

Parallel Lifted Planning via Semi-Naive Datalog Evaluation

Dominik Drexler Oliver Joergensen Jendrik Seipp
29th of June, 2026

Linköping University, Linköping, Sweden

Motivation: Lifted Classical Planning & the Grounding Bottleneck

- ▶ Lifted planners operate directly on the first-order task representation, avoiding task grounding.
- ▶ The bottleneck shifts to expensive per-state reasoning in lifted subroutines.
- ▶ We parallelize these subroutines via semi-naive Datalog evaluation, complementary to parallel search.

- Core lifted subroutines are **Datalog rule evaluation**: successor generation, axioms, grounding, and the delete-relaxed planning graph heuristics.

Relaxed planning graph (RPG) program: for action schema $a(\bar{x})$ and positive effect $p_e(\bar{x}_e)$:

$$r_{a,p_e} : p_e(\bar{x}_e) \leftarrow p_1(\bar{x}_1), \dots, p_m(\bar{x}_m), \neg p_{m+1}(\bar{x}_{m+1}), \dots, \neg p_n(\bar{x}_n)$$

- We **parallelize semi-naive fixed-point evaluation**: only rule instances touching a newly derived fact:

$$\Delta_i^{t+1} = \{\text{Head}(\bar{r}) \mid \bar{r} \in \text{gr}(\mathcal{R}_i, J_i^t), \text{Body}(\bar{r}) \cap \Delta_i^t \neq \emptyset\} \setminus J_i^t$$

The Substitution Consistency Graph

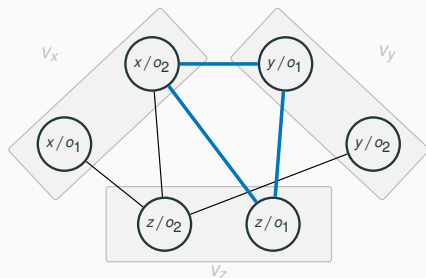
- The substitution consistency graph $\mathcal{H}(r, J)$ for a rule r and a set of facts J has **vertices** (candidate substitutions)

$$\mathcal{V}(\mathcal{H}(r, J)) = \{x/o \mid x \in \mathcal{X}(r), o \in \mathcal{O}\}$$

- and **edges** (consistent pairs)

$$\mathcal{E}(\mathcal{H}(r, J)) = \mathcal{V} \times \mathcal{V} \setminus (\mathcal{I}^\neq \cup \mathcal{I}^+ \cup \mathcal{I}^-)$$

Edges violating well-formedness or rule applicability are removed.



3-clique $\Leftrightarrow$ ground rule instance

- **Theorem** (Ståhlberg 2023): Sound (body literal arities ≤ 2), complete.
- Clique enumeration targets cyclic rules by keeping pairwise constraints explicit.
- No rule transformations necessary.

- Semi-naive evaluation only needs rule instances that use a newly derived fact.
- In the consistency graph, newly derived facts may create new edges:

$$\Delta\mathcal{E}_{i,r}^t = \mathcal{E}(\mathcal{H}(r, J_i^t)) \setminus \mathcal{E}(\mathcal{H}(r, J_i^{t-1}))$$

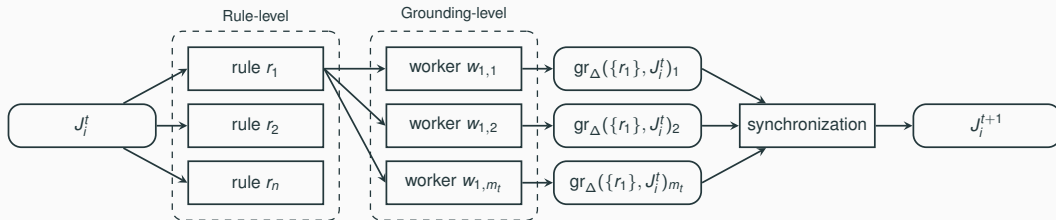
Algorithmic idea

- Seed clique enumeration at Δ -edges.
- Every newly relevant clique contains at least one Δ -edge.
- If a clique has several Δ -edges, emit it only from its owner edge.

Result

- Sound (body literal arities ≤ 2).
- Complete.
- Duplicate-free.

Two-Level Parallel Architecture

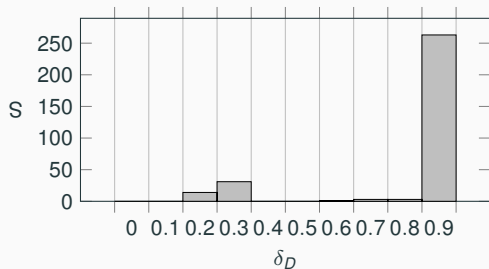


- **L1 (rule-level)**: evaluate rules of the same stratum concurrently; synchronize once at the merge.
- **L2 (grounding-level)**: split one expensive rule by distributing Δ -edges across workers.

- ▶ **Tyr:** our planner in C++ with Python bindings: <https://github.com/planning-and-learning/tyr>
- ▶ **Configurations:** Tyr- N with $N \in \{1, 2, 4, 8\}$ rule-level workers, $m_t = 1$ grounding worker by default.
- ▶ **Search:** lazy greedy best-first search with h_{FF} , boosted dual-queue (preferred / non-preferred).
- ▶ **Benchmarks:** Hard-To-Ground (HTG); 10 min, 16 GiB.

Where Does the Time Go? (Datalog Fraction)

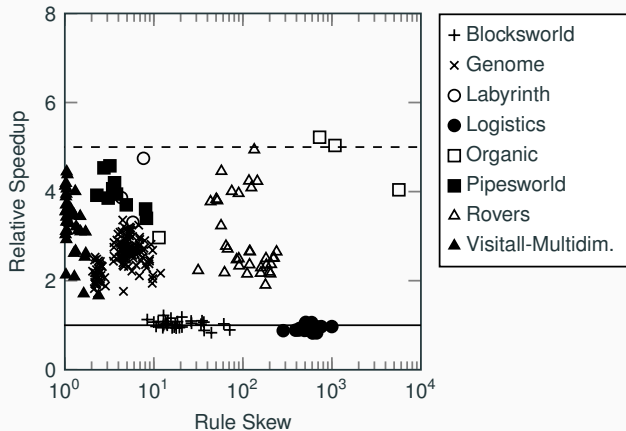
► Datalog fraction $\delta_D = T_D/T_{\text{tot}}$ — how much of the runtime is Datalog execution?



Amdahl's law

$$\delta_D = 0.9 \Rightarrow \text{speedup} \leq \frac{1}{1 - 0.9} = 10.$$

Observed Speedups vs. Rule Skew



Rule skew = max / median rule runtime across rules.

- ▶ Speedups mostly **2–4×**, up to **6×**.
- ▶ **Low skew** (Visitall ≈ 1) → up to **6×**.
- ▶ **High skew** (Blocksworld) → little gain.

- ▶ A parallel lifted planner via semi-naive Datalog evaluation.
- ▶ Δ -edge-anchored clique enumeration.
- ▶ Two levels of parallelism: rule-level (L1) and grounding (L2).
- ▶ Meaningful speedups: $2\text{--}4\times$, up to $6\times$ on 8 cores.