Confidence is a certificate

Kenneth Pernyér, Reflective Labs
Stockholm, Sweden
kenneth.pernyer@reflective.se · reflective.se · ORCID 0009-0006-4543-1156

Most real business decisions are constrained optimization problems dressed in plain language, and a language model asked to solve one will return a schedule that reads well and can't be checked. This paper describes Ferrox, which exposes CP-SAT, minimum-cost flow and a MIP solver as Suggestors inside the convergence loop of my substrate report, so that a proof of optimality and a fluent explanation can be produced by different participants in the same Formation. The engineering is unremarkable and the semantics aren't: a heuristic answer and a proven one both enter the context, and the only thing separating them is a scalar confidence. I argue that this scalar is doing type-level work. A confidence of one means *proven optimal*, 0.85 means *feasible, optimality not proven*, and a heuristic is capped at 0.65 however good it looks — these are classes of certificate, not probabilities, and averaging them with the probability attached to a model's output is a category error. I give the confidence rules, three benchmark instances, and one result that's worse than it looks good: on a time-windowed routing instance the solver proves that only eight of twenty customers can be served, which is a modelling finding rather than a solver failure and is reported here as such. I close by proposing that a scalar is a lossy projection of a certificate lattice I haven't built.

1. Introduction

"Schedule the field crews for next week." "Which projects should we fund this quarter?" "Route the vans." Each of these is a sentence in plain language and a constrained optimization problem underneath, and the gap between the two is where a great deal of organizational software lives badly.

Language models are extraordinarily good at the outer work: reading the unstructured request, pulling the relevant history, formulating the problem, and explaining the answer to the person who has to act on it. They aren't optimizers. Given sixty tasks, twelve technicians, five skills and a six-hour horizon, a model will produce a schedule that sounds reasonable. It can't prove the schedule is the best available, can't guarantee every constraint is respected, and can't say how far from optimal it is. Those three absences aren't a prompting problem.

Ferrox fills the inner loop with solvers that can do all three. It wraps three slices of Google's OR-Tools — CP-SAT, the GLOP simplex, and a minimum-cost flow — and HiGHS for mixed-integer and linear programming (Huangfu and Hall, 2018), behind narrow Rust contracts in which `unsafe` is forbidden outside the binding crates. They are mature descendants of branch and bound (Land and Doig, 1960) and of thirty years of computational improvement that dwarfs the hardware (Bixby, 2012). They register as Suggestors in the same Formation as the language models, read the same context, and write confidence-tagged proposals to it. The substrate is the one described in my previous report (Pernyér, 2025) and I don't restate it.

Outcome	Confidence
Optimal solution found	$c = \text{visit ratio}$ — exactly 1.0 when every task is served
Feasible, not proven	$c = \text{visit ratio} \times 0.85$
Infeasible or error	$c = 0$
Greedy heuristic	$c = \text{throughput ratio} \times \text{cap}$, with $\text{cap} \leq 0.65$

Table 1 | The confidence rules. The cap is the load-bearing part: a greedy plan can't report above 0.65 no matter how good it happens to be, so it always yields to a proven plan and never displaces one.

The interesting question isn't how to call a solver. It's what it means for a proven answer and a guessed answer to sit in the same context, distinguished only by a number.

2. The certificate ladder

Every Ferrox Suggestor reports a confidence, and the rule that produces it is fixed per outcome class rather than tuned.

The ordering these rules induce is what makes a mixed Formation safe.

Definition (Certificate class). A proposal's certificate class is what its producer can demonstrate about it: **proven** ($c = 1$) — this is optimal and the solver has a proof; **feasible** ($c \approx 0.85$) — all constraints hold, optimality is unproven; **heuristic** ($c \leq 0.65$) — constraints hold, no claim beyond that; **none** ($c = 0$) — infeasible, or the run failed.

Because the bands don't overlap, a downstream consumer that selects by confidence is selecting by certificate class first and by quality second, which is the intended reading. The greedy scheduler that fits fifty-six of sixty tasks in three hundredths of a millisecond is genuinely useful — it seeds the context immediately, so the operator's screen is never empty and the language model downstream has something to explain while the solver works — and it must never be able to outrank a proof.

2.1. Confidence isn't a probability

This is where I depart from the common reading, and the departure is the argument of this paper.

A confidence of 0.85 on a feasible-but-unproven plan doesn't mean the plan is optimal with probability 0.85. It means: *all constraints are satisfied, and I couldn't prove optimality inside the budget.* A confidence of 0.60 on a greedy plan doesn't mean it's right sixty percent of the time; it means *I am a heuristic and I am telling you so in the only channel available.* Neither is a frequency, neither is calibrated against outcomes, and neither should be multiplied by anything.

The category error to avoid is combining these numbers with the probability a language model attaches to its output. Those are different kinds of quantity: one is a claim about what has been demonstrated, the other is a claim about what is likely. Averaging them produces a number with no interpretation at all — and, worse, one that can let a fluent guess outrank a proof if the weights are unlucky. Selection by maximum is safe under the banded scheme of Table 1 precisely because it never mixes.

Proposition (Band safety). Under Table 1, for any proven proposal p and any heuristic proposal q, $c_q \leq 0.65 < c_p$ whenever p serves at least 66% of the instance. Selection by maximum confidence therefore never prefers a heuristic to a proof on instances the solver substantially serves.

The qualification in that statement is real and I state it rather than hide it: a proven-optimal plan that can only serve half the tasks reports 0.5, and loses to a greedy plan that serves nearly all of them at 0.6. That's the intended behaviour — serving more work is better than proving less work is maximal — but it means the scalar is genuinely ordering two different things at once, which is the weakness discussed in Section 6.

3. Which solver, and why it's a catalog

Wrapping a solver suite is easy. Deciding *which* solver a request should reach is the part that determines whether the capability is usable by an agent, and I treat it as a first-class surface rather than as documentation.

The two backends are good at different things, and the difference is structural rather than a matter of maturity. CP-SAT fuses SAT solving with lazy clause generation (Ohrimenko et al., 2009) and linear relaxations, which makes it decisive on finite-domain models that mix logic and arithmetic: optional intervals, no-overlap, disjunctive sequencing, circuit constraints, all-different structure. These are exactly the models where a pure mixed-integer formulation struggles, because the linear relaxation is weak and the branch-and-bound tree explodes. Its standing isn't my claim to make; it has dominated the MiniZinc Challenge for years (Stuckey et al., 2014).

HiGHS covers the workload CP-SAT is the wrong tool for: continuous and mixed-integer *linear* models where integrality and a quantified optimality gap matter. Its dual revised simplex (Huangfu and Hall, 2018), interior point and branch-and-cut are the reason it's the default backend under a good deal of scientific Python, and my MIP Suggestor reports the gap it returns rather than discarding it.

I wrap only the slices with a product case — CP-SAT, GLOP and minimum-cost flow from OR-Tools, MIP and LP from HiGHS — and leave the rest of both suites alone. The suite also carries a first-order large-scale LP method and a specialized routing library built on guided local search; neither is exposed, because an exposed capability is one I have to support, version and defend.

Definition (Suggestor catalog). A machine-readable mapping from use case to the Suggestors that can serve it, including explicitly *deferred* entries. `recommend_for_use_case` returns ranked candidates by symbol, so a planning layer can select a solver from intent without knowing solver internals — and a use case with no entry reports as unserved rather than being routed to whichever backend happens to be compiled in.

The deferred entries are the part I'd defend hardest. A catalog that lists only what works invites the caller to assume that anything absent is unsupported by accident. Listing "SMT counterexample search — deferred, see the assurance extension" tells a planner where the capability lives and that I know it's missing here, which is the difference between a gap and a silence.

Instance	Served	Time	Conf.
Assignment, greedy	56 / 60	0.03 ms	0.60
Assignment, CP-SAT	60 / 60	260 ms	1.00
Job shop, greedy	2038	0.3 ms	0.55
Job shop, CP-SAT	1044	30 s	0.85
Routing, nearest	5 / 20	< 0.1 ms	0.15
Routing, CP-SAT	8 / 20	4.9 s	0.40

Table 2 | Assignment: 60 tasks, 12 agents, 5 skills, 360-minute horizon. Job shop: 15 jobs on 10 machines, Taillard-style (Taillard, 1993), reported as makespan rather than count. Routing: 20 customers with time windows, Solomon-style (Solomon, 1987), depot central, 480-minute horizon.

4. Three instances

I report three benchmark instances. They are small, they are public in shape if not in data, and they are chosen because each shows something different.

The assignment instance is the clean case: the solver proves that all sixty tasks fit, the greedy scheduler had dropped four, and the difference is four customer appointments that either happen or are rescheduled. At forty technicians and eight hundred orders a week, the compounding isn't a technical curiosity — it's headcount.

The job shop instance shows the second band doing its job. CP-SAT improves makespan by 48.8% over greedy dispatching and *does not prove* optimality within thirty seconds, so it reports 0.85 rather than 1.0. The Formation uses it, marked unproven. Nothing about the pipeline pretends the number is a proof.

4.1. The routing result is a modelling finding

The third instance is the one worth publishing. On twenty customers with time windows, nearest-neighbour serves five and CP-SAT serves eight — a sixty percent improvement, and an answer that's *optimal* for the model as written. Eight of twenty is also, from any operational standpoint, a bad day.

The temptation is to report the improvement and stop. The honest reading is that a proof of optimality is a proof about a model, and this model says the instance is over-constrained: with these windows, this horizon and this depot, twelve of the twenty customers can't be served at all. That's information the organization urgently wants and no heuristic could have produced, because a heuristic that serves five simply looks like a heuristic that could try harder.

I take a position on what to do with it. An optimal-but-poor result should surface as a **constraint finding**, not as a plan: the useful output is *which window, if relaxed by how much, admits how many more customers*, which is a sensitivity question the solver can answer and the current implementation does not ask. That's the largest gap between what Ferrox does and what it should do.

5. Anytime behaviour

The greedy and exact Suggestors aren't alternatives; they run together. The heuristic seeds the context in under a millisecond; the solver refines it over seconds. Because the context is append-only and both proposals are promoted, the Formation is **anytime** without anyone implementing anytime logic: whatever the budget, the best certificate available at that moment is already in the context, and the exit taken — converged, budget exhausted, blocked, escalated (Pernyér, 2025) — says which.

This is the practical reason the substrate's monotonicity axiom is worth its cost. In a system where later results overwrite earlier ones, a timeout leaves you with whatever happened to be last. In a monotone context, a timeout leaves you with everything that was established, ordered by what can be demonstrated about it.

6. Beyond the scalar

A single number is a lossy projection of what the solver knows, and I can say precisely what it loses. A CP-SAT run that stops early knows its incumbent objective **and** its dual bound, which together give an optimality gap; a MIP run knows the same; an infeasible run may know an irreducible infeasible subsystem — the specific set of constraints that can't hold together. The scalar carries none of this.

What I'd prefer is a small lattice of certificates ordered by strength — $\text{none} < \text{heuristic} < \text{feasible}(\gamma) < \text{proven}$, with the gap γ carried explicitly and an infeasibility certificate as a first-class value rather than a zero — and a selection rule that compares certificates lexicographically before comparing quality. Under such a scheme the routing result wouldn't have been an 0.40 plan; it would have been a proof, attached to a feasibility diagnosis, that the request as posed can't be met.

> **Open direction (Certificate lattice).** Define a partial order on certificates carrying (i) the optimality gap where one exists, (ii) an infeasibility subsystem where the instance is over-constrained, and (iii) the model version the certificate is relative to. Investigate whether a Formation ordered by such a lattice retains the confluence property proved for the scalar case.

I haven't built this. The clause about confluence is the reason for caution: the substrate's guarantees rest on proposals merging by set union under a total order on confidence, and a partial order admits incomparable

proposals, which is exactly the situation the merge was designed not to face.

7. The dependency you inherit

A paper about putting native solvers inside a governed decision loop owes the reader an account of what else comes in with them, and this is the section such papers usually omit.

The two backends have very different supply-chain shapes. HiGHS is a self-contained C++ codebase with essentially no bundled third-party tree; its vulnerability surface is its own code, and historically it has carried no notable published vulnerabilities. Upgrades can be leisurely and fix-driven. OR-Tools bundles Abseil and Protobuf, and Protobuf in particular carries a steady drip of parsing denial-of-service advisories — not because it's badly maintained, but because a widely-deployed parser attracts attention.

The consequence for policy is asymmetric and I state it as such: for OR-Tools, *a vulnerability in a bundled dependency* is the primary trigger for an unscheduled upgrade, while HiGHS moves on its own schedule. Both are vendored from tag-pinned, commit-verified sources and swept in the release gates, so an advisory anywhere in the chain surfaces as a failing gate rather than as a production incident.

This belongs in a research report for a reason that generalizes past my stack. The assurance argument for a governed system is only as strong as its weakest link, and a system whose correctness rests on a solver's answer has taken a dependency on that solver's parser, its allocator and its transitive tree. A certificate of optimality computed by a process that can be crashed by a malformed input is a guarantee about mathematics attached to a component with an operational failure mode. Naming which dependency drives the upgrade cadence is the least a report like this can do.

The Rust side of the boundary is arranged to keep this contained: native linking lives in `*-sys` crates, `unsafe` is forbidden in the library that Formations actually call, and default builds link no native solver at all.

8. Limitations

The certificate is relative to the model, not to the world. Every guarantee in this paper is of the form *optimal for the constraints as formulated*, and the formulation is produced upstream, frequently by a language model reading unstructured input. A perfectly proven answer to a subtly mis-transcribed problem is worse than an obviously rough one, because it arrives with a certificate.

The catalog reduces mis-routing and doesn't eliminate it: a use case is matched by name, and a request that resembles a modelled class without belonging to it will be served by a Suggestor that answers a different question well.

Native solver builds are expensive: a first clone should expect fifteen to thirty minutes of C++ compilation before any OR-Tools-backed check can run, and the same for HiGHS. This is a real adoption cost and it's why the catalog includes pure-Rust baselines that need no native toolchain.

The problem classes here are scheduling, assignment, routing and knapsack. They cover a large share of what organizations actually ask, and they aren't a general optimization capability: a request that doesn't fit one of the modelled classes is routed to a Suggestor that can't help it, and the catalog exists to make that visible rather than to hide it.

Finally, the benchmarks are single instances, not distributions. They illustrate behaviour; they don't establish performance, and I haven't run the standard instance families (Solomon, 1987, Taillard, 1993) end to end inside a Formation.

9. Conclusion

Putting a solver in the same loop as a language model is easy. Making the two answers comparable is the part that required a decision, and the decision was to treat confidence as a statement about what can be demonstrated rather than about what is likely. Proven, feasible, heuristic, none — four bands that do not overlap, so that selection by maximum can never prefer fluency to proof.

The scalar is a compromise I can now see the shape of. It orders certificate strength and solution quality on one axis, which is why a proof about a poor instance can lose to a heuristic about a good one, and it discards the optimality gap and the infeasibility subsystem that the solver already computed. The routing instance is where that hurts: I reported a plan when I should have reported a diagnosis.

Language models understand the request; solvers prove the answer. Language in, certainty out — and the interesting engineering is in the channel between them, which is narrower than it should be.

> **Code and correspondence.** The work described here is implemented, not sketched. The source behind every claim in this paper — the engine, the extension, the fixtures and the tests that hold the invariants — can be shared on request, including the parts that did not work. Write to kenneth.pernyer@reflective.se. We are equally interested in being told where the argument is wrong.

References

Bixby, R.E., 2012. A brief history of linear and mixed-integer programming computation. Documenta Mathematica 107–121.

Huangfu, Q., Hall, J.A.J., 2018. Parallelizing the dual revised simplex method. Mathematical Programming Computation 10, 119–142.

Land, A.H., Doig, A.G., 1960. An automatic method of solving discrete programming problems. Econometrica 28, 497–520.

Ohrimenko, O., Stuckey, P.J., Codish, M., 2009. Propagation via lazy clause generation. Constraints 14, 357–391.

Pernyér, K., 2025. Proposal, gate, commitment: deterministic convergence as a substrate for organizational decisions. Reflective Labs technical report.

Solomon, M.M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35, 254–265.

Stuckey, P.J., Feydy, T., Schutt, A., Tack, G., Fischer, J., 2014. The MiniZinc challenge 2008–2013. AI Magazine 35, 55–60.

Taillard, É.D., 1993. Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285.

A. Confidence, operationally

C1 Proven The solver reports optimality. Confidence is the visit ratio — the fraction of the instance served — so a proof that serves everything reports exactly 1.0 and a proof that serves less reports proportionally less.

C2 Feasible All constraints hold, optimality is unproven within the budget. Confidence is the visit ratio scaled by 0.85.

C3 Heuristic A greedy or constructive method. Confidence is the throughput ratio scaled by a cap that never exceeds 0.65.

C4 None Infeasible, timed out with no feasible incumbent, or errored. Confidence is 0 and the proposal carries the reason.

C5 Routing The Suggestor catalog maps a use case to the Suggestors that can serve it, including explicitly deferred entries. A use case with no entry is reported as unserved rather than routed to the nearest compiled solver.

B. The three models in outline

B.1. Assignment with time windows

Tasks T, agents A, a skill relation $S \subseteq T \times A$, per-task windows $[e_t, l_t]$ and durations d_t. Boolean x_{ta} assigns t to a; interval variables carry d_t within $[e_t, l_t]$; a no-overlap constraint per agent enforces exclusivity. The objective maximizes $\sum_t \sum_a x_{ta}$ — tasks served, not makespan — which is why the confidence is a visit ratio.

B.2. Job shop

Jobs J, machines M, each job a fixed sequence of operations with durations. Interval variables per operation, no-overlap per machine, precedence within a job, and an objective minimizing the makespan $\max_j C_j$. Reported as makespan rather than as a served count, so the visit ratio is 1 by construction and the band alone distinguishes proven from feasible.

B.3. Routing with time windows

Customers C with windows and service times, a depot, and a horizon. The objective maximizes customers served subject to travel time, windows and the return deadline. When the optimum serves few customers, the binding constraints — not the objective — are the result worth reporting, which is the argument of Section 6.

