# IPC 2026 Numeric Track: The *Panino* Solver

**Giacomo Rosa**[1,*], **Jean Honorio**[1,*], **Nir Lipovetzky**[1,*], **Sebastian Sardiña**[2,§]

[1]The University of Melbourne, Australia
[2]RMIT University, Melbourne, Australia
[*]{g.rosa, jean.honorio, nir.lipovetzky}@unimelb.edu.au, [§]sebastian.sardina@rmit.edu.au

## Abstract

We describe *Panino*, a planner submitted to the IPC 2026 Numeric Track. *Panino* is a greedy best-first search planner built on top of *jpddlplus* and designed for linear numeric planning problems. Its main feature is a partitioned numeric novelty measure that adapts classical novelty to features induced by numeric and propositional subgoals. Literals appearing in action preconditions and goals induce both Boolean achievement features and distance features measuring progress toward their satisfaction. Novelty is then evaluated over conjunctions of size at most two, considering first achievements of propositional features and strict improvements of numeric distance features within heuristic partitions. The partitions are induced by the subgoaling-based ($h^{add}$) heuristic, which is also used as a tie-breaking component in search. We present two configurations: *Panino*, targeting the Agile track with a cost-maximising final tie-breaker to favour rapid solution discovery, and *Panino-anytime*, targeting the Satisficing track by using a cost-minimising tie-breaker and iterative cost-bounded restarts to improve solution quality. The resulting planner integrates numeric novelty with heuristic guidance in a lightweight single-planner, single-open-list search architecture for numeric planning.

## Introduction

The planner we present, the ***Panino*** (PArtitioned NumerIc NOvelty) solver, is a greedy-best first search solver which extends a subgoaling-based $h^{add}$ heuristic solver (Scala, Haslum, and Thiébaux 2016) via a new version of ***novelty*** (Lipovetzky and Geffner 2012, 2017) which we define as an adaptation for numeric planning problems. Our main contribution is the definition and integration of our definition of numeric novelty to the planner, thus we focus on introducing this aspect of the planner in this paper.

### Classical Novelty

Novelty in classical planning is defined over all conjunctions $t$ of size $\leq w$ over facts $f \in F$, where $w$ is set as a hyper-parameter. Thus, relevant conjunctions of size 1 are $\{\{f\} \mid f \in F\}$, conjunctions of size 2 are $\{\{f_i, f_j\} \mid f_i, f_j \in F\}$, and so on.

A state *satisfies* a conjunction, written $s \models t$, iff all facts in said conjunction are true in the state. A state is said to be ***novel*** in a search iff there exists a conjunction $t$ satisfied in state $s$, that has never been satisfied in any other state $s'$ generated before $s$ in the search. The ***novelty*** of a state is then the size of the smallest conjunction that is *novel* in that state.

### Numeric Novelty

**Novelty over subgoals.** A key aspect of the *numeric novelty* definition we introduce is that it is not directly defined over propositional or numeric variables in a numeric planning problem, but rather over the set of *subgoals* (atomic literals and propositional variables) derived from action preconditions and goal conditions present in a numeric planning problem. In classical planning, defining novelty over subgoals or facts is almost equivalent: any fact that does not appear in any action precondition or goal condition may be considered redundant to the search process, and is often removed by preprocessors during problem grounding. In numeric planning, there is a much more meaningful distinction, as literals can represent distinct linear numeric equalities or inequalities that describe specific bounds in the geometry of the problem, and thus provide a means of discretizing the problem.

**Propositional features.** The two key concepts on which our definition of numeric novelty relies on are ***propositional features***, and ***distance features***.

Let $\mathcal{S} = \mathcal{S}_B \cup \mathcal{S}_N$, where $\mathcal{S}_B$ is the set of propositional subgoals and $\mathcal{S}_N$ is the set of numeric subgoals. For each state $s$ and subgoal $g$, we define:

- ***Propositional feature.*** For $g \in \mathcal{S}_B \cup \mathcal{S}_N$,

$$\psi_g(s) \in \{0, 1\}, \quad \psi_g(s) = 1 \text{ iff } s \models g.$$

  Note that propositional features are created to evaluate the achievement of both propositional as well as numeric subgoals in a state $s$.

- ***Distance feature.*** For $g \in \mathcal{S}_N$, and given numeric literals in *normal form* (i.e. $\xi \trianglerighteq 0$),

$$\delta_g(s) = \begin{cases} -[\xi_g]^s & \text{if } g \text{ is an inequality,} \\ |[\xi_g]^s| & \text{if } g \text{ is an equality.} \end{cases}$$

  Where $[\xi_g]^s$ is the expression of numeric subgoal $g$ evaluated in state $s$. Since we use numeric literals in *normal form* (i.e. $\xi \trianglerighteq 0$), the definition evaluates the distance to that subgoal's achievement.

Thus, $\delta_g(s) = 0$ iff $s \models g$, and $\psi_g(s) = 1$ iff $\delta_g(s) = 0$.

In other words, *propositional features* map both propositional and numeric subgoals to a Boolean value that represents whether or not the subgoal is satisfied by state $s$; *distance features* evaluate left hand side of numeric subgoals in normal form in state $s$, and return the numeric distance required to satisfy said subgoal. Furthermore, if the subgoal is satisfied, they return a negative value, which represents the gap that exists between the subgoal's satisfaction and its lhs value in the state. This allows novelty to not just account for reductions in the minimum distance to a subgoal, but also increased gaps from the subgoal's satisfaction, increasing the set of novel states.

**Numeric conjunctions and novelty.** We limit ourselves to describe novelty over conjunctions of size $\leq 2$, as this is the limit we implement in our planner.

Conjunctions of size 1 contain either a propositional, or a numeric feature. A propositional feature is said to be ***novel*** when achieved for the first time. A distance feature is said to be ***novel*** when its best seen value is strictly improved (lowered), i.e., when for some numeric subgoal $g$, $\delta_g(s) < \min\{\delta_g(s') : s' \text{ previously generated}\}$.

Conjunctions of size 2 take two forms. The first one is a conjunction of two propositional features, $\{\psi_{g'}, \psi_{g''}\}$. Novelty is evaluated in the same way as with classical novelty: if a state satisfies both propositional features, and no other state visited previously in the search satisfied both propositional features. The second form is a conjunction of one propositional and one numeric feature, $\{\psi_{g'}, \delta_{g''}\}$. Such conjunction is novel in a state $s$ iff $\psi_{g'}$ is satisfied in $s$, and $\delta_{g''}(s)$ evaluates to a lower value than it does in all previously visited states $s'$ *that also satisfy* $\psi_{g'}$.

The *novelty* of a state is then the size of the smallest conjunction that is novel in that state. If no conjunction of size 1 or 2 is novel in that state, then we assign a novelty of 3.

For a more detailed look at the theory behind our definition of numeric novelty, see Rosa et al. (2026).

## Partitioned Numeric Novelty

In our planner, we implement ***partitioned novelty*** (Lipovetzky and Geffner 2017). A partition is a subset of generated states induced by a partitioning key: given a function $p$ mapping each state to a key, two states $s$ and $s'$ belong to the same partition iff $p(s) = p(s')$. Novelty is then evaluated only with respect to previously generated states in the same partition.

The numeric novelty definitions above are applied independently within each partition. Thus, a propositional feature is novel if it has not been achieved before in that partition, while a distance feature $\delta_g$ is novel if it strictly improves on the best value previously observed for $\delta_g$ in that partition. Analogously, conjunctions of size 2 are evaluated only against previously generated states in the same partition.

As a result, the same feature or conjunction may be novel in one partition even if it has already been seen in another. This preserves the exploratory role of novelty while conditioning it on the partitioning information used by the search.

**Heuristic configurations.** Our planner configurations adopt partitioned numeric novelty with conjunctions of size 1 and 2 as described in the previous section, using ***subgoaling-based*** $h^{add}$ (Scala, Haslum, and Thiébaux 2016) as partition function. We refer to this heuristic as $w_{add}$.

## Proposed Planner Configurations

We propose two different configurations, one for the satisficing track, and the other for the agile track of the competition. Our configurations target both simple numeric problems and linear numeric problems. They are implemented in the ***jpddlplus*** library (Scala 2024).

**Agile track configuration.** We propose *Panino*, a *greedy best-first search* solver with tie-breaking function $\{w_{add}, h^{add}, c_{max}\}$. That is, we use $w_{add}$ as first tie-breaker, then the same subgoaling-based $h^{add}$ that is used for novelty partitioning is also used as secondary tie-breaker. Finally, we break ties by *max-cost*, that is, preferring the expansion of plans with higher cost. This generally has a negative impact on plan costs, but the Agile track is focused on the time taken to solve problems; thus, we aim to find problem solutions more quickly on average by expanding longer plans.

**Satisficing track configuration.** Our Satisficing track variant, *Panino-anytime*, makes two changes to the Agile track planner. First, the heuristic configuration becomes $\{w_{add}, h^{add}, c_{min}\}$. We change the last tie-breaker to $c_{min}$ to favour the discovery of solutions with lower cost. The second difference is that we run the planner in *anytime* mode. Thus, once the solver finds an initial solution, if there is still time remaining, it will restart the search, setting the previous solution's cost as the maximum depth of the search (pruning any plan that exceeds this cost), thus seeking better solutions. This is performed iteratively as long as there is still time, or no better solution is found (in which case, the last found solution is supposed to be optimal).

## References


Lipovetzky, N.; and Geffner, H. 2012. Width and serialization of classical planning problems. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 540–545.

Lipovetzky, N.; and Geffner, H. 2017. Best-first width search: Exploration and exploitation in classical planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, volume 31.

Rosa, G.; Honorio, J.; Scala, E.; Lipovetzky, N.; and Sardina, S. 2026. A Notion of Width for Numeric Planning Problems. In *the ICAPS 2026 Workshop on Heuristics and Search for Domain-Independent Planning (HSDIP)*.

Scala, E. 2024. The JPDDLPLUS API. https://github.com/hstairs/jpddlplus. Accessed: 2026-05-05.

Scala, E.; Haslum, P.; and Thiébaux, S. 2016. Heuristics for numeric planning via subgoaling.