nplanning: A Numeric Forward-Search Planner for the IPC 2026 Numeric Track

Daniel Lutalo

School of Computing, Australian National University, Canberra, Australia
daniel.lutalo@anu.edu.au

Abstract

nplanning is a self-contained C++17 forward state-space planner submitted by Team 6 to the Linear Numeric Planning (LNP) fragment of the satisficing and agile IPC 2026 numeric tracks. Because Simple Numeric Planning (SNP) is a sub-fragment of LNP, the same entries also cover the SNP domains. A thin PDDL front-end built on the `ppmajal` parser feeds an integrated grounder that simplifies the task and infers, per action and per numeric variable, the cost and monotonicity information that guides search. Search combines numeric-aware delete-relaxation heuristics, a chained-numeric relaxed-plan heuristic, width-based novelty exploration, and sound numeric state-dominance pruning. We document the four submitted configurations and the components they compose.

System Overview

nplanning is a single binary preceded by a small Java bridge. A PDDL domain and problem are parsed by `ppmajal`, the parsing and data-structure library underlying ENHSP (Scala, Haslum, and Thiébaux 2016), and serialised to a typed lifted-task description; the C++ core reads this, grounds it, and searches. The planner targets LNP: typed first-order STRIPS with linear numeric conditions and unconditional additive and assignment numeric effects, together with an instance metric. Successor generation evaluates conditions and effects exactly (including non-linear arithmetic where it occurs), so the planner is sound and complete on the full input, while the heuristics treat the linear fragment precisely and approximate the remainder. States are stored as a bit-vector of Boolean atoms plus a vector of `float32` numeric values; a per-task node bound derived from the state size keeps the search frontier inside the competition memory limit, and the satisficing and agile portfolios release memory between phases.

Grounding and Preprocessing

The grounder instantiates each action schema over type-correct object tuples, folds *static* predicates and functions (those no action modifies), and discards instances whose static preconditions are unsatisfiable. A delete-relaxation reachability fixpoint then removes actions whose positive propositional preconditions are unreachable from the initial state, and a final compaction pass drops atoms and numeric variables that no action, goal, or metric reads. Two structural analyses feed search:

- **Action costs.** The instance metric is compiled into a weighted sum over numeric variables; each action is assigned the net change it makes to that sum (constant and static-fluent sub-terms are folded to their values). Tasks with a non-linear or absent metric default to unit action cost.
- **Variable monotonicity.** Each numeric variable is classified as *increase-good*, *decrease-good*, *must-match*, or *irrelevant* by inferring its monotonicity from the linear structure of every action precondition and the goal (after pushing negations inward); variables used with conflicting directions, in equalities, or non-linearly fall back conservatively to *must-match*. This classification makes the dominance pruning sound.

Heuristics

All heuristics operate over the delete relaxation, with each numeric condition costed by *subgoaling*: the number of action firings needed to close a numeric gap is estimated as $\lceil gap \,/\, r \rceil$, where r is the largest single-action effect on the relevant variable in the helping direction (Scala, Haslum, and Thiébaux 2016; Scala et al. 2020).

- h_{add} (additive) extends the classical additive relaxation heuristic (Bonet and Geffner 2001; Hoffmann 2003) to numeric conditions, summing the subgoaling estimates of the goal conditions.
- h_{mrp} is a numeric relaxed-plan heuristic that propagates *action* costs through chains of supporting actions: each numeric subgoal is charged its firing count times the cost of the cheapest action that moves the relevant variable the right way. This yields a sharper gradient than h_{add} on domains where reaching a numeric condition requires a chain of producing actions.
- A relaxed-plan extraction marks *helpful* (preferred) actions (Hoffmann and Nebel 2001), augmented with actions that push an unsatisfied numeric goal variable in the needed direction.

Entry	Flag	Strategy
LNP-SAT-1	`--track sat`	three-phase portfolio
LNP-SAT-2	`--search anytime`	anytime restarting WA*
LNP-AGILE-1	`--track agile`	novelty-first portfolio
LNP-AGILE-2	`--search dq`	dual-queue GBFS

Table 1: The four submitted configurations.

Search

Search is forward best-first over the components below. *Dominance pruning*, applied by every engine, prunes a state whenever an equal-or-cheaper state with at-least-as-good values on every monotone variable has already been seen (holding must-match variables fixed) (Torralba and Hoffmann 2015); it auto-disables when too few variables discriminate states, avoiding quadratic frontier scans.

- **GBFS**: greedy best-first search on a single heuristic.
- **Dual-queue GBFS**: alternates a preferred-operator queue with the standard queue, boosting the preferred queue after improvements (Helmert 2006; Richter and Helmert 2009).
- **Novelty multi-queue**: a round-robin over preferred, width-1 novelty, width-2 novelty, h_{mrp}, and h_{add} queues, in the spirit of best-first width search (Lipovetzky and Geffner 2012, 2017); numeric features are discretised by order of magnitude so novelty applies to numeric state (Francès et al. 2017).
- **Anytime restarting search**: a GBFS $\rightarrow$ dual-queue $\rightarrow$ novelty cascade for a first plan, then restarting bounded weighted-A* (Pohl 1970) with descending weights, each run bounded by the incumbent plan cost so that only cheaper plans are kept (Richter, Thayer, and Ruml 2010; Richter and Westphal 2010).

Submitted Configurations

The four entries are two per track for the LNP fragment; all share one binary, selecting behaviour by command-line flags, and all cover the SNP domains as well. Satisficing entries run to the per-task time limit; agile entries return on the first plan. Table 1 summarises them.

Satisficing. LNP-SAT-1 is a three-phase portfolio that runs dual-queue GBFS/h_{add}, then GBFS/h_{mrp}, then the novelty multi-queue (the first two phases each receive a fixed fraction of the budget and the last takes the remainder), returning the first plan any phase finds. LNP-SAT-2 is the anytime engine: it obtains a first plan with the cascade above and then performs restarting bounded weighted-A* (weights $5, 3, 2, 1.5$) to reduce plan cost for as long as time remains.

Agile. LNP-AGILE-1 is tuned for first-plan speed with no plan-quality refinement: it runs the novelty multi-queue, then dual-queue GBFS, then plain GBFS, returning immediately on the first plan. LNP-AGILE-2 runs a single dual-queue GBFS/h_{add} with preferred operators: the fastest-to-first-plan engine on domains where helpful-action guidance alone suffices.

References

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence*, 129(1–2): 5–33.

Francès, G.; Ramírez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely Declarative Action Representations Are Overrated: Classical Planning with Simulators. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

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

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Lipovetzky, N.; and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*.

Pohl, I. 1970. Heuristic Search Viewed as Path Finding in a Graph. *Artificial Intelligence*, 1(3–4): 193–204.

Richter, S.; and Helmert, M. 2009. Preferred Operators and Deferred Evaluation in Satisficing Planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*.

Richter, S.; Thayer, J. T.; and Ruml, W. 2010. The Joy of Forgetting: Faster Anytime Search via Restarting. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Scala, E.; Haslum, P.; and Thiébaux, S. 2016. Heuristics for Numeric Planning via Subgoaling. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2020. Subgoaling Techniques for Satisficing and Optimal Numeric Planning. *Journal of Artificial Intelligence Research*, 68: 691–752.

Torralba, Á.; and Hoffmann, J. 2015. Simulation-Based Admissible Dominance Pruning. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*.