proof that every feasible integer solution satisfies $c^\top x \geq z^*$, which shows that z^* cannot be improved.

The upper bound is straightforward to verify: one substitutes x^* into the constraints and the objective. The lower bound requires a mathematical argument. The certificate provides such an argument.

2.3 LP Duality

Our approach rests on LP duality to derive lower bounds. Consider an LP relaxation $\min\{\, c^\top x \mid A'x \geq b' \,\}$, where A' collects all constraints active at a search-tree node (original constraints, variable bounds, and branching constraints; $\leq$-rows are treated as their $\geq$-equivalent via negation). By strong duality, when an optimal solution exists, there exist multipliers $y^* \geq 0$ with

$$A'^\top y^* = c \qquad \text{and} \qquad b'^\top y^* = z_{\text{LP}}.$$

For any feasible x of the LP relaxation (and hence for any feasible integer solution), we have

$$c^\top x \;=\; (A'^\top y^*)^\top x \;=\; y^{*\top} A'x \;\geq\; y^{*\top} b' \;=\; z_{\text{LP}}.$$

The multipliers y^* thus certify the LP bound z_{LP}.

When an LP relaxation is infeasible, *Farkas' lemma* guarantees the existence of multipliers $y \geq 0$ with $A'^\top y = 0$ and $b'^\top y > 0$. These multipliers certify infeasibility: the weighted sum of the constraints yields the contradiction $0 \geq \beta$ with $\beta > 0$.

2.4 VIPR Certificates

The idea behind the Verified Integer Programming Result (VIPR) format [3] is to encode optimality proofs as a sequence of derived constraints. A VIPR certificate consists of four parts:

- CON: The original constraints, including explicit variable bounds, each numbered $C_0, C_1, \ldots, C_{m-1}$.
- SOL: A feasible solution x^* witnessing the upper bound.
- DER: A sequence of derived constraints $D_0, D_1, \ldots$, each justified by one of the inference rules below.
- RTP: The claim to be verified; for optimality, `range` z^* z^* (the objective equals z^*).

Each derived constraint is equipped with an implicit *assumption set* that tracks which branching hypotheses it depends on. By definition, the final derivation must have an empty assumption set.

The VIPR proof system provides four inference rules:

1. LIN *(linear combination).* Given constraints $C_{i_1}, \ldots, C_{i_k}$ and rational multipliers $\lambda_1, \ldots, \lambda_k$, the weighted sum $\sum_j \lambda_j C_{i_j}$ yields a new valid inequality. The combination must be *suitable*: the sign of each product $\lambda_j \cdot s(C_{i_j})$ must be consistent (all non-negative, or all non-positive), where $s(C) = +1$ for $\geq$-constraints, $s(C) = -1$ for $\leq$-constraints, and $s(C) = 0$ for equalities. The assumption set is the union of the referenced constraints' assumption sets.
2. RND *(integer rounding).* When all nonzero coefficients on the left-hand side are integers attached to integer variables, the left-hand side takes only integer values on feasible integer solutions. A $\geq$-constraint can therefore be tightened by rounding the right-hand side up to $\lceil \cdot \rceil$; a $\leq$-constraint by rounding down to $\lfloor \cdot \rfloor$.
3. ASM *(assumption).* This rule introduces a branching hypothesis (e.g., $x_j \leq \lfloor v \rfloor$); the constraint's own index enters its assumption set.
4. UNS *(unsplit).* Given two derivations under complementary assumptions ($x_j \leq k$ and $x_j \geq k{+}1$), this rule combines them by case analysis. Both branching assumptions are discharged from the merged assumption set.

A certificate checker such as `viprchk` [3] verifies every step in exact rational arithmetic. It checks suitability, recomputes each linear combination, and confirms that the final derivation dominates the claimed bound with an empty assumption set.

3 Approach

Our oracle-guided method for constructing VIPR certificates proves the optimality of ILP solutions. We treat the ILP solver as a black-box *oracle* that answers queries but is never trusted. Correctness rests entirely on the certificate. This certificate can then be independently verified using a certificate checker.

We begin by calling the ILP solver on the full problem to obtain a claimed optimal value z^* and a feasible solution x^*. We record x^* in the SOL section of the certificate as an upper-bound witness. The checker verifies that x^* satisfies all constraints and that its objective value equals z^*. The core task is then to prove the matching lower bound, that is, that no feasible integer solution achieves an objective value strictly below z^*. We achieve this by performing a *branch-and-bound* search over the LP relaxation. The per-node logic is illustrated in Figure 1.

At each node, we query the oracle for the optimal solution of the current LP relaxation. This relaxation consists of the original constraints and any branching constraints accumulated along the path from the root. We model all variable bounds and branching constraints as explicit LP constraint rows rather than as solver-internal variable bounds. This ensures that every constraint contributing to the LP dual has a corresponding dual multiplier accessible through the solver's interface. Otherwise, bound contributions would appear as reduced costs, requiring a separate extraction and mapping step.

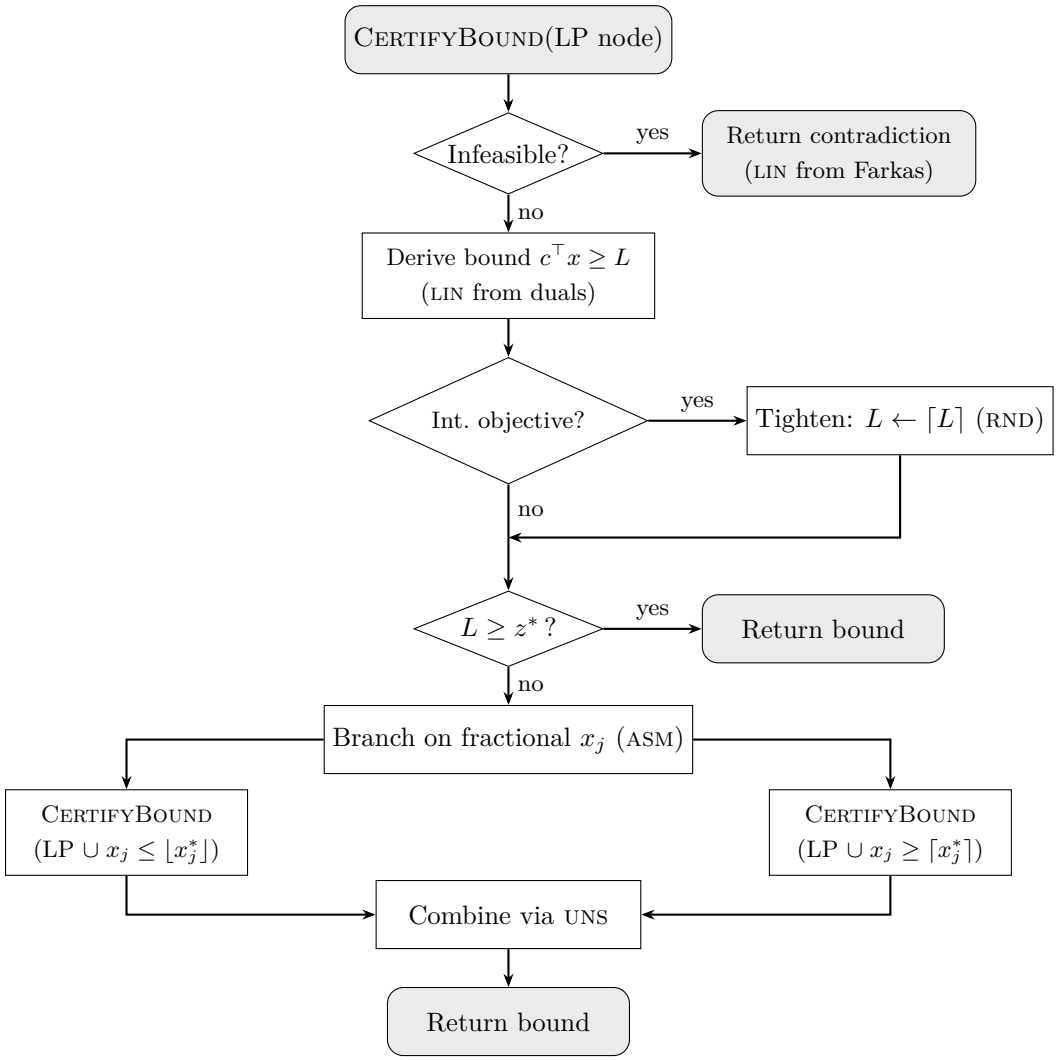

Figure 1 Per-node processing in the oracle-guided branch-and-bound. Each node queries the LP oracle and produces a certified derivation step. Grey boxes mark return points. VIPR inference rules are shown in parentheses.

When the LP relaxation is infeasible (typically because branching constraints exclude all solutions in this region) we obtain an *infeasibility certificate* in the form of a set of multipliers. Specifically, the weighted combination of the constraints yields a contradiction of the form $0 \geq \beta$ with $\beta > 0$. We translate these multipliers into an exact lin derivation step. A contradiction at a node certifies that every integer solution in this branch is infeasible. We therefore prune the node.

When the LP is feasible, the oracle reports dual multipliers that certify the LP optimum. By duality, these multipliers combine into a weighted sum of the constraints that yields a lower bound on the objective: $c^\top x \geq L$. However, the multipliers returned by the oracle are floating-point numbers. To produce an exact derivation, we approximate each multiplier by a rational number with a bounded denominator and verify the result in exact arithmetic. When the approximation at one precision level fails verification, we proceed to higher levels.

We refer to this procedure as *cascaded rationalization* (Section 4.3). The resulting LIN step is a weighted sum $\sum_i \lambda_i C_i$ of constraints C_i with rational multipliers λ_i. The certificate checker can independently verify this step.

If the derived lower bound L is fractional, we can often strengthen it. When every nonzero objective coefficient is an integer attached to an integer variable, the objective value of any feasible integer solution must be integral. Therefore, L can be tightened to $\lceil L \rceil$ via the RND rule. This rounding step converts a bound that is close to z^* but not quite sufficient to one that meets or exceeds it. If the derived bound, after possible rounding, satisfies $\lceil L \rceil \geq z^*$, no integer solution in this branch can improve upon z^*, and we prune the node.

When the bound does not suffice for pruning, some integer variable takes a fractional value x_j^* in the LP solution. We branch on this variable by introducing two *assumptions*: $x_j \leq \lfloor x_j^* \rfloor$ for the left child and $x_j \geq \lceil x_j^* \rceil$ for the right child. Each assumption is encoded as an ASM step in the certificate. We then recurse on each child, adding the respective branching constraint to the LP and querying the oracle again.

Afterwards, we combine the derivations from both children into a UNS (unsplit) step. This step reasons by cases. Observe that every integer must satisfy either $x_j \leq \lfloor x_j^* \rfloor$ or $x_j \geq \lceil x_j^* \rceil$. Since the two cases are exhaustive and both lead to $c^\top x \geq z^*$, the bound holds unconditionally. The two branching assumptions are discharged, and the resulting derivation has only the assumptions inherited from ancestors in the search tree. At the root, all assumptions are discharged, and the final derivation states $c^\top x \geq z^*$ with no assumptions (the search tree is finite under bounded-integer branching).

The certificate proves $z^* \leq \mathrm{OPT} \leq z^*$, hence $\mathrm{OPT} = z^*$. The upper bound $\mathrm{OPT} \leq z^*$ follows directly from the recorded solution x^*. The lower bound $\mathrm{OPT} \geq z^*$ follows from the branch-and-bound derivation, which holds for all feasible points without assumptions. Neither bound relies on trusting the oracle. Instead, the certificate checker verifies every step in exact rational arithmetic.

The construction extends in principle to mixed-integer programs: the LIN and ASM steps and the branch-and-bound logic are unchanged. Two MIP-specific issues require care. The RND step requires the objective to be integer-valued on feasible integer solutions; otherwise pruning relies on LIN alone (or on a rational scaling of the bound followed by RND, when applicable). The SOL section requires a feasible x^* in exact rationals, which for instances with continuous variables coupled by equality constraints needs a rational LP feasibility solver outside our oracle-only design. Our generated benchmarks are pure ILPs; the MIPLIB 2017 evaluation (Section 5) includes mixed-integer instances and shows the framework's MIP coverage whenever SOL recovery succeeds.

Running Example

We illustrate the construction on the binary program

$$z^* = \min\{\, 2x_1 + 3x_2 \;\mid\; 3x_1 + 2x_2 \geq 4,\ x_1, x_2 \in \{0,1\} \,\}.$$

The unique feasible solution is $(x_1, x_2) = (1, 1)$ with $z^* = 5$. The CON section lists five constraints: the model constraint $C_4\colon 3x_1 + 2x_2 \geq 4$ and four variable bounds $C_0\colon x_1 \geq 0$, $C_1\colon x_1 \leq 1$, $C_2\colon x_2 \geq 0$, and $C_3\colon x_2 \leq 1$. The SOL section records the feasible solution $(1, 1)$.

Root node. The LP oracle returns the optimum $x_1 = 1$, $x_2 = 1/2$ with $z_{\mathrm{LP}} = 7/2$. The dual multipliers for the two binding constraints (C_4 and C_1) yield a LIN derivation:

$$D_0\colon\ \tfrac{3}{2}\, C_4 - \tfrac{5}{2}\, C_1 \ \Rightarrow\ 2x_1 + 3x_2 \geq \tfrac{7}{2}.$$

The left-hand side matches the objective exactly; no LHS repair is needed. Since all objective coefficients are integers on integer variables, RND tightens the bound:

$$D_1\colon\ \lceil \tfrac{7}{2} \rceil = 4 \ \Rightarrow\ 2x_1 + 3x_2 \geq 4.$$

But $4 < 5 = z^*$, so the root is not pruned. We branch on x_2 (fractional at $1/2$).

Left child ($x_2 \leq 0$). The assumption $D_2\colon x_2 \leq 0$ is introduced via ASM. Adding this constraint makes the LP infeasible: $3x_1 \geq 4$ has no solution with $x_1 \leq 1$. The Farkas multipliers yield:

$$D_3\colon\ \tfrac{3}{2}\, C_1 - \tfrac{1}{2}\, C_4 + D_2 \ \Rightarrow\ 0 \leq -\tfrac{1}{2},$$

a contradiction certifying infeasibility under assumption D_2.

Right child ($x_2 \geq 1$). The assumption $D_4\colon x_2 \geq 1$ is introduced via ASM. The LP optimum is $x_1 = 2/3$, $x_2 = 1$ with $z_{\mathrm{LP}} = 13/3$. The dual multipliers give:

$$D_5\colon\ \tfrac{2}{3}\, C_4 + \tfrac{5}{3}\, D_4 \ \Rightarrow\ 2x_1 + 3x_2 \geq \tfrac{13}{3}.$$

Rounding: $D_6\colon \lceil 13/3 \rceil = 5 \geq z^*$. This branch is pruned.

Unsplit. Every integer satisfies either $x_2 \leq 0$ or $x_2 \geq 1$. The UNS rule combines the two branches:

$$D_7\colon\ \mathrm{UNS}(D_3, D_2, D_6, D_4) \ \Rightarrow\ 2x_1 + 3x_2 \geq 5,$$

with an empty assumption set. In VIPR semantics, an absurdity such as D_3 vacuously dominates any constraint, so the bound from the right branch carries through. Together with the feasible solution $(1, 1)$, this certifies $\mathrm{OPT} = z^* = 5$.

All multipliers in this example have denominators at most 3, well within the lowest cascade level ($D = 10^6$). The complete certificate has eight derivation steps and is verified by `viprchk` in exact rational arithmetic.

4 Deriving Exact Certificates from Floating-Point Duals

The central technical challenge is translating the oracle's floating-point dual multipliers into exact rational multipliers that yield valid LIN derivation steps. LP solvers compute in double-precision floating-point arithmetic (≈ 15.9 significant digits), while the certificate checker requires exact rational arithmetic. A naïve conversion of floating-point numbers to rationals produces denominators of the form 2^k that are unnecessarily large and can introduce artifacts.

We use a unified procedure to handle both objective bound derivation (target left-hand side: the objective function c) and infeasibility derivation (target: zero). The procedure consists of four steps: rationalization, linear combination, LHS repair, and verification.

4.1 Rationalization via Continued Fractions

Given a floating-point multiplier $\pi_i \in \mathbb{R}$, we approximate it by a rational number p/q with $|q| \leq D$ for a denominator bound D, using the classical continued-fraction algorithm [9, 17]. The standard bounded-denominator variant returns a nearest such rational in $O(\log D)$ iterations under unit-cost arithmetic.

The resulting rational multipliers $\lambda_i = p_i/q_i$ are then used to compute the linear combination exactly. Writing each constraint C_i as $a_i^\top x \geq b_i$, we obtain LHS and RHS of the derived constraint,

$$\text{LHS} = \sum_i \lambda_i \cdot a_i, \qquad \text{RHS} = \sum_i \lambda_i \cdot b_i,$$

where all arithmetic is performed in exact rational arithmetic using arbitrary-precision integers.

4.2 LHS Repair

The linear combination $\sum_i \lambda_i \cdot a_i$ should yield the objective coefficients c as its left-hand side. In practice, the oracle's floating-point multipliers are only approximate. Rationalization can also introduce discrepancies. As a result, the computed LHS may not match the target. Let $\delta_j := c_j - \hat{c}_j$ denote the gap between the target coefficient c_j and the computed coefficient $\hat{c}_j$ of x_j.

To correct each nonzero δ_j, we add a suitable variable bound constraint. Each bound constraint involves only a single variable, so the addition changes only the coefficient of that variable, by exactly δ_j. We distinguish two cases.

- If $\delta_j > 0$, we use the lower-bound row $x_j \geq l_j$ with multiplier δ_j, shifting the right-hand side by $\delta_j \cdot l_j$.
- If $\delta_j < 0$, we use the upper-bound row in its $\geq$-form $-x_j \geq -u_j$ with multiplier $|\delta_j|$ (equivalently, the raw row $x_j \leq u_j$ with multiplier δ_j), shifting the right-hand side by $\delta_j \cdot u_j$.

Both choices respect the suitability condition (Section 4.4): δ_j is non-negative on a $\geq$-row and δ_j is non-positive on a $\leq$-row. The right-hand-side shift makes the derived bound weaker but does not affect correctness.

4.3 Cascaded Rationalization

The denominator bound D controls the trade-off between approximation quality and coefficient size. A small D gives compact multipliers but may introduce too much error for the derivation to verify. Conversely, a large D gives more faithful multipliers but yields larger certificate entries.

Gleixner and Steffy [9] showed that rational reconstruction with speculative denominator bounds can recover exact solutions from oracles with limited precision. We adapt this idea to the certificate setting using a cascade of 13 levels: $D \in \{10^6, 10^7, \ldots, 10^{18}\}$. At each level, we attempt the full procedure: rationalization, linear combination, LHS repair, and suitability verification. We accept the first level that produces a valid derivation.

A key difference from exact LP solving is that we do not need to recover the exact dual solution. We only require multipliers that produce a *valid* derivation step, a weaker requirement. Consequently, a single oracle call per node suffices. This is in contrast to iterative refinement [10], which requires multiple calls to converge to an exact solution.

In our experiments, the lowest level ($D = 10^6$) handles the vast majority of branch-and-bound nodes. The remaining nodes require higher levels, typically because branching produces LP duals whose exact representation involves large denominators. The cascade terminates when we reach $D = 10^{18}$. Beyond this point, the resolution of double-precision floating-point makes further refinement unlikely to help.

4.4 Suitability Verification

After LHS repair, we verify that the multipliers satisfy the *suitability* condition required by the LIN rule. For a $\geq$-derivation, every product $\lambda_i \cdot s(C_i)$ must be non-negative. LP duality naturally produces multipliers satisfying this condition, but rationalization and repair can occasionally introduce violations. If the suitability check fails, we proceed to the next cascade level, where we use a tighter denominator bound.

For Farkas certificates (infeasible nodes), the derivation sense (whether $\geq$ or $\leq$) is not fixed in advance but depends on the multipliers: all products must be consistently non-negative or consistently non-positive. Since the sign convention of the LP solver is not fixed, we attempt both the original and the negated multiplier vector at each cascade level.

5 Experimental Evaluation

We evaluate our oracle-guided approach against SCIP's exact mode [4] on both generated benchmark instances and on all 223 instances of the MIPLIB 2017 benchmark set [8]. The VORC source code and the experimental data are linked from the supplementary material on the title page.

5.1 Setup

Our tool VORC is implemented in Rust and uses Gurobi 12.0.3 as the LP oracle. Two distinct parameter regimes apply. For the *initial MIP solve* that obtains z^* and a feasible solution x^*, we use Gurobi's defaults (`Presolve=-1`, `Method=-1`, `MIPGap=0`); this preserves Gurobi's full MIP machinery and full speed. For the *per-node LP queries* during certificate construction, we force dual simplex (`Method=1`), disable presolve, and turn off dual reductions, so that the constraint structure remains intact for dual-multiplier extraction. We additionally tighten `IntFeasTol`, `FeasibilityTol`, and `OptimalityTol` to 10^{-9} for sharper integrality and feasibility checks. We use a cascaded continued-fraction approximation with denominator bounds $10^6, 10^7, \ldots, 10^{18}$ to convert dual multipliers to exact rationals. Each level is verified in exact rational arithmetic before acceptance.

We compare VORC against SCIP 10.0.1 in exact mode, which produces VIPR certificates through its built-in certificate generation. Both tools write VIPR v1.1 certificates that are verified by `viprchk`. All experiments run on a single core of an Intel Xeon E5-2650v4 at 2.2 GHz with 8 GB RAM per job.

The two tools differ in how the certificate is produced. VORC runs an external branch-and-bound search and reconstructs each proof step in exact arithmetic from the floating-point dual multipliers returned by Gurobi. SCIP exact mode logs proof steps during its own internal branch-and-bound search and uses an exact LP solver (SoPlex or QSopt_ex) to compute the rational multipliers required at each step. This difference shapes the comparisons that follow: SCIP's tight coupling with its own solver is fast in wall-clock time, while VORC's externally reconstructed proofs tend to be more compact.

5.2 Generated Benchmarks

We generated 74 instances across four classes (3 seeds per configuration, except for ks $n = 20$ where the seed search found only two sufficiently hard instances):

- *Binary knapsack* (ks): $n \in \{20, 50, 100, 200, 500, 1000, 2000, 5000\}$.
- *Set cover* (sc): $n \in \{10, 20, 30, 50, 100\}$.
- *General integer* (gi): $n \in \{5, 10, 15, 20, 30, 50\}$.
- *Multi-dimensional knapsack* (mdk): $n/d \in \{30/8, 50/10, 100/10, 100/15, 200/10, 200/15\}$.

Timeouts are 600 s for $n < 200$ and 3600 s for $n \geq 200$.

Table 1 Certificate size comparison: VORC vs. SCIP exact mode. Each row averages 3 seeds; a superscript k/m on the size column indicates that only k of the m generated seeds were certified by both tools and the average is taken over those k seeds (ks $n = 20$ has $m = 2$ since the seed search produced only two sufficiently hard instances; all other configurations have $m = 3$). Configuration mdk 200/15 is omitted because neither tool produced a certificate within the time limit. "Ratio" is the geometric mean of per-instance $|\mathcal{C}_{\text{SCIP}}|/|\mathcal{C}_{\text{VORC}}|$; ratios at least 10× are shown in bold. Construction times are end-to-end and include the underlying solver time (initial MIP solve plus certificate construction for VORC; exact-mode solve plus emission for SCIP), measured in seconds. Timeout: 3600 s for $n \geq 200$, 600 s otherwise.

			Cert. size		Constr. time (s)	
Class	n	Ratio	VORC	SCIP	VORC	SCIP
Binary Knapsack	$20^{2/2}$	7.7×	49 KB	346 KB	0.4	0.2
	50	7.1×	247 KB	1.5 MB	1.0	0.6
	100	3.9×	2.0 MB	7.8 MB	5.6	2.1
	200	**135**×	227 KB	32.8 MB	1.1	8.8
	500	**215**×	52.0 MB	185.7 MB	128	52.6
	1000	**782**×	96 KB	155.7 MB	0.6	40.0
	2000	**19**×	1.9 MB	32.4 MB	4.8	8.2
	5000	**64**×	533 KB	54.5 MB	1.5	20.2
Set Cover	10	1.7×	2 KB	3 KB	0.1	0.1
	20	3.3×	4 KB	14 KB	0.0	0.1
	30	**15**×	10 KB	308 KB	0.1	0.2
	50	**45**×	206 KB	9.1 MB	0.9	6.6
	100	**70**×	2.5 MB	167.6 MB	11.5	168
General Integer	$5^{2/3}$	1.5×	3 KB	9 KB	0.1	0.0
	10	8.7×	4 KB	52 KB	0.1	0.1
	15	1.8×	25 KB	49 KB	0.1	0.0
	20	**19**×	111 KB	1.8 MB	0.6	0.3
	30	**29**×	192 KB	4.2 MB	0.9	0.6
	50	**16**×	704 KB	16.7 MB	2.2	2.2
Multi-dim. Knapsack	30/8	9.4×	147 KB	1.3 MB	0.6	0.4
	50/10	**10**×	1.3 MB	14.2 MB	3.8	2.8
	100/10	6.1×	7.2 MB	40.3 MB	12.7	11.5
	100/15	1.7×	60.9 MB	96.2 MB	77.4	31.8
	$200/10^{2/3}$	5.6×	51.2 MB	275.9 MB	64.5	66.9

Table 1 reports the certificate sizes and construction times for both approaches. Both tools produce verified certificates on 69 instances. VORC yields smaller certificates on 67 of 69 instances. The geometric mean size ratio across all 69 instances is 14.4× (median 9.6×).

The advantage varies across problem classes. Binary knapsacks exhibit the largest ratios, reaching 782× at $n = 1000$, where the LP relaxation is solved at the root without branching, while SCIP explores a substantial search tree. Set cover instances scale similarly, with ratios up to 70× at $n = 100$. For general integer problems, the advantage is a moderate but consistent 8.5× geometric mean. Multi-dimensional knapsacks with denser constraint matrices show a smaller advantage that decreases with density.

In Table 2, we decompose the size advantage into two factors: the number of derivation lines (DER entries) and the average size per line. Across all classes, the overall size ratio decomposes roughly evenly into these two factors.

Table 2 Certificate size advantage by instance class (geometric mean over all instances in the class). "Size ratio" is $|\mathcal{C}_{\text{SCIP}}|/|\mathcal{C}_{\text{VORC}}|$, with larger values meaning that VORC's certificate is more compact; "Lines" and "Per-line" decompose the total ratio multiplicatively into a structural component (fewer derivation steps) and a per-step component (more compact steps). The last column gives the number of instances solved by both tools.

Class	Size ratio	Lines ratio	Per-line ratio	Solved
Binary Knapsack	42.4×	3.6×	11.9×	23
Set Cover	12.1×	3.4×	3.6×	15
General Integer	8.5×	3.6×	2.4×	17
Multi-dim. Knapsack	5.6×	3.8×	1.5×	14
All classes	14.4×	3.6×	4.0×	69

Table 3 Effect of the multidimensional-knapsack generator parameter d (number of knapsack/resource constraints, $n = 100$ fixed) on the certificate size ratio. Here d acts as a benchmark-specific density control rather than a general LP-density notion. Each row averages 3 seeds.

	Cert. size			Constr. time (s)	
d	VORC	SCIP	Ratio	VORC	SCIP
3	164 KB	1.6 MB	**13**×	0.80	0.40
5	2.0 MB	7.8 MB	3.9×	5.38	2.25
8	9.2 MB	37.5 MB	4.2×	21.8	11.0
10	7.2 MB	40.3 MB	6.1×	12.9	12.2
12	41.8 MB	86.5 MB	2.1×	57.6	28.4
15	60.9 MB	96.2 MB	1.7×	78.7	32.2
20	104.6 MB	189.8 MB	1.8×	124	55.7

The fewer-lines factor is consistent with the oracle approach, pruning the tree earlier or exploring fewer nodes than SCIP's conflict-driven search. The per-line compactness arises in part because each oracle derivation directly encodes an LP dual certificate with a small set of active constraints. In contrast, SCIP's conflict analysis can produce derivations that reference many constraints. For binary knapsacks, the 42× advantage arises primarily from per-line compactness (11.9×). For multi-dimensional knapsacks, the advantage relies almost entirely on fewer lines (3.8× lines, only 1.5× per line).

To study the effect of *constraint density*, we ran a sweep over multi-dimensional knapsack instances with $n = 100$ variables and density $d \in \{3, 5, 8, 10, 12, 15, 20\}$. The results are summarized in Table 3 and Figure 2.

With increasing density, the size ratio generally decreases from 13× at $d = 3$ to 1.8× at $d = 20$. While both certificate sizes increase with density, the oracle certificates grow faster. For example, at $d = 3$, the oracle certificate is 164 KB versus SCIP's 1.6 MB, while at $d = 20$, the gap narrows to 105 MB versus 190 MB.

We attribute this trend to the per-line factor: with few constraints, each oracle derivation involves a small LP with sparse duals, yielding compact lines. As density increases, each LP has more active constraints, and the dual certificates become as verbose as SCIP's conflict-analysis derivations. The crossover at high density ($d \geq 15$ for $n = 200$) represents a regime where SCIP's approach is more effective.

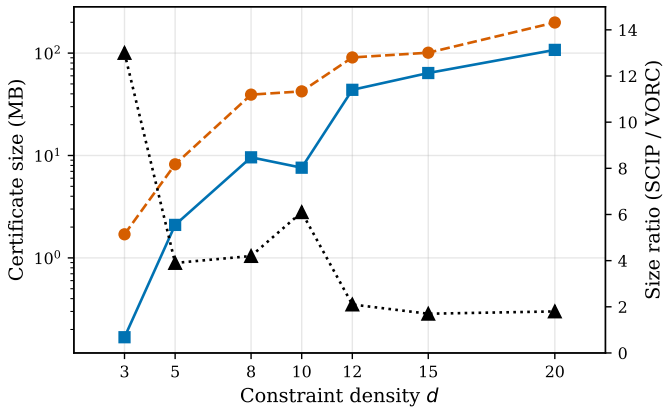

Figure 2 Certificate size and size ratio as a function of the generator parameter d (multi-dimensional knapsack, $n = 100$, averaged over 3 seeds). Left axis (log scale): ■ solid line: VORC; ● dashed line: SCIP. Right axis (linear): ▲ dotted line: size ratio $|\mathcal{C}_{\text{SCIP}}|/|\mathcal{C}_{\text{VORC}}|$.

5.3 MIPLIB Results

We evaluated both tools on all 223 instances tagged *benchmark* in MIPLIB 2017 (`https://miplib.zib.de/tag_benchmark.html`) with a 3600 s timeout (Table 4). VORC solved 21 instances (including one infeasibility certificate), and SCIP solved 33. Seven instances were solved by both; on six of them, the oracle certificate is 2.3–125× smaller. Hence the two methods are complementary.

6 Conclusion

We have introduced VORC, a tool for constructing VIPR certificates by querying an LP solver that exposes dual multipliers and Farkas certificates, without requiring an exact LP solver or access to solver internals.

Our experiments show that on generated benchmarks, the oracle approach yields certificates that are 14.4× smaller than SCIP's on average (Table 1). The advantage lies roughly equally in fewer derivation lines and more compact individual lines (Table 2). This advantage diminishes as constraint density increases (Table 3). On MIPLIB, 14 instances are handled by the oracle approach but not by SCIP, while SCIP handles 26 that the oracle approach cannot (Table 4). The two methods are complementary.

We outline several directions for future work. The oracle approach employs a simple branch-and-bound strategy without cutting planes or presolve. Incorporating these techniques could extend its reach to the more challenging MIPLIB instances where SCIP currently dominates. The modular architecture allows replacing Gurobi with any LP solver that exposes dual multipliers and Farkas certificates, including open-source alternatives such as HiGHS. It would be interesting to evaluate how oracle quality affects certificate construction.

The VIPR format supports infeasibility certificates, and our branch-and-bound machinery naturally extends to this setting. We run the search without an objective target, so each leaf must close via a Farkas absurdity (LP infeasible at that node); the merged unsplit derivations propagate this absurdity to the root with an empty assumption set, certifying infeasibility under `RTP infeas`. Among our failing MIPLIB instances, several are infeasible MIPs with feasible LP relaxations; we already handle one such instance (`neos859080`) successfully.

Table 4 MIPLIB 2017 instances solved by VORC (3600 s timeout, 223 benchmark instances). "Both" = solved by both tools; "VORC only" = solved by VORC but not SCIP. SCIP solved 33 instances total (26 uniquely). "Ratio" is $|\mathcal{C}_{\text{SCIP}}|/|\mathcal{C}_{\text{VORC}}|$; "—" marks rows where the ratio is undefined (no SCIP certificate) or below 1 (SCIP smaller, as for ex9).

	Cert. size		Time (s)		
Instance	VORC	SCIP	VORC	SCIP	Ratio
Solved by both (7 instances):					
decomp2	1.5 MB	185.8 MB	5.0	586	124.6×
ex9	5.0 MB	687 B	126	2085	—
markshare_4_0	1007 MB	4.8 GB	3127	602	4.9×
neos-827175	2.7 MB	53.1 MB	20	185	19.7×
neos8	23.6 MB	134.8 MB	1255	1338	5.7×
rmatr100-p10	43.5 MB	2.7 GB	324	775	64.0×
triptim1	14.2 MB	33.3 MB	140	807	2.3×
Solved by VORC only (14 instances):					
academictimetablesmall	4.4 MB	—	1861	t/o	—
cryptanalysiskb128n5obj16	7.3 MB	—	2822	t/o	—
ex10	11.2 MB	—	346	t/o	—
fhnw-binpack4-48	369 KB	—	3.3	t/o	—
neos-1171448	1.2 MB	—	6.7	t/o	—
neos-1171737	506 KB	—	5.5	t/o	—
neos-3004026-krka	1.4 MB	—	22	t/o	—
neos-933966	3.5 MB	—	41	t/o	—
neos-960392	4.7 MB	—	48	t/o	—
neos859080[†]	2.4 MB	—	4.7	t/o	—
ns1116954	5.4 MB	—	756	t/o	—
ns1952667	2.7 MB	—	24	t/o	—
peg-solitaire-a3	594 KB	—	435	t/o	—
supportcase18	944 KB	—	97	t/o	—

[†]Infeasibility certificate.

Finally, we note that a formally verified certificate checker, in the spirit of `cake_lpr` for SAT, would strengthen the trust model by replacing trust in the checker's implementation with trust in the proof assistant's kernel that established its correctness.

References

1 David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

2 Sander Borst, Leon Eifler, and Ambros Gleixner. Certified constraint propagation and dual proof analysis in a numerically exact MIP solver, 2024. `arXiv:2403.13567`.

3 Kevin K. H. Cheung, Ambros M. Gleixner, and Daniel E. Steffy. Verifying integer programming results. In Friedrich Eisenbrand and Jochen Könemann, editors, *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, Lecture Notes in Computer Science, pages 148–160. Springer, 2017. `doi:10.1007/978-3-319-59250-3_13`.

4 William J. Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Math. Program. Comput.*, 5(3):305–344, 2013. `doi:10.1007/S12532-013-0055-6`.

5 Leon Eifler and Ambros M. Gleixner. A computational status update for exact rational mixed integer programming. *Math. Program.*, 197(2):793–812, 2023. `doi:10.1007/S10107-021-01749-5`.

6 Leon Eifler and Ambros M. Gleixner. Safe and verified gomory mixed-integer cuts in a rational mixed-integer program framework. *SIAM J. Optim.*, 34(1):742–763, 2024. doi:10.1137/23M156046X.

7 Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying all differences using pseudo-boolean reasoning. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1486–1494. AAAI Press, 2020. doi:10.1609/AAAI.V34I02.5507.

8 Ambros M. Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff T. Linderoth, Marco E. Lübbecke, Hans D. Mittelmann, Derya B. Özyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. *Math. Program. Comput.*, 13(3):443–490, 2021. doi:10.1007/S12532-020-00194-3.

9 Ambros M. Gleixner and Daniel E. Steffy. Linear programming using limited-precision oracles. *Math. Program.*, 183(1):525–554, 2020. doi:10.1007/S10107-019-01444-6.

10 Ambros M. Gleixner, Daniel E. Steffy, and Kati Wolter. Iterative refinement for linear programming. *INFORMS J. Comput.*, 28(3):449–464, 2016. doi:10.1287/IJOC.2016.0692.

11 Alexander Hoen and Ambros M. Gleixner. Analyzing the numerical correctness of branch-and-bound decisions for mixed-integer programming. In Guido Tack, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 22nd International Conference, CPAIOR 2025, Melbourne, VIC, Australia, November 10-13, 2025, Proceedings, Part II*, Lecture Notes in Computer Science, pages 35–50. Springer, 2025. doi:10.1007/978-3-031-95976-9_3.

12 Hannes Ihalainen, Andy Oertel, Yong Kiam Tan, Jeremias Berg, Matti Järvisalo, Magnus O. Myreen, and Jakob Nordström. Certified maxsat preprocessing. In Christoph Benzmüller, Marijn J. H. Heule, and Renate A. Schmidt, editors, *Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part I*, Lecture Notes in Computer Science, pages 396–418. Springer, 2024. doi:10.1007/978-3-031-63498-7_24.

13 Wietze Koops, Daniel Le Berre, Magnus O. Myreen, Jakob Nordström, Andy Oertel, Yong Kiam Tan, and Marc Vinyals. Practically feasible proof logging for pseudo-boolean optimization. In Maria Garcia de la Banda, editor, *31st International Conference on Principles and Practice of Constraint Programming, CP 2025, Glasgow, Scotland, August 10-15, 2025*, LIPIcs, pages 21:1–21:27. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.CP.2025.21.

14 Peter Lammich. Efficient verified (UN)SAT certificate checking. *J. Autom. Reason.*, 64(3):513–532, 2020. doi:10.1007/S10817-019-09525-Z.

15 George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley interscience series in discrete mathematics and optimization. Wiley, 1988. doi:10.1002/9781118627372.

16 Arnold Neumaier and Oleg Shcherbina. Safe bounds in linear and mixed-integer linear programming. *Math. Program.*, 99(2):283–296, 2004. doi:10.1007/S10107-003-0433-3.

17 Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

18 Yong Kiam Tan, Marijn J. H. Heule, and Magnus O. Myreen. cake_lpr: Verified propagation redundancy checking in cakeml. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II*, Lecture Notes in Computer Science, pages 223–241. Springer, 2021. doi:10.1007/978-3-030-72013-1_12.

19 Kenan Wood, Runtian Zhou, Haoze Wu, Hammurabi Mendes, and Jonad Pulaj. Satisfiability modulo theories for verifying MILP certificates. *J. Symb. Comput.*, 135:102543, 2026. doi:10.1016/J.JSC.2025.102543.