PRINTEMPS and Its Hybrid Combinations with Exact and SCIP for the Pseudo-Boolean Competition 2026

Masahiro Sakai
Email: masahiro.sakai@gmail.com

Yuji Koguma
Email: yuji.koguma@gmail.com

***Abstract*—PRINTEMPS is a metaheuristic solver originally developed for general integer linear programming (ILP) problems. It was extended to handle pseudo-Boolean (PB) problems for the Pseudo-Boolean Competition 2025 [2]. For the Pseudo-Boolean Competition 2026, we submit three solvers: PRINTEMPS itself and two hybrid drivers, `exact-printemps` and `scip-printemps`, that run Exact [4] or SCIP [6] first and hand the incumbent and fixed variables over to PRINTEMPS as a warm start. We also widened the OPB parser's coefficient range from 32-bit to 64-bit signed integers, although the internal arithmetic remains double-precision and therefore restricts the exactly representable range to roughly 53 bits.**

I. Introduction

PRINTEMPS (PoRtable INTEger Mathematical Programming Solver) is a metaheuristic solver developed by the second author for general integer linear programming (ILP) problems [1]. The PB-specific extension and the rationale of the encoding of nonlinear terms and soft constraints are unchanged from our Pseudo-Boolean Competition 2025 submission, and we refer the reader to that description [2] for the details. This document focuses on what is new for 2026:

- widened OPB-parser coefficient range from 32-bit to 64-bit signed integers, with a remaining 53-bit limitation on the internal double-precision arithmetic (Sec. III-B);
- the two hybrid solvers `exact-printemps` and `scip-printemps` that pair PRINTEMPS with an off-the-shelf complete PB / MIP solver (Sec. IV);
- several safeguards against SCIP's double-precision numerical issues in `scip-printemps` (Sec. IV-C);
- minor PRINTEMPS-side improvements collected since the 2025 submission (Sec. III-D).

II. Supported Problems

The submitted solvers target the same six categories as our 2025 submission: **DEC-LIN**, **DEC-NLC**, **OPT-LIN**, **OPT-NLC**, **PARTIAL-LIN**, and **SOFT-LIN**. None of the three submitted solvers participates in the `-CERT` tracks: PRINTEMPS is heuristic and cannot prove optimality or infeasibility, and although Exact and SCIP are themselves complete, the hybrid drivers do not collect or emit a VeriPB-style certificate.

III. PRINTEMPS

A. Computational overview (recap)

PRINTEMPS searches for solutions to an ILP using Weighted Tabu Search that minimizes an objective function penalized by constraint violations [7], together with several ILP-specific enhancements such as instance size reduction, dependent variable extraction, neighborhood filtering, and incremental penalty evaluation [8]. The PB-specific extensions—linearization of products of binary variables and the introduction of slack variables with weighted penalties for soft constraints—are described in detail in our 2025 submission [2] and are reused unchanged.

B. Wider coefficient range

In the 2025 submission, each OPB coefficient had to fit within the range of a 31-bit signed integer at the parser level. For 2026, the OPB reader and the metadata accumulators have been migrated to `int64_t`, so coefficients up to the range of a 63-bit signed integer are accepted by the parser.

Internally, however, PRINTEMPS represents coefficients as IEEE-754 double-precision floating-point values, whose mantissa carries only 53 bits. To avoid silently producing wrong answers when a coefficient cannot be represented exactly, the `pb_competition_2025_solver` front-end inspects the bit width of the largest absolute coefficient (the `intsize` field of the OPB metadata) and, if it exceeds 53, prints `s UNSUPPORTED` and exits before solving. The hybrid drivers inherit this behavior because their second phase invokes the same PRINTEMPS executable. Compared to the 2025 submission, this widens the supported range from 31 to 53 bits, but instances with truly 64-bit coefficients remain out of scope.

C. Warm-starting interfaces

To make PRINTEMPS usable as the second phase of a hybrid pipeline (Sec. IV), the bundled executable `pb_competition_2025_solver` gained two new file-based command-line options:

- `-i FILE` — read an *initial solution* as a list of `xN VALUE` lines. The values are used to seed the local search.
- `-f FILE` — read a list of *fixed variables* in the same format. Variables listed here are pinned to the given value throughout the search.

Both inputs are post-processed by PRINTEMPS so that values supplied for original OPB variables are propagated to the auxiliary variables introduced internally during linearization, including negated-literal aliases and product variables.

D. Other PRINTEMPS-side improvements

The most user-visible non-hybrid change since the 2025 submission is that infeasibility detected during preprocessing (for example, contradictory variable bounds produced by bound propagation on the imported constraints) is now reported cleanly as `s UNSATISFIABLE` together with a human-readable message, instead of surfacing as an unhandled exception. The remaining changes are a handful of bug fixes and default-value tweaks that do not affect the algorithm.

IV. Hybrid Solvers

A. Motivation

PRINTEMPS is a local-search solver and depends strongly on the quality of the starting point. Complete PB / MIP solvers, on the other hand, often find good incumbents and prove a substantial number of variables to be globally fixed during presolve and root-node processing, but may exhaust their time budget before closing the gap on hard optimization instances. The two hybrid drivers, `exact-printemps` and `scip-printemps`, are intended to combine the two: spend an initial budget on a complete solver, hand its incumbent and the variables it has proved fixed to PRINTEMPS, and let PRINTEMPS continue to improve the assignment within the remaining time. We paired PRINTEMPS with two complementary complete solvers: Exact [4], a native PB / ILP solver whose search combines conflict analysis with cutting-plane reasoning, and SCIP [6], a widely used general-purpose MIP solver.

B. Two-phase pipeline

Both hybrid drivers follow the same two-phase shape:

1) **Phase 1 (complete solver).** `exact-printemps` spawns Exact as a subprocess; `scip-printemps` drives SCIP in-process via the Rust crate `russcip`. The instance is solved up to a configurable budget (default 300 s; capped by the overall wall-clock limit). The phase returns: a verdict; an optional incumbent assignment; the list of variables whose lower and upper bounds coincide after presolve and root-node processing (i.e. those proved *globally fixed*); and primal / dual bounds.
2) **Phase 2 (PRINTEMPS).** The driver invokes the bundled `pb_competition_2025_solver` on the same OPB instance with the remaining time budget. The incumbent is passed via the `-i` option of Sec. III-C; the list of fixed variables is passed via the `-f` option (gated by the `--use-fixed-literals` driver flag, which is enabled in the submitted command lines). If both files describe the same variable, the fixed-variable file wins.

If phase 1 returns a *final* verdict—`OPTIMUM FOUND`, `UNSATISFIABLE`, or `SATISFIABLE` on a decision instance—phase 2 is skipped, and phase 1's answer is emitted. Otherwise, phase 1's tentative `s ...` and `v ...` lines are downgraded to PB comments (`c exact-final: ...`, `c scip-incumbent: ...`) and PRINTEMPS' answer is emitted instead. If PRINTEMPS exits without a feasible solution while phase 1 did find one, the driver falls back to the phase-1 incumbent and synthesizes a `s SATISFIABLE` / `v ...` block from it.

The information exchanged between the two phases is collected in a single solver-agnostic payload carrying the incumbent, the fixed-variable list, and the primal / dual bounds. Fields are optional, so each phase populates only what it can produce—for example SCIP additionally supplies a dual bound, which is persisted to disk but not consumed by PRINTEMPS today. The driver forwards the OPB instance to the first-phase solver as-is; the nonlinear categories thus rely on the product linearization inside Exact / SCIP / PRINTEMPS rather than on the driver.

C. Guarding against SCIP numerical issues

SCIP solves in IEEE-754 double precision, which represents integers exactly only up to 2^{53}. This can bite in two ways: on instances with large coefficients, feasibility itself can be judged wrong (e.g. a constraint of the form $a - b \geq 1$ with $a, b \approx 2^{64}$ loses the ± 1 and looks trivially satisfiable); and even with smaller coefficients, LP relaxations can run into numerical instability mid-solve, corrupting the dual bound and any optimality or infeasibility *proof* built on it. To keep `scip-printemps` from emitting a wrong final answer under these conditions, three complementary guards are installed. The first two are always on; the third is on by default and controlled by a threshold flag.

- **Exact-integer incumbent verification.** Before SCIP's incumbent is used or persisted, the driver re-parses the original OPB and re-evaluates every constraint against the incumbent using `i128` checked arithmetic. On any violation—or if the check cannot be completed exactly (coefficient / activity out of `i128` range, or an unparsable token)—the *entire* SCIP result (incumbent, bounds, verdict, warm-start files) is discarded and the run falls through to PRINTEMPS as if SCIP had found nothing. In addition, once the incumbent has passed feasibility verification, its objective value is recomputed in `i128`; if the recomputed value disagrees with the objective SCIP reports, the incumbent is kept as a warm start, but the `OPTIMUM FOUND` claim is demoted to `SATISFIABLE`, since a disagreeing objective indicates SCIP may have pruned the true optimum.
- **Numerical-reliability guard.** This idea was inspired by the reference driver of the SCIP-NaPS solver submitted to the Pseudo-Boolean Competition 2025 [5][1]. SCIP's message output is tee'd to a log file and scanned for the strings `numerical troubles` (LP

[1]The technique is not discussed in the SCIP-NaPS solver description; we identified it in the driver source that accompanies the submission.

solver giving up on a node) and `out of range` (a coefficient the reader/presolver could not represent). When either appears, SCIP's *proof* as a whole is suspect: `OPTIMUM FOUND` is demoted to `SATISFIABLE`, `UNSATISFIABLE` is dropped to `Unknown` (there is no incumbent to fall back on), and the dual-derived information (the dual bound and the fixed-variable list) is cleared, because a bad dual reduction could pin a variable to the wrong value and steer PRINTEMPS away from the true optimum. The independently verified incumbent, if any, is preserved as a warm start.

- **`intsize` skip (`--scip-max-intsize N`, default 53).** PB-competition OPB files carry an `intsize=` field in their header comment (the bit length needed to represent the sum of the absolute values of the coefficients plus the degree in the largest constraint or objective). When it exceeds N, the SCIP phase is skipped entirely, and the instance is handed straight to PRINTEMPS. The default matches the `f64` exact-integer limit. Unlike the two verifiers above, this preempts failures that verification *cannot* catch, such as a false `UNSATISFIABLE` or a wrong optimum with no incumbent to re-check.

`exact-printemps` does not need these guards: Exact uses arbitrary-precision arithmetic internally.

D. Signal handling

A dedicated thread in the driver listens for `SIGINT`, `SIGTERM`, and `SIGXCPU` and forwards them to the currently active child via `kill(pid, sig)`; children are placed in their own process group so that a terminal Ctrl-C is not double-delivered. Both Exact and PRINTEMPS gracefully emit their best-known solution on these signals, and the driver waits for them to flush before exiting. One known limitation: `russcip`'s `solve()` consumes the model, so the driver currently cannot call `SCIPinterruptSolve` from a side thread; a signal arriving during phase 1 of `scip-printemps` is therefore observed only after SCIP terminates by its own time limit, and in that case phase 2 is skipped.

E. Limitations and future work

The current hybrid drivers are deliberately restricted to a simple sequential "phase 1 then phase 2" pipeline. Richer compositions we considered but did not implement in time for this submission include: (i) running PRINTEMPS and the complete solver *in parallel* and letting each pull the other's best incumbent as it improves; (ii) *alternating* the two solvers in multiple rounds, so that PRINTEMPS' improved incumbent is fed back to Exact / SCIP as a warm start and its dual bound in turn tightens the search space seen by PRINTEMPS. The inter-phase payload is already directional-agnostic, and the SCIP hand-off already carries a dual bound that is currently only persisted to disk; these are the mechanical prerequisites for such an extension. Wiring the loop itself, together with a matching flag on the PRINTEMPS binary to consume a dual bound, is left for future work.

V. Conclusion

For the Pseudo-Boolean Competition 2026, we submit PRINTEMPS together with two hybrid drivers, `exact-printemps` and `scip-printemps`, that respectively pair PRINTEMPS with Exact and SCIP. The PB-specific extensions of PRINTEMPS itself are inherited unchanged from the 2025 submission; the main additions are the wider coefficient range, the file-based warm-starting interfaces, the two-phase hybrid drivers built on top of them, and, for `scip-printemps`, a layered set of safeguards (exact-integer verification of the incumbent and its objective, a numerical-reliability guard on SCIP's messages, and an `intsize`-based skip) that prevent SCIP's double-precision arithmetic from turning into a wrong final answer.

References

[1] PRINTEMPS C++ metaheuristics modeler/solver for general integer optimization problems, https://snowberryfield.github.io/printemps/.

[2] Y. Koguma and M. Sakai, "An Incomplete Approach to Pseudo-Boolean Problems with Metaheuristic Solver PRINTEMPS," Solver description, Pseudo-Boolean Competition 2025, https://www.cril.univ-artois.fr/PB25/descr/PRINTEMPS-Pseudo_Boolean_Competition_2025.pdf.

[3] Pseudo-Boolean Competition 2026, https://www.cril.univ-artois.fr/PB26/.

[4] Exact: an exact integer linear program solver, https://gitlab.com/nonfiction-software/exact.

[5] SCIP-NaPS submission, Pseudo-Boolean Competition 2025, https://www.cril.univ-artois.fr/PB25/descr/NaPS_PB2025.pdf.

[6] SCIP: Solving Constraint Integer Programs, https://www.scipopt.org/.

[7] K. Nonobe and T. Ibaraki, "A Tabu Search Approach to the Constraint Satisfaction Problem as a General Problem Solver," *Eur. J. Oper. Res.*, Vol. 106, pp. 599–623, 1998, doi:10.1016/S0377-2217(97)00294-4.

[8] Y. Koguma, "Tabu Search-Based Heuristic Solver for General Integer Linear Programming Problems," *IEEE Access*, Vol. 12, pp. 19059–19076, 2024, doi:10.1109/ACCESS.2024.3361323.