PlanForge: A Rust Planning Framework for Numeric Planning

Markus Fritzsche[1], **Martín Pozo**[2], **Mikhail Gruntov**[3], **Alexander Shleyfman**[4], **Daniel Gnad**[5,1]
[1]Linköping University, Sweden
[2]Universidad Carlos III de Madrid, Spain
[3]Technion, Israel
[4]Bar-Ilan University, Israel
[5]Heidelberg University, Germany
markus.fritzsche@liu.se, marcosmartin.pozo@uc3m.es, gruntovm@campus.technion.ac.il, alexash@biu.ac.il, daniel.gnad@uni-heidelberg.de

Abstract

PlanForge is a numeric planner written entirely in Rust. Its architecture follows Numeric Fast-Downward (Aldinger and Nebel 2017), which itself extends classical Fast Downward (Helmert 2006), but the translator, preprocessor, search, and heuristics are all reimplemented as crates in a single Rust workspace, removing the Python translation step inherited from Fast Downward. The framework supports the full numeric PDDL2.1 fragment; for IPC-2026 we submit configurations only to the simple-numeric-planning (SNP) tracks, using an admissible numeric LM-cut heuristic, numeric canonical domain abstractions, and a Metric-FF style relaxed-plan heuristic. A central technical change with respect to Numeric Fast-Downward is a tolerance-based canonicalization of numeric values that removes a class of false negatives in state-duplicate detection and missed heuristic-table lookups caused by floating-point round-off in numeric arithmetic. The code is available on GitHub but is currently highly experimental; a stable release on `crates.io` is planned.

Code—https://github.com/mrlab-ai/PlanForge

Introduction

Heuristic search with admissible heuristics is the dominant approach to optimal planning, both in classical (Helmert 2006) and in numeric (Kuroiwa et al. 2022) planning. For numeric planning, Numeric Fast-Downward (Aldinger and Nebel 2017) has become the de-facto reference codebase: it provides the SAS$^+$ pipeline of Fast Downward (Helmert 2006) adapted to PDDL2.1-style numeric fluents, and is the platform that most recent contributions in the area build on, including numeric LM-cut (Kuroiwa et al. 2022; Kuroiwa, Shleyfman, and Beck 2022) and numeric pattern databases (Gnad et al. 2025; Fritzsche et al. 2026a). Like Fast Downward, Numeric Fast-Downward is implemented in C++, with a Python translator that grounds and normalizes the PDDL input before the C++ search component takes over.

PlanForge is a new planning framework that we propose as a Rust-based alternative to Numeric Fast-Downward. It targets researchers who are more comfortable with Rust than with C++ for prototyping new heuristics and search techniques, and benefits from Rust's memory safety, single-binary deployment, and reproducible builds. The framework consists of a Cargo workspace of small crates—a translator, a preprocessor, a search component, a SAS$^+$ task representation, a CLI driver, and a portfolio driver—and produces a single statically linked binary. Compared to Numeric Fast-Downward, PlanForge is therefore "purely" in Rust: there is no Python translator and no scripting glue; the full PDDL-to-SAS$^+$ pipeline is a faithful port of the original `translate.py` into safe Rust.

PlanForge is openly available on GitHub at the URL above, but should at this point be treated as highly experimental: interfaces change between commits and a stable, versioned release on `crates.io` is planned but has not yet been cut. The framework stays close to Numeric Fast-Downward at the algorithmic level so that ports of existing techniques remain straightforward, and deviates only where we found the original behavior numerically fragile. The largest such deviation is a tolerance-based canonicalization of floating-point numeric values, which we describe next.

Numeric Canonicalization. In simple numeric planning (Hoffmann 2003; Kuroiwa et al. 2022), the numeric component of a state is a vector of IEEE-754 floating-point numbers (IEEE 2019) that is updated by repeatedly applying additive effects. IEEE-754 addition is not associative, so two action sequences that are mathematically equivalent can produce numeric components that differ in their lowest-order bits. In a planner this leaks into anywhere a numeric value is used as a hash or equality key: the state registry stores the same logical state under multiple slots, domain abstractions place a value into the wrong partition cell, and the FF-style numeric envelope fails to detect that a fixed point has been reached.

PlanForge fixes this by canonicalizing every numeric value before any hash or equality test. Concretely, with $\varepsilon = 10^{-12}$ we map $v \mapsto \mathrm{round}(v/\varepsilon) \cdot \varepsilon$, snapping every finite value to an integer multiple of ε; non-finite values are preserved unchanged so they remain distinguishable. Equality between two numeric values a, b is decided by $|a - b| \leq \max(\varepsilon, \varepsilon \cdot \max(|a|, |b|))$, i.e. a relative tolerance that degrades gracefully to the absolute one near zero. This canonical form is applied at the state-registry duplicate-detection boundary, at all domain-abstraction lookups, and in the FF-relaxation envelope-change test. We adopt this fix

because we observed many false negatives in Numeric Fast-Downward's duplicate detection: pairs of logically identical states whose numeric components differed only in their lowest-order bits were treated as distinct, inflating the closed list and letting A* re-expand effectively identical states. Under canonicalization those phantom duplicates collapse to a single registry entry, and a numeric value that is logically present in an abstract partition is always found by lookup. The resulting planner is closer in behavior to a planner using rational arithmetic, without paying its performance cost.

Heuristics

PlanForge is a general numeric planning framework: the translator, preprocessor, and search components accept the full numeric PDDL2.1 fragment, and additional heuristics for linear numeric planning and related extensions are being developed inside the same workspace. For IPC-2026 we submit configurations only for the simple-numeric-planning (Hoffmann 2003; Kuroiwa et al. 2022) (SNP) tracks; accordingly, we describe here only the heuristics that appear in those configurations. On top of an A* / greedy-best-first search core (Hart, Nilsson, and Raphael 1968; Helmert 2006) we use three of them.

Numeric LM-Cut. We reimplement the simple-numeric LM-cut heuristic $h^{\text{LM-CUT}}$ of Kuroiwa et al. (2022), which generalizes the classical LM-cut heuristic of Helmert and Domshlak (2009) to simple numeric planning. The implementation follows the configuration submitted to the IPC-2023 numeric track and we use $h^{\text{LM-CUT}}$ as a standalone admissible heuristic.

Numeric Canonical Domain Abstractions. Numeric domain abstractions are admissible abstraction heuristics built by counterexample-guided abstraction refinement (Seipp and Helmert 2018) generalized to numeric variables. PlanForge constructs a collection of such abstractions and combines them admissibly using the canonical heuristic, i.e. as the maximum over additive subsets. CEGAR refinement is driven by sequence flaws (Pozo, Torralba, and López 2024), ported from the classical to the numeric setting: both the progression and regression variants are implemented, and the submitted configurations use the progression variant. The implementation is a Rust port of the C++ implementation accompanying our companion paper (Fritzsche et al. 2026b), where we present the approach in detail and evaluate it empirically against numeric pattern databases.

Numeric FF. For non-admissible search we ship a Metric-FF style relaxed-plan heuristic (Hoffmann 2003). Each numeric variable carries a $(\max, \min)$ envelope through the relaxed planning graph, and a comparison-axiom fact becomes available once the envelope makes the comparison satisfiable. Combined with a dual-queue lazy open list and helpful-action preferred operators, this is our default guide for non-admissible search.

Implementation and Configurations

PlanForge is shipped as a Cargo workspace; the binaries used by the runner scripts below are `planforge` (single-stage) and `planforge-portfolio` (staged). Both are produced from the same workspace and share the translator, preprocessor, SAS$^+$, and search libraries, so heuristics defined in one place are available to both entry points. Memory and CPU budgets are enforced through `RLIMIT_AS` and `RLIMIT_CPU`, with a polled in-process check that lets the planner stop heuristic construction cleanly before any external supervisor fires.

Config	Track	Search	Heuristic
PlanForge-Agile	Agile	GBFS	h^{FF}
PlanForge-Sat	Sat	GBFS	h^{FF}
PlanForge-Opt-1	Opt	A* (portfolio)	$h^{\text{LM-CUT}} \to h^{\text{CDA}}$
PlanForge-Opt-2	Opt	A*	h^{CDA}

Table 1: Submitted SNP configurations. h^{FF}: numeric Metric-FF with helpful-action preferred operators on a dual-queue lazy open list. $h^{\text{LM-CUT}}$: simple-numeric LM-cut. h^{CDA}: canonical numeric domain abstractions. The Opt-1 portfolio runs $h^{\text{LM-CUT}}$ for 5 min / 7 GiB, then falls back to h^{CDA} on the cached SAS$^+$ task.

We submit four SNP configurations, one per track plus one alternative for the optimal track; Table 1 summarizes them. The four runner scripts in the repository root—`run-snp-agile`, `run-snp-sat-1`, `run-snp-opt-1`, and `run-snp-opt-2`—correspond one-to-one to these configurations; each takes the IPC `DOMAIN`, `PROBLEM`, and `PLAN` positional arguments and dispatches to `planner.sif`.

PlanForge-Agile / PlanForge-Sat. For the agile and satisficing SNP tracks we use the same single-stage configuration: greedy best-first search guided by the numeric-FF heuristic with helpful-action preferred operators, routed through a dual-queue lazy open list. Internally this is invoked as `planforge --search 'gbfs(ff())' --internal-run`; the `--internal-run` flag suppresses re-entrancy logic that is only useful when PlanForge is invoked from its own portfolio.

PlanForge-Opt-1. Our default optimal configuration is a two-stage portfolio. Stage 1 runs A* with the numeric LM-cut heuristic under a 5 min, 7 GiB budget. If it returns without a plan—timeout, graceful memory abort, or unbounded heuristic—the portfolio binary reuses the already translated SAS$^+$ task and invokes Stage 2: A* with canonical domain abstractions, given the remaining wall-clock budget and a 300 s construction budget for the abstraction collection.

PlanForge-Opt-2. The second optimal configuration is single-stage: A* with the same canonical-domain-abstraction collection used by Stage 2 of the portfolio—a 300 s construction budget, a 100 000-state per-abstraction size cap, and a one-million-state total collection cap.

References

Aldinger, J.; and Nebel, B. 2017. Interval Based Relaxation Heuristics for Numeric Planning with Action Costs. In *KI*, 15–28.

Fritzsche, M.; Gnad, D.; Gruntov, M.; and Shleyfman, A. 2026a. Managing Infinite Abstractions in Numeric Pattern Database Heuristics. In *AAAI*, 36227–36235.

Fritzsche, M.; Gruntov, M.; Shleyfman, A.; and Gnad, D. 2026b. Domain-Abstraction Heuristics for Simple Numeric Planning. In *SoCS*. To appear.

Gnad, D.; Alon, L.-O.; Weiss, E.; and Shleyfman, A. 2025. PDBs Go Numeric: Pattern-Database Heuristics for Simple Numeric Planning. In *AAAI*, 26507–26515.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR*, 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *ICAPS*, 162–169.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *JAIR*, 20: 291–341.

IEEE. 2019. IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019. IEEE.

Kuroiwa, R.; Shleyfman, A.; and Beck, J. C. 2022. LM-Cut Heuristics for Optimal Linear Numeric Planning. In *ICAPS*, 203–212.

Kuroiwa, R.; Shleyfman, A.; Piacentini, C.; Castro, M. P.; and Beck, J. C. 2022. The LM-Cut Heuristic Family for Optimal Numeric Planning with Simple Conditions. *JAIR*, 75: 1477–1548.

Pozo, M.; Torralba, Á.; and López, C. L. 2024. Gotta Catch 'Em All! Sequence Flaws in CEGAR for Classical Planning. In *ECAI*, 4287–4294. IOS Press.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *JAIR*, 62: 535–577.