ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240990

Count-Based Novelty Exploration in Classical Planning

Giacomo Rosa[a,*] **and Nir Lipovetzky**[a,**]

[a]The University of Melbourne, Australia

Abstract. Count-based exploration methods are widely employed to improve the exploratory behavior of learning agents over sequential decision problems. Meanwhile, Novelty search has achieved success in Classical Planning through recording of the first, but not successive, occurrences of tuples. In order to structure the exploration, however, the number of tuples considered needs to grow exponentially as the search progresses. We propose a new novelty technique, classical count-based novelty, which aims to explore the state space with a constant number of tuples, by leveraging the frequency of each tuple's appearance in a search tree. We then justify the mechanisms through which lower tuple counts lead the search towards novel tuples. We also introduce algorithmic contributions in the form of a trimmed open list that maintains a constant size by pruning nodes with bad novelty values. These techniques are shown to complement existing novelty heuristics when integrated in a classical solver, achieving competitive results in challenging benchmarks from recent International Planning Competitions. Moreover, adapting our solver as the frontend planner in dual configurations that utilize both memory and time thresholds demonstrates a significant increase in instance coverage, surpassing current state-of-the-art solvers.

1 Introduction

Research on *width-based search methods* [16] has had a significant impact in planning, over the past decade, through the introduction of search algorithms which rely on *novelty* heuristics to induce an efficient exploration of the state-space. Novelty metrics achieve this by comparing a state's information content with that of states visited in the past. Width-based algorithms adopting Novelty alongside traditional heuristics have been central to improving state-of-the-art results in Classical Planning in recent years [14], with search performance often being attributed to a balance between exploration and exploitation, where Novelty drives the exploration while traditional heuristics direct exploitation. This does not come without limitations, as Lipovetzky and Geffner [16, 17] show that the complexity of computing novelty metrics needed to solve planning problems is exponential in their cardinality. In practice, this causes novelty metrics of cardinality greater than 2 to be computationally unfeasible, limiting the technique's effectiveness in domains that would benefit from a higher cardinality. The cardinality is connected with a hardness measure for Classical Planning known as *classical planning atomic width*. Multiple contributions have sought to address this limitation. Lipovetzky and Geffner [18] introduce *partition functions*, which subdivide planning problems into smaller sub-problems through the use of *partitioning heuristics* to control the direction of search and increase the number of novel nodes. Katz et al. [13] provide a definition of novelty of a state with respect to its heuristic estimate, providing multiple novelty measures which quantify the novelty degree of a state in terms of the number of novel and non-novel state facts. More recently, Singh et al. [27] introduce *approximate novelty*, which uses an approximate measurement of state novelty which is more time and memory efficient, proving capable of estimating novelty values of cardinality greater than 2 in practical scenarios. Relating Novelty with other concepts, such as *dominance pruning*, also constitutes an active area of research [5, 8].

All mentioned techniques limit themselves to the original idea of measuring state information content through the occurrence of tuples in the search history. Instead, we propose a count-based measure of state novelty, *classical count-based novelty*, which seeks to induce efficient exploration of the state space by making use of the additional information contained in the count of occurrences of tuples in the search history. This addresses shortcomings of the current Novelty framework (see [14]), which we refer to as *width-novelty* to distinguish from our contributions in this paper. Our proposed count-based metric is not limited by width-novelty's binary classification of novel information, providing a more fine-tuned separation of the degree of novelty of a state and maintaining its informedness without the risk of exhausting novel nodes. A key motivation behind our study is thus to obtain a more general novelty framework that can maintain its efficacy across diverse sets of problems in Classical Planning, such as domains that require higher atomic widths.

In this regard, we note that count-based exploration techniques are well studied in relation to the exploration-exploitation problem in Multi-Arm Bandits and Reinforcement Learning (RL) settings. Such algorithms record state visitation to obtain an *exploration bonus* used to guide the agent towards a more efficient exploration of the state-space, where algorithms such as MBIE-EB [28] achieve theoretical bounds on sample complexity in tabular settings. The focus of our research diverges from these methods, as we aim to discover a heuristic to control the order of state exploration in a Classical Planning context. Instead of state counts, we base our approach on the frequency of tuple events, inspired by work on width-novelty in the field of Classical Planning [16, 17, 19]. Still, our contributions provide a useful basis to connect count-based exploration across the two fields.

We also introduce algorithmic contributions in the form of a simple memory-efficient open list designed with count-based novelty in mind. Polynomial width-based planning algorithms prune nodes whose novelty cardinality is worse than a given bound to achieve a more efficient search [19]. Inspired on this idea, our contribution allows us to prune nodes with bad novelty values with a gradual and self-balancing cutoff without maintaining an explicit threshold value.

Finally, we demonstrate the effectiveness of our proposed planning

* rosag@student.unimelb.edu.au
** nir.lipovetzky@unimelb.edu.au

algorithms as fast but memory-intensive *frontend* solvers through an effective use of a *memory threshold*, which allows us to relate the progress of search to the amount of information we store from the history of a search. Many successful solvers such as FF, Probe or Dual-BFWS rely on such dual strategy [12, 15, 18], with the frontend of such solvers playing a key role in their performance. The performance of our proposed frontend planner could improve such solvers even further.

Our contributions consist of a new novelty technique, with theoretical analysis tying it to existing width-novelty measures, trimmed open list, a planner, $BFNoS$, which integrates these techniques, and a procedure to adapt BFNoS as an effective frontend planner in a dual strategy. We structure our paper as follows. In section 3 we introduce classical count-based novelty metrics for Classical Planning and provide related theoretical findings. We then propose a novel open list implementation to exploit classical count-based novelty more efficiently. Section 5 is divided into two components: we first compare the performance of solvers incorporating our proposed techniques, and then show the impact of our frontend solver when used in conjunction with different time-and-memory thresholds and backend solvers, providing state-of-the-art performance.

2 Background

Classical Planning The classical planning model is defined as $S = \langle S, s_0, S_G, A, f\rangle$, where S is a discrete finite state space, s_0 is the initial state, S_G is the set of goal states, and $A(s)$ denotes the set of actions $a \in A$ that deterministically map one state s into another $s' = f(a, s)$, where $A(s)$ is the set of actions applicable in s. We adopt a notation whereby, in a classical planning problem, a state is visited (generated) sequentially at each time-step t. Let $s_t \in S$ denote the t^{th} visited (generated) state in a search problem. We use $s_{0:t}$ to denote the sequence of $t+1$ states generated at time-steps $0, 1, ..., t$. A solution to a classical planning model is given by a plan, a sequence of actions $a_0, ..., a_{x_m}$ that induces a state sequence $s_{0:x_{m+1}}$ such that $a_{x_i} \in A(s_{x_i})$, $s_{x_{i+1}} = f(a_{x_i}, s_{x_i})$, and $s_{x_{m+1}} \in S_G$.

We use STRIPS planning language [9] to define a classical planning problem $P = \langle F, O, I, G\rangle$, where F denotes the set of boolean variables, O denotes the set of operators, $I \subseteq F$ is the set of atoms that fully describe the initial state, and $G \subseteq F$ is the set of atoms present in the goal state. An optimal plan consists of the shortest possible solution to a given problem P. In this research, we look at *satisficing* planners, that is, planners which are not constrained to searching for optimal plans, but rather aim for computing good-quality plans fast.

Width-Based Search *Best-First Width Search* (BFWS) [19] refers to a family of planners which adopt a *greedy best-first search* algorithm, using a novelty measure as first heuristic. A greedy best first search planner is a planner which visits nodes in the order specified solely by an *evaluation function* h, potentially breaking ties through the use of secondary heuristics. The main peculiarity of using a primary novelty heuristic comes from the fact that it is goal-unaware, thus prioritizing an efficient exploration of the state space over seeking states which are expected to be closer to the goal. The search is then directed to the goal through the use of secondary tie-breaking heuristics as well as *partition functions*, that is, evaluation functions h used to partition the set of states considered in the computation of novelty measures into disjoint subsets, ignoring occurrences of variables in states belonging to separate subsets.

Count-based exploration Count-based exploration methods have been studied to address the exploration-exploitation dilemma inherent in learning algorithms by allowing agents to prioritize actions that lead to states with uncertain or unexplored dynamics, thereby facilitating more effective learning of the environment's structure. This is often achieved by incorporating an exploration bonus added to the agent's reward upon visiting a state, encouraging the exploration of states with low visitation counts. Among the best-known examples is the UCB1 bandit algorithm [2], which performs a near-optimal balancing of exploration and exploitation in the stateless multi-armed bandit problem. This is achieved by selecting actions, referred to as the arms of a bandit, which maximize an *upper confidence bound*, the sum of the empirical average rewards $Q_t(i)$ of selecting arm i, and a confidence interval term $\sqrt{\frac{2\log N}{N(i)}}$, where $N(i)$ is the count of pulls of arm i, and N is count of total arm pulls.

3 Count Based Novelty

Classical count-based novelty operates over states that assign a value to a finite number of variables $v \in V$ over finite and discrete domains. In problems defined via STRIPS, without loss of generality, $V = F$ are boolean variables. Let V be the set of all variables, and $U^{(k)} = \{X \subseteq V \mid |X| = k\}$ the set of all k-element variable conjunctions. A tuple $u \in U^{(k)}$, specifically $u = \{v_1, v_2, \ldots, v_k\}$, represents a conjunction of k variables. Given a state s that assigns a boolean value to each variable in V, the value of the tuple u in state s, denoted $s(u)$, is defined as the conjunction of the values of the k variables in u, $s(u) = s(v_1) \wedge s(v_2) \wedge \cdots \wedge s(v_k)$, where $s(v_i)$ is the value of variable v_i in state s. We say $s(u)$ is *true* if all $v \in s(u)$ are *true*, and tuple u is *true* in s if $s(u)$ is *true*. Let $s_{0:t}(u)$ denote the sequence of values of tuple $u \in U^{(k)}$ in state sequence $s_{0:t}$, and let $U^{+(k)}(s) \subseteq U^{(k)}$ denote the set of tuples u in state s where $s(u)$ is *true*.

Definition 1 (Classical count-based novelty)**.** *The count-based novelty $c^U(s)$ of a newly generated state s at time-step $t+1$ given a history of generated states $s_{0:t}$ and set of variable conjunctions $U = U^{(k)}$ for some tuple size k is:*

$$c^U(s_{t+1}) := \min_{u \in U^+(s_{t+1})} (N_t^u(s_{t+1}))$$

Where $N_t^u(s_{t+1})$ counts the number of states $s_i \in s_{0:t}$ where $s_i(u) = s_{t+1}(u)$.

That is, for each tuple u that is *true* in s_{t+1}, we count the number of states $s_i \in s_{0:t}$ where $s_i(u)$ is *true*, and we select the minimum out of those counts.

Following prior work on Novelty [18], we also define a version of count-based novelty which uses *partition functions* to separate the search space into distinct sub-spaces.

Definition 2 (Partitioned classical count-based novelty)**.** *The partitioned count-based novelty $c^U(s)$ of a newly generated state s at time-step $t+1$ given partition functions $h_1, ..., h_m$ is:*

$$c^U_{h_1,\ldots,h_m}(s_{t+1}) := \min_{u \in U^+(s_{t+1})} (N^u_{t;h_1,\ldots,h_m}(s_{t+1}))$$

Where $N^u_{t;h_1,\ldots,h_m}(s_{t+1})$ counts the number of states $s_i \in \{s_{0:t} \mid h_1(s_i) = h_1(s_{t+1}) \wedge ... \wedge h_m(s_i) = h_m(s_{t+1})\}$ where $s_i(u) = s_{t+1}(u)$.

In other terms, we are obtaining tuple counts relative to the partition of previously generated states where $h_j(s_i) = h_j(s_{t+1})$ for all $1 \leq j \leq m$, as opposed to the full state history $s_{0:t}$. It trivially follows that $N^u_{t;h_1,\ldots,h_m}(s_{t+1}) \leq N^u_t(s_{t+1})$ and $c^U_{h_1,\ldots,h_m}(s_{t+1}) \leq c^U(s_{t+1})$.

3.1 Theoretical results

In this section, we justify the notion that classical count-based novelty achieves an efficient exploration of the state space that benefits planner performance, and present the mechanisms through which this is achieved. Firstly, we focus on the exploratory aspect of our heuristic, by detailing how size-1-tuple counts can be leveraged to direct the search towards lesser explored areas of the state space. We do so by exploiting a Hamming distance measure of a state to all previously visited states, as it provides an intuitively appealing means of quantifying how different a newly visited state is to the solver's visitation history. By demonstrating that information on size-1-tuple counts leads to improved bounds with respect to the Hamming distance of newly visited states in Theorems 3 to 6, we highlight the extent to which classical count-based novelty identifies under-explored areas of the state space. This exploratory aspect alone, however, does not validate the heuristic's effectiveness, as it fails to reveal whether the novel information is beneficial to the search. We address this aspect in Theorems 7 and 8 by using information on size-1-tuple counts and average Hamming distance of states to estimate the expected number of novel tuples. Groß et al. [8] show that novel tuples benefit search performance by indicating potential new paths towards the goal. Our results identify the two mechanisms through which classical count-based novelty increases the expectation of such novel tuples.

We define node $n_i = n_i(s_i)$ as referring to a state s_i, where the sequence $n_{0:t}$ corresponds to sequence $s_{0:t}$. The distinction between a node n_i and its corresponding state s_i lies in the equality operator: $n_i = n_j$ iff $i = j$, implying that $s_i = s_j$, whereas $s_i = s_j$ denotes the equality of all underlying variable values v in s_i and s_j. Crucially, throughout the entire section we assume that $U = U^{(1)} = V$, that is, we are only looking at counts over single-variable tuples. We thus simplify the tuple notation by denoting $s^i = s(v_i)$. Let $L = |s| = |V|$, and *Hamming distance* $H(n, n_j) = H(n(s), n_j(s_j)) = \sum_{i=0}^{L-1} 1_{s^i \neq s^i_j}$. We then define *normalized Hamming distance* as $\delta(n, n_j) = \frac{1}{L}(H(n, n_j)) = \frac{1}{L}\sum_{i=0}^{L-1} 1_{s^i \neq s^i_j}$, and the *average normalized Hamming distance* of a node n with respect to all nodes in $n_{0:t}$ as $\alpha_{0:t}(n) = \frac{1}{W}\sum_{i=0;n_i \neq n}^{t} \delta(n, n_i)$ where $W = t$ if $n \in n_{0:t}$ or $W = t+1$ otherwise, noting that in the first case we are skipping a node's comparison with itself.

Let the *empirical count distribution* be $\mu^i_t(s) = \frac{N^{v_i}_t(s)}{t+1}$, and $\mu^{min}_t(s) = \min_{i \in V}(\mu^i_t(s))$, noting that the minimum is over the entire set of variables $V = U^{(1)}$ rather than the set of *true* variables $U^{+(1)}$ in a state s used in Definition 1 and 2, and $0 \leq \mu^i_t(s) \leq 1$. We provide a justification of this change through Propositions 1 and 2, demonstrating a correspondence between empirical counts and Hamming distances over $U^{(1)}$ in binary vectors, and the same metrics over the set $U^{+(1)}$ in binary vectors that include negated variables. This allows us to align our results with a STRIPS representation that includes negated variables. For the set of L variables V, we define s_{neg} for states s over $V_{neg} = V \cup \{\neg v \mid v \in V\}$ such that $s_{\text{neg}}(v) = s(v)$, $s_{\text{neg}}(\neg v) = \neg s(v)$ for all $v_i \in V$. Since $H(s, s') = |\{i \mid s(v_i) \neq s'(v_i)\}|$, we define $H_{\text{true}}(s, s') = |\{i \mid s(v_i) \neq s'(v_i), s(v_i) = 1\}|$ and $H_{\text{false}}(s, s') = |\{i \mid s(v_i) \neq s'(v_i), s(v_i) = 0\}|$, noting that $H(s, s') = H_{\text{true}}(s, s') + H_{\text{false}}(s, s')$.

Proposition 1. *The Hamming distance $H(s, s')$ between states s and s' equals the true Hamming distance $H_{true}(s_{neg}, s'_{neg})$, considering only variables in s_{neg} that are true.*

Proof. Since for each variable $s(v_i) = 1 \in s$ there are two variables $s_{neg}(v_i) = 1 \in s_{neg}$ and $s_{neg}(\neg v_i) = 0$, and for each variable $s(v_i) = 0 \in s$ there are two variables $s_{neg}(\neg v_i) = 1 \in s_{neg}$ and $s_{neg}(v_i) = 0$, follows that $H_{\text{true}}(s_{neg}, s'_{neg}) = H_{\text{true}}(s, s') + H_{\text{false}}(s, s') = H(s, s')$ □

Proposition 2. *The empirical count $N^{v_i}_t(s)$ for any value $s(v_i)$ corresponds to the empirical count $N^{v_x}_t(s_{neg})$, where $x = i$ if $s(v_i) = 1$ and $x = j$ if $s(v_i) = 0$ where $s_{neg}(v_j) \equiv s_{neg}(\neg v_i)$.*

Proof. From the definition of s_{neg}, for every variable $s_j(v_i) = 0 \in s_{0:t}(v_i)$ we have that $s_{neg;j}(v_i) = 0$ and $s_{neg;j}(\neg v_i) = 1$. The proof for $s_j(v_i) = 1$ case is symmetrical. Proposition 2 follows. □

It then follows from Proposition 2 that selecting the minimum count $c^{V_{neg}}(s_{neg;t+1}) = \min_{v \in V}(N^v_t(s_{t+1}))$.

Theorem 3. *The average normalized Hamming distance $\alpha_{0:t}(s)$ of a state s to the $t+1$ states in history $s_{0:t}$ is upper bounded by:*

$$\alpha_{0:t}(s) \leq 1 - \mu^{min}_t(s)$$

Proof. The average Hamming distance of $s(v_i)$ with respect to the value of variable i in all states $s_j \in s_{0:t}$ is equivalent to

$$\frac{1}{t+1}\sum_{j=0}^{t} 1_{s^i_j \neq s^i} = 1 - \frac{1}{t+1}\sum_{j=0}^{t} 1_{s^i_j = s^i} = 1 - \frac{1}{t+1}N^{v_i}_t(s)$$
$$= 1 - \mu^i_t(s)$$

Thus we have

$$\alpha_{0:t}(s) = \frac{1}{t+1}\sum_{j=0}^{t}\frac{1}{L}\sum_{i=0}^{L-1} 1_{s^i_j \neq s^i} = \frac{1}{L}\sum_{i=0}^{L-1}\frac{1}{t+1}\sum_{j=0}^{t} 1_{s^i_j \neq s^i}$$
$$= \frac{1}{L}\sum_{i=0}^{L-1}(1 - \mu^i_t(s)) \leq \frac{1}{L}\sum_{i=0}^{L-1}(1 - \mu^{min}_t(s))$$
$$= 1 - \mu^{min}_t(s) \tag{1}$$

□

Parent-child average distance comparison We provide a set of results on the average normalized Hamming distances of a child node with respect to its parent node, and the impact that the count-based novelty of a node has on this value. Incorporating constraints on the changes between parent and child nodes enables us to obtain much tighter bounds compared to Theorem 3, reflecting the parent-child dynamic that exists between expanded and generated nodes. Let n^c and n^p be the child and parent node respectively, where an action $a \in A$ is performed on n^p to flip the value of e variables, which we refer to as the *effects*. Let n^c be a newly generated node n_t.

Theorem 4. *Lower and upper bounds for $\alpha_{0:t}(n^c)$ are given by:*

$$\alpha_{0:t}(n^p) - \frac{t-1}{t}\frac{e}{L} \leq \alpha_{0:t}(n^c) \leq \alpha_{0:t}(n^p) + \frac{t-1}{t}\frac{e}{L}$$

Proof. In the lower bound, all e effect variables change their corresponding valuation to match with all states in history except for the parent node, reducing Hamming distance to each state by 1 for each effect e. Parent and child states share all variable valuations except for the e effects, which change valuation from parent to child node. This yields, for all cases where $n' \in n_{0:t}$, $n' \neq n^c$ and $n' \neq n^p$

$$\delta(n^c, n') \geq \frac{1}{L}(H(n^p, n') - e) = \delta(n^p, n') - \frac{e}{L} \tag{2}$$

$$\delta(n^c, n^p) = \frac{1}{L}(H(n^p, n^p) + e) = \frac{e}{L} \tag{3}$$

We can redefine the average $\alpha_{0:t}(n^c)$ as

$$\alpha_{0:t}(n^c) = \frac{1}{t}\Big[\sum_{i=0; n_i \notin \{n^p, n^c\}}^{t} \Big(\delta(n^c, n_i)\Big) + \delta(n^c, n^p)\Big] \tag{4}$$

Since we define that $n^c = n_t$:

$$\alpha_{0:t}(n^p) = \frac{1}{t}\Big[\sum_{i=0; n_i \notin n^p}^{t-1} \Big(\delta(n^p, n_i)\Big) + \delta(n^c, n^p)\Big] \tag{5}$$

Substituting (2) and (3) into (4), noting that $\sum_{i=0; n_i \notin \{n^p, n^c\}}^{t} (\frac{e}{L}) = (t-1)\frac{e}{L}$, and then substituting (5) yields Theorem 4.

For the upper bound, we note that it is symmetrical in that in the upper bound all effects *e* are novel, that is, their variable valuation in n^c has never been observed in $n_{0:t-1}$. Thus we get $\delta(n^c, n') \leq \delta(n^p, n') + \frac{e}{L}$. Following the same procedure yields the upper bound. □

Theorem 5. *Given a minimum empirical count distribution $\mu = \mu_{t-1}^{min}(n^c)$, the upper bound $\alpha_{0:t}(n^c)$ with respect to μ is given by:*

$$\alpha_{0:t}(n^c) \leq \alpha_{0:t}(n^p) + \frac{t-1}{t}\frac{e - 2e\mu}{L}$$

Proof. Since μ is the minimum feature occurrence, acting as a constraint, upper bound occurs when all effects e have occurrence equal to μ, that is, the minimum possible occurrence they are allowed to have. Thus, for $t-1$ nodes $n' \in n_{0:t-1}$, $n' \neq n^p$, we have that $\delta(n^c, n') = \frac{1}{L}(H(n^p, n') + e)$ a total of $(1-\mu)\cdot(t-1)$ times, and $\delta(n^c, n') = \frac{1}{L}(H(n^p, n') - e)$ a total of $\mu \cdot (t-1)$ times. Proof follows from derivation in Theorem 4. □

Theorem 6. *Lower bound for $\alpha_{0:t}(n^c)$ when $\mu_{t-1}^{min}(n^c) = 0$ is given by:*

$$\alpha_{0:t}(n^c) \geq \alpha_{0:t}(n^p) - \frac{t-1}{t}\frac{e-2}{L}$$

Proof. In the lower bound, one effect is novel, and $e-1$ effects match all previous history except n^p. Thus $\delta(n^c, n') = \frac{1}{L}(H(n^p, n') - (e-1) + 1)$. Proof follows from derivation in Theorem 4. □

A comparison of the bounds in Theorem 4 with those in Theorems 5 and 6 demonstrates the relation between novelty count and Hamming distance through improved bounds, in terms of changes in the average distances from parent to child node, for nodes with a low empirical count N, which acts through the empirical count distribution μ. The upper bound in Theorem 5 details the main improvement, signalling greater potential of the child node being located in newer areas of the state space. Theorem 6 is a notable special case for novel variable valuations never encountered before, which guarantees an improvement of the lower bound through the novel information that could not have already been observed in the parent node. Through the recursive nature of Theorems 4 to 6, we also conclude that paths consisting of low count-based novelty nodes are more likely to exhibit rapidly increasing average Hamming distances, thus facilitating a quicker exploration of novel state spaces. We cannot establish a tighter lower bound in Theorem 5 because the least common feature might not be an effect, however modifying count-based novelty metrics to consider effect occurrences could overcome this limitation. Still, greater Hamming distances alone fail to explain how count-based novelty benefits search efficiency. We provide Theorems 7 and 8 to tie our results to prior theoretical contributions on novelty-based search (see [5, 8, 17]) through an analysis of the expected count of novel tuples of size *k* (*k*-tuples).

Estimating novel *k*-tuples Let history $s_{0:t}$ represent $t+1$ independent and uniformly distributed binary vectors of size L. A tuple is novel if its valuation in $s = s_{t+1}$ was not observed in any state in history $s_{0:t}$.

Theorem 7. *The expected number of novel tuples of size k found in $s = s_{t+1}$ given search history $s_{0:t}$ is given by:*

$$\binom{L}{k}\Big[1 - (1 - \alpha_{0:t}(s))^k\Big]^{t+1} \tag{6}$$

Proof. From equation (1) we can obtain $\alpha_{0:t}(s) = \frac{1}{(t+1)\cdot L}\sum_{j=0}^{t}\sum_{i=0}^{L-1} 1_{s_j^i \neq s^i} = \mathbb{E}_{s_j \in s_{0:t}, v_i \in V}[\mathbf{1}_{\{s_j(v_i) \neq s(v_i)\}}] = P(s_j(v_i) \neq s(v_i) \text{ for some } j, i)$. Thus, the probability that it has the same value becomes $1 - \alpha_{0:t}(s)$, and for a tuple of size k, the probability that any of its constituent variable values is different in s_j than in s is $1 - (1 - \alpha_{0:t}(s))^k$. Calculating the union for a tuple over the full history and multiplying by the number of possible tuples of size k yields the expectation in (6). □

Theorem 8. *The expected number of novel tuples of size k found in state $s = s_{t+1}$ given information on occurrence count $N = N_t^v(s)$ for some variable $v \in V$ and search history $s_{0:t}$ is given by:*

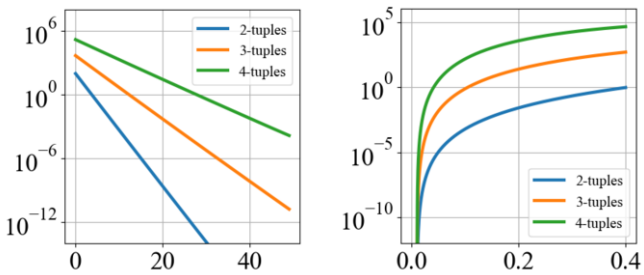
$$\binom{L-1}{k}\Big[1-(1-\beta_{0:t}(s))^k\Big]^{t+1} + \binom{L-1}{k-1}\Big[1-(1-\beta_{0:t}(s))^{k-1}\Big]^{N}$$

where $\beta_{0:t}(s)$ represents the average normalized Hamming distance after discounting the contribution of variable v:

$$\beta_{0:t}(s) = \frac{\alpha_{0:t}(s)\cdot L - (1 - \frac{N}{t+1})}{L-1}$$

Proof. The left-hand side component of the addition is given by equation (6) taken over tuples deriving from variables except for the variable v whose empirical count N we observe. The right-hand side component is given by the probability $1 - (1 - \beta_{0:t}(s))^{k-1}$ that, for some variable x other than v in a *k*-tuple containing v and in a state s_j where $s_j(v) = s(v)$, $s_j(x) \neq s(x)$. Thus, the tuple's valuation in s_j is different than in s. Taking a union over N states with matching v valuation and multiplying by the total number of tuples in s containing v yields the right-hand side component. Summing the two expectations proves the theorem. □

Theorem 7 reveals that, without count information, the expectation decreases exponentially with increasing t, rendering the measure effective only for small history sizes. Conversely, Theorem 8 introduces a component independent of t and exponential in count number N, emphasizing the crucial role of the minimum count function in identifying states likely to contain novel tuples, necessary to fulfill new action preconditions.



(a) $\mathbb{E}$[# novel k-tuple] vs. N (b) $\mathbb{E}$[# novel k-tuple] vs. $\alpha(s)$

Figure 1: $\mathbb{E}$[# novel k-tuple] according to Theorem 8. In (a) N is a variable, and in (b) $\alpha(s)$ is a variable. Otherwise, parameters are set as $L = 100$, $t = 50000$, $N = 5$, $\alpha(s) = 0.3$. A realistic $\alpha(s)$ value was determined through simulation[1].

We thus highlight the double role played by count-based novelty in inducing novel tuples, as shown in Figure 1: directly, through the greater probability that a novel tuple may contain a variable with low occurrence N, as well as indirectly, through the effect that a low count N induces a greater average Hamming distance compared to history $s_{0:t}$ (Theorems 3 to 6), which in turn increases the expected number of novel tuples. We note that this analysis is naïve in that it does not account for the structure present in domains, which alters the probability of co-occurrence of variables. We seek to emphasize that based on our proposed theoretical analysis, it is not the greater Hamming distance alone that leads to meaningful beneficial performance, as shown by Theorem 7, but rather its effect in conjunction with low variable occurrence in the second term of Theorem 8.

4 Trimmed Open List

Balancing the amount of memory occupied by low-rank nodes is a common strategy which allows for better ranked exploratory nodes to appear further down the search. Polynomial width-novelty planners prune nodes with novelty value greater than a threshold, as they are deemed not useful for the search. Similarly, count exploration methods generate many nodes with high counts, which are unlikely to ever be expanded. However, adopting a threshold as in the width-novelty case is unfeasible due to the granularity of the metric.

To address the challenge of high memory usage by poorly ranked nodes in the open list, we introduce the *Trimmed Open List* (Alg. 1). Built on a binary heap, this open list limits its growth by pruning less promising nodes when it exceeds a predefined size limit Z. This pruning process involves randomly selecting a leaf node, comparing its heuristic value with a new node n using the open list's comparison function, and then pruning or swapping nodes based on their heuristic values. A unique heapify-up operation is applied to the inserted leaf, which, unlike standard heaps, is not required to be the last element.

Furthermore, we developed a *Double Trimmed Open List* for heuristic alternation [23], accommodating dual open lists for node insertion under distinct heuristics and enabling alternate node retrieval. This variant employs the same pruning strategy but distinguishes itself by tracking each node's interaction with the open lists – either being popped or trimmed. A node becomes eligible for deletion when its interaction count equals the number of lists it is associated with, provided it is not in the closed list. This ensures a node is removed only when it is confirmed to be redundant, safeguarding against premature deletion crucial for the lazy expansion of successors.

[1] We remove a randomly generated root binary vector of size L=100 from the front of an open list, generate 4 children nodes uniformly at random flipping 3 binary variables each, and append each into the open list. We repeated the process to create 10000 nodes, and measured the average Hamming distance of the last 100 nodes, yielding an average value of ≈ 35.

Algorithm 1 Trimmed Open List

procedure TRIMMED OPEN LIST(new node N, heap H, heap size limit Z)
  $S \leftarrow$ size of H
  if $S < Z$ **then** ▷ If heap hasn't reached size limit Z
    insert N into H
    heapify-up(H) ▷ Reorder last element
  else
    $i \leftarrow$ uniformly random leaf index of H
    $O \leftarrow H[i]$ ▷ Node at random leaf index
    if N has a better heuristic value than O **then**
      $H[i] \leftarrow N$ ▷ Replace O with N
      heapify-up(H, i) ▷ Reorder element at index i
      discard O
    else
      discard N

5 Experiments

Our experiments were conducted using Downward Lab's experiment module [25], adhering to the IPC satisficing track constraints of 1800 seconds and 8 *GB* memory. Each test was ran on a single core of a cloud instance AMD EPYC 7702 2GHz processor. We implemented all proposed planners in C++, using LAPKT's [21] planning modules. For hybrid experiments, LAMA-First [22] and Scorpion-Maidu [3, 26] served as backend components, employing Fast-Downward (FD) [10] and the IPC2023 code repository [4], respectively. Except for Approximate-BFWS, BFWS variants utilized the FD grounder for grounding [11], however in problems where the FD grounder produces axioms (unsupported by LAPKT), LAPKT automatically switched to the Tarski grounder [7]. Approximate-BFWS exclusively used the Tarski grounder, following its initial setup and IPC-2023 configuration. We utilized IPC satisficing track benchmarks as in [27], selecting the latest problem sets for recurring domains. We conducted two sets of experiments. The first benchmarked our planners against the base BFWS(f_5) solver, evaluating the degree to which our proposed classical count-based novelty and trimmed open list techniques improve the coverage of BFWS(f_5) and its exploration efficiency, measured as the number of expansions required to find a solution. The second set of experiments compared our hybrid configurations to Dual-BFWS, Approximate-BFWS, LAMA-First, and a "first" version of the IPC-2023 satisficing track winner Scorpion-Maidu that runs its first iteration, in order to assess the coverage gains obtainable by adopting our proposed frontend solver alongside existing solvers in a dual configuration, relying on memory thresholds alongside more traditional time thresholds to trigger the frontend to fallback.

5.1 Count-based solvers

We define three new planning solvers to evaluate the performance of our proposed trimmed open list and classical partitioned count-based novelty techniques. All our solvers are based on the BFWS(f_5) search algorithm [18]. f_5 is the evaluation function $\langle w, \#g \rangle$ where w is the novelty measure and the goal counter $\#g$ counts the number of atomic goals not true in s. The novelty measure w is computed given partition functions $\#g$ and $\#r(s)$, that is $w_{\langle \#g, \#r \rangle}$ (see

[18]). We use w to refer to both width-novelty as well our proposed count-based novelty metric, adopting notation $f_5(X)$, where X is the chosen novelty measure w. Let W_x be the partitioned width-novelty metric with max-width $= x$ [18]. Let $C_1 = c^V$ be a partitioned classical count-based novelty metric over size-1 features $v \in V$. We define the following search algorithms:

- $BFWS_t(f_5(W_2))$: Standard BFWS(f_5) with max-width=2. Subscript *t* denotes use of a Single Trimmed Open List.
- $BFCS_t(f_5(C_1))$: A BFWS(f_5) solver where $w = C_1$. Subscript *t* denotes use of a Single Trimmed Open List.
- $BFNoS_t(f_5(C_1), f_5(W_2))$: Best-first search solver using a Double Trimmed Open List, employing the evaluation function $f_5(C_1)$ – with $w = C_1$ – for first open list and evaluation function $f_5(W_2)$ – with $w = W_2$ – for the second open list, alternating expansions between lists. We refer to this solver as BFNoS (Best First Novelty Search), as it uses multiple novelty heuristics.

The trimmed open list is capped at a constant depth $D = 18$ (maximum size $Z = 524,287$) determined by empirical testing. We note that empirically, small changes in depth ($D = 17$ or $D = 19$) did not alter coverage results beyond the deviation recorded in experiments.

		Coverage		
Model	**% Score**	**Total (1831)**	**IPC 2023 (140)**	**IPC 2018 (200)**
$BFWS(f_5(W_2))$	76.76%	1510	67	120
$BFCS(f_5(C_1))$	77.57%	1510	75	129
$BFWS_t(f_5(W_2))$	79.78%±0.13	1555±1.64	66±0.45	134±1.82
$BFCS_t(f_5(C_1))$	81.53%±0.33	1568±4.76	78±0.89	146±2.79
$BFNoS$	**83.32%±0.18**	**1600±3.90**	**87±1.10**	**149±1.34**

Table 1: % score and coverage comparison of proposed variants. % score is the average of the % of instances solved in each individual domain, calculated over all benchmark domains. Values represent the mean and include the standard deviation across 5 measurements.

Analysis of proposed techniques We refer to the results of our experiment in Table 1 showing significant improvement in BFWS(f_5)'s coverage. The trimmed open list's smaller memory footprint substantially boosts the instance coverage of BFCS and BFWS solvers alike, demonstrating the versatility of its node filtering mechanism even when dealing with the W_2 heuristic's narrower range.

BFCS$_t(f_5)$ outperforms BFWS$_t(f_5)$ in both coverage and normalized score. This advantage is especially evident in problems with high atomic widths, such as *Ricochet-Robots* [6] from IPC2023 [29], where BFCS$_t(f_5)$ consistently solves 19 out of 20 instances compared to the BFWS$_t(f_5)$'s single solve. This underscores count-based novelty's scalability in complex problems, contrasting with W_x metrics which have to revert to secondary heuristics after exhausting novel nodes, and demonstrates the *O(n)* count-based novelty variant's capacity to seek novel tuples of size > 1 as predicted by our analysis in Section 3.1. However, while C_1 can prioritize states with a higher expected number of 2-tuples, it cannot explicitly detect the presence of 2-tuples like W_2, and BFCS$_t(f_5)$ does show reduced coverage compared to BFWS$_t(f_5)$ in various domains, suggesting that W_2 and C_1 heuristics offer complementary strengths.

This synergy is exemplified by BFNoS$_t(f_5(C_1), f_5(W_2))$, which surpasses both in coverage due to its dual-heuristic approach. Notably, it also secures a significant 3.5% gain in normalized scores compared to BFWS$_t(f_5)$, indicative of the planner's enhanced cross-domain generalization. To our knowledge, this is the first instance demonstrating performance gains from combining distinct goal-unaware exploration heuristics in planning, as opposed to combining goal-aware exploitation heuristics as in [23]. On problems solved by both BFNoS and BFWS$_t(f_5)$, integrating the C_1 heuristic also reduces the number of node expansions required on average to solve instances as their size increases. This is illustrated in the upper-right quadrant of Figure 2, which highlights a general improvement in tackling large domains.

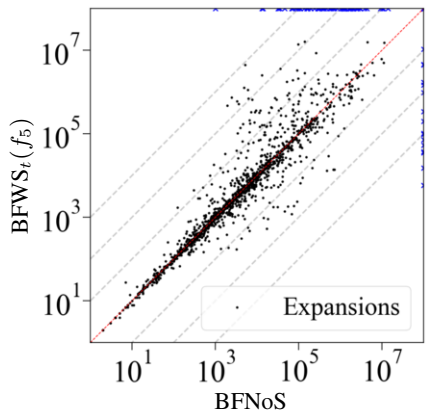


Figure 2: Number of nodes expanded across instances solved by BFNoS and BFWS$_t(f_5)$. Blue crosses represent instances not solved by at least one planner.

5.2 *Hybrid solvers with BFNoS frontend*

We adopt BFNoS$_t(f_5(C_1), f_5(W_2))$ as a frontend solver, capped by a 6 GB memory threshold and a time threshold close to the overall time limit, to enable backend fallback for all unresolved searches. We pair it with three backend planners from literature: the Dual-BFWS backend component (*BFNoS-Dual*), LAMA-First (*BFNoS-LAMA*), and the "first" version of Scorpion-Maidu (*BFNoS-Maidu-*h^2), in its IPC2023 configuration with the h^2*-preprocessor* [1]. These were chosen for their complementary heuristics to our frontend's f_5 partitioning, promoting diverse solution strategies to enhance coverage diversity. Frontend time thresholds are set to 1600 *sec* with BFNoS-Dual and BFNoS-LAMA, and 1400 *sec* for BFNoS-Maidu-h^2, to account for up to 180 *sec* of preprocessing allowance.

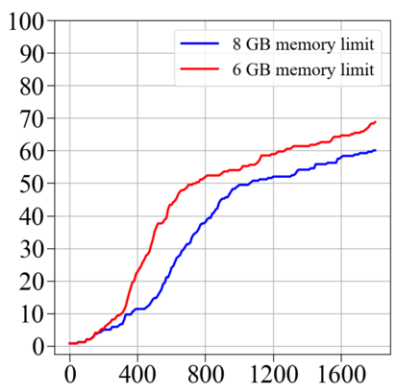


Figure 3: Cumulative % of all failures attributed to search-time memory failures (y-axis) vs. time of failure (*sec*) (x-axis), for BFNoS solvers with 6 *GB* and 8 *GB* memory and 1800 *sec* time limits.

Memory threshold Solvers tend to exhibit diminishing returns in the number of instances solved with respect to both time and memory. Thus, as memory usage increases while solving an instance, it is more likely that the instance will not be solvable within the defined memory constraint, indicating that the adopted heuristics are not effective for that particular problem. Novelty heuristics W_2 and

Domain	BFNoS	Dual-BFWS	Apx-BFWS (Tarski)	LAMA-First	Maidu	Maidu with h^2	BFNoS-Dual	BFNoS-LAMA	BFNoS-Maidu-h^2
agricola-sat18-strips (20)	15±0.0	12	**18±0.9**	12	12	13	15±0.5	15±0.5	15±0.5
airport (50)	**47±0.6**	46	**47±0.6**	34	38	45	46±0.6	46±0.5	46±0.6
caldera-sat18-adl (20)	18±0.0	**19**	**19±0.6**	16	16	16	16±0.0	17±0.5	18±0.0
cavediving-14-adl (20)	**8±0.5**	**8**	**8±0.5**	7	7	7	**8±0.0**	**8±0.0**	**8±0.5**
childsnack-sat14-strips (20)	1±1.1	**9**	5±0.6	6	6	6	8±0.0	6±0.0	6±0.5
citycar-sat14-adl (20)	**20±0.0**	**20**	**20±0.0**	5	6	6	**20±0.0**	**20±0.0**	**20±0.0**
data-network-sat18-strips (20)	17±0.6	13	**19±0.5**	13	16	16	16±0.8	15±1.1	16±0.8
depot (22)	**22±0.0**	**22**	**22±0.0**	20	**22**	**22**	**22±0.0**	**22±0.0**	**22±0.0**
flashfill-sat18-adl (20)	14±1.3	**17**	15±1.6	14	15	14	**17±0.5**	16±0.6	16±0.9
floortile-sat14-strips (20)	2±0.5	2	2±0.0	2	2	**20**	2±0.0	2±0.0	**20±0.0**
folding (20)	9±0.0	5	5±0.5	**11**	**11**	**11**	9±0.0	9±0.0	9±0.0
freecell (80)	**80±0.0**	**80**	**80±0.0**	79	**80**	**80**	**80±0.0**	**80±0.0**	**80±0.0**
hiking-sat14-strips (20)	**20±0.0**	18	**20±0.0**	**20**	**20**	**20**	**20±0.0**	**20±0.0**	**20±0.0**
labyrinth (20)	15±0.5	5	**18±0.5**	1	0	2	15±0.5	15±0.5	15±0.5
maintenance-sat14-adl (20)	**17±0.0**	**17**	**17±0.0**	11	13	13	**17±0.0**	**17±0.0**	**17±0.0**
mystery (30)	18±0.6	**19**	**19±0.0**	**19**	**19**	**19**	**19±0.0**	**19±0.0**	**19±0.0**
nomystery-sat11-strips (20)	14±0.8	**19**	14±0.5	11	**19**	18	**19±0.0**	15±0.6	17±0.0
nurikabe-sat18-adl (20)	16±0.6	14	17±0.5	9	11	16	17±0.6	17±0.0	**18±0.0**
org-synth-split-sat18-strips (20)	8±0.5	12	8±0.0	**14**	**14**	**14**	11±0.5	**14±0.0**	**14±0.9**
parcprinter-sat11-strips (20)	9±0.6	16	11±1.3	**20**	**20**	**20**	**20±0.0**	**20±0.0**	**20±0.0**
pathways (30)	26±0.9	**30**	28±1.1	23	25	25	**30±0.0**	27±0.8	27±0.7
pipesworld-notankage (50)	**50±0.0**	**50**	**50±0.0**	43	45	45	**50±0.0**	**50±0.0**	**50±0.0**
pipesworld-tankage (50)	43±1.6	42	**44±0.6**	43	43	43	43±0.8	43±0.5	43±0.6
recharging-robots (20)	**14±0.6**	12	**14±0.8**	13	13	13	**14±0.5**	**14±0.0**	**14±0.5**
ricochet-robots (20)	**20±0.5**	**20**	18±0.6	14	18	18	**20±0.0**	**20±0.0**	**20±0.0**
rubiks-cube (20)	5±0.0	6	5±0.6	**20**	**20**	**20**	5±0.0	**20±0.0**	16±0.6
satellite (36)	34±0.8	33	34±0.5	**36**	**36**	**36**	34±0.6	35±0.0	35±0.0
schedule (150)	149±1.3	**150**	149±1.3	**150**	**150**	**150**	149±0.7	**150±0.0**	**150±0.0**
settlers-sat18-adl (20)	13±1.5	7	12±0.7	17	**18**	**18**	12±0.5	17±0.0	17±0.5
slitherlink (20)	**5±0.6**	**5**	**5±0.7**	0	0	0	**5±0.5**	3±0.6	4±0.7
snake-sat18-strips (20)	**20±0.0**	17	**20±0.0**	5	14	14	**20±0.0**	**20±0.0**	**20±0.0**
sokoban-sat11-strips (20)	15±1.1	17	14±0.9	19	19	**20**	15±0.5	19±0.0	**20±0.0**
spider-sat18-strips (20)	17±1.3	16	16±1.1	16	16	17	**18±0.0**	**18±0.0**	**18±0.9**
storage (30)	**30±0.5**	29	**30±0.0**	20	25	25	29±0.5	29±0.0	29±0.6
termes-sat18-strips (20)	10±0.8	10	5±1.5	**16**	14	14	10±0.5	14±0.0	14±0.0
tetris-sat14-strips (20)	**20±0.0**	17	**20±0.0**	16	17	20	**20±0.0**	**20±0.0**	**20±0.0**
thoughtful-sat14-strips (20)	**20±0.0**	**20**	**20±0.2**	15	19	19	**20±0.0**	**20±0.0**	**20±0.0**
tidybot-sat11-strips (20)	**20±0.0**	18	**20±0.2**	17	**20**	**20**	**20±0.0**	**20±0.0**	**20±0.0**
transport-sat14-strips (20)	**20±0.0**	**20**	**20±0.2**	17	18	16	**20±0.5**	**20±0.0**	**20±0.0**
trucks-strips (30)	8±0.8	19	13±1.5	18	20	**22**	17±0.5	16±0.0	20±0.0
woodworking-sat11-strips (20)	**20±0.0**	**20**	12±1.1	**20**	**20**	**20**	**20±0.0**	**20±0.0**	**20±0.0**
Coverage (1831)	1600±3.9	1603	1606±3.9	1535	1590	1626	1641±1.9	1662±2.3	**1688±3.3**
% Score (100%)	83.32% ±0.18	83.23%	83.51% ±0.27	79.06%	82.84%	85.31%	86.23% ±0.09	87.87% ±0.17	**89.79% ±0.22**
Frontend % coverage share	-	-	-	-	-	-	97%	96%	94%

Table 2: Comparative performance analysis across various domains. *% score* is the average of the % of instances solved in each domain. *Frontend % coverage share* refers to the % of covered instances solved by the BFNoS frontend. Values for BFNoS variants and Approximate-BFWS represent the mean and include the standard deviation across 5 measurements. Domains that are fully solved by all planners are omitted but included in the appendix [24].

C_1 provide a quick evaluation of state novelty which, in combination with efficient partition functions, allow solvers to expand considerably more nodes per unit time than counterparts adopting more informed but expensive heuristics such as h_{ff} [12], albeit at a greater memory cost. In other words, they can reach the "flatter" region of the coverage-memory relation more quickly. We leverage this trait to introduce dual configuration planners in which the frontend seeks to prioritize coverage, but also fail as quickly as possible when memory usage reaches those flatter areas, accounting for different domains' characteristics with respect to memory usage. Only search-time memory usage is considered, as tested dual solvers do not fall-back to the backend when grounder errors occur. At 8 *GB* and 1800 *sec* limits, BFNoS hits half of its total failures by reaching the memory limit when half of the available time has passed, with memory failures constituting over 60% of the total (Fig. 3), making it a suitable frontend solver for our strategy.

Discussion Table 2 shows that BFNoS$_t(f_5(C_1), f_5(W_2))$ alone achieves comparable coverage to all baselines with the exception of the IPC2023 configuration of Scorpion-Maidu, which is the only baseline solver incorporating a preprocessor.

The hybrid solvers adopting a BFNoS$_t(f_5(C_1), f_5(W_2))$ frontend outperform all baselines, denoting a meaningful increase in coverage and normalized % score compared to their respective backends, particularly for the hybrid configuration with LAMA-First backend, which covers 62 and 127 more instances compared to BFNoS and LAMA-First respectively. BFNoS-Maidu-h^2 gains an edge on BFNoS-LAMA mainly thanks to the additional preprocessor in the backend, which only gets used if the frontend solver fails. Still, we note that our implementation of BFNoS-LAMA and BFNoS-Maidu-h^2 perform the grounding operation a second time for the backend, resulting in redundant computations which may use up significant time in hard-to-ground domains.

Our proposed frontend solver is the main component of all hybrid planners, being responsible for over 94% of all solved instances in all cases. We note that the combination with the Dual-BFWS backend performs more poorly than other hybrid configurations, yet this does not come as a surprise. The Dual-BFWS backend is a Novelty

planner which shares many more similarities with our own proposed frontend, most crucially the W_2 primary heuristic, thus causing more overlap in coverage between the frontend and backend. Still, even this version suffices to outperform tested benchmarks. We also note a significant increase in coverage of proposed hybrid planners on problem sets from IPC2023 and IPC2018, with BFNoS-LAMA covering 101 and 164 instances, and BFNoS-Maidu-h^2 covering 98 and 166 instances on average respectively.

In all cases, the implementation of hybrid planner configurations leveraging only two powerful solvers allows us to keep such dual solvers as simple as possible, which we argue helps us reduce the risk of overfitting to the set of available benchmarks.

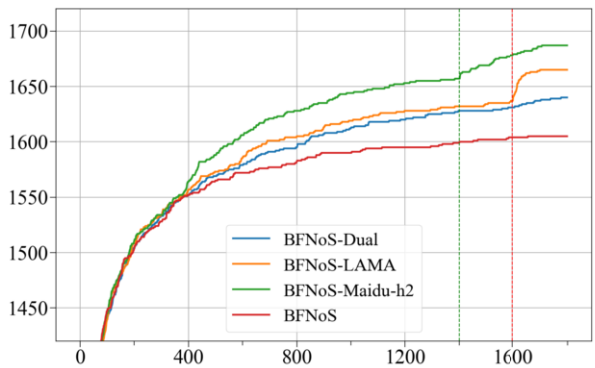


Figure 4: Instance coverage (y-axis) vs. time (*sec*) (x-axis). Comparison of BFNoS with the three presented hybrid configurations. The solvers are run in an identical configuration to previous sections, with BFNoS using the full 8 *GB* memory allowance when run on its own, and a 6 *GB* memory threshold when run as the frontend of hybrid configurations. The vertical lines signal the 1400 *sec* (green) and 1600 *sec* (red) time thresholds.

Coverage-over-time considerations We highlight the coverage-over-time benefits of adopting our proposed memory threshold strategy alongside BFNoS. The memory-threshold fallbacks of our proposed frontend start rapidly increasing shortly before the 400 *sec* mark (Fig. 3), and this fact is reflected in the increasing gap in instance coverage between the base BFNoS solver and dual configurations in Figure 4 beginning at that point in time. This trait is beneficial both for the average planning time of covered instances, as well as for satisficing planners, which gain more time to improve the quality of discovered plans. In this respect, memory-threshold-only variants of proposed dual solvers (Table 3) – that forego the use of time thresholds altogether – still perform competitively compared to benchmark solvers in Table 2. However, there still are limitations to the technique, as exemplified by the *Rubik's Cube* domain [20] from IPC 2023. BFNoS fails to solve 15 out of the 20 instances available, but does not reach the 6 *GB* memory threshold before the time limit, precluding fallback in memory-threshold-only dual configurations. This domain is the main contributor to the leap in instance coverage for BFNoS-LAMA and BFNoS-Maidu-h^2 that follows their respective time thresholds in Figure 4.

Hybrid Solver	Coverage	% Score
BFNoS-Dual$_{MO}$	1636±3.3	85.90%±0.14
BFNoS-LAMA$_{MO}$	1638±4.9	86.13%±0.26
BFNoS-Maidu$_{MO}$	1643±3.7	86.45%±0.17
BFNoS-Maidu-$h^2{}_{MO}$	1662±4.7	88.02%±0.21

Table 3: Coverage and % score of hybrid planner variants adopting only a 6 *GB* memory threshold across all domains. Subscript MO refers to memory-threshold-only versions. The BFNoS-Maidu planner is a variant of BFNoS-Maidu-h^2 that does not use the h^2 preprocessor with its Scorpion-Maidu backend. Values represent the mean and standard deviation across 5 measurements.

6 Conclusion

In this paper we introduce the concept of count-based novelty as an alternative novelty exploration framework in Classical Planning, showing that the arity-1 variant of our proposed metric is capable of effectively predicting states with novel k-tuples only using a constant number of tuples. We introduce the use of counts and Hamming distances to relate the exploratory behavior of count-based novelty to the existing body of knowledge on Novelty. We also propose single and double Trimmed Open List variants which allow us to upper bound the open list size by pruning nodes unlikely to be expanded. Our techniques used in the BFNoS solver demonstrate the effectiveness of combining distinct novelty metrics, achieving competitive coverage compared to state-of-the-art planners. Finally, we detail a dual-configuration strategy adopting a BFNoS frontend solver and introducing memory thresholds alongside customary time thresholds, justify the suitability of our proposed strategy, and demonstrate improved coverage performance compared to state-of-the-art planners.

Our work provides foundational knowledge on count-based novelty through basic solutions that mirror our theoretical analysis. Future directions may include alternative count-based variants over tuples or extracted features to guide exploration in general and domain-specific planners, as well as adaptation to the existing body of work blending Novelty and heuristic estimates. Our contributions also provide the basis to bridge Classical Planning with the broader paradigm of count-based exploration, benefiting knowledge transfer with related areas such as Reinforcement Learning. Trimmed open lists and memory thresholds also proved to be simple yet effective solutions, pointing at the potential for a deeper analysis of similar ideas in Classical Planning.

Acknowledgements

This research was supported by use of the Nectar Research Cloud and by the Melbourne Research Cloud. The Nectar Research Cloud is a collaborative Australian research platform supported by the NCRIS-funded Australian Research Data Commons (ARDC). We extend our gratitude to Anubhav Singh for providing the PDDL file modifications for the *Tidybot*, *Storage*, and *GED* domains used with the Tarski grounder, and to Augusto B. Corrêa for providing the information needed to run the 'First' version of Scorpion-Maidu using Fast Downward.

References

[1] V. Alcázar and A. Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 25, pages 2–6, 2015.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

[3] A. B. Corrêa, G. Francès, M. Hecher, D. M. Longo, and J. Seipp. Scorpion Maidu: Width search in the Scorpion planning system. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*, 2023.

[4] A. B. Corrêa, G. Francès, M. Hecher, D. M. Longo, and J. Seipp. Scorpion maidu satisficing ipc2023-classical. https://github.com/ipc2023-classical/planner8/tree/ipc2023-classical, 2023. Accessed: 2024-04-20.

[5] S. Dold and M. Helmert. Novelty vs. potential heuristics: A comparison of hardness measures for satisficing planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20692–20699, 2024.
[6] D. Fiser and R. Eifler. Ricochet Robots PDDL domain. https://github.com/ipc2023-classical/domain-ricochet-robots, 2023. Accessed: 2024-04-20.
[7] G. Francés, M. Ramirez, and Collaborators. Tarski: An AI planning modeling framework. https://github.com/aig-upf/tarski, 2018.
[8] J. Groß, A. Torralba, and M. Fickert. Novel is not always better: On the relation between novelty and dominance pruning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9875–9882, 2020.
[9] P. Haslum, N. Lipovetzky, D. Magazzeni, C. Muise, R. Brachman, F. Rossi, and P. Stone. *An introduction to the planning domain definition language*, volume 13. Springer, 2019.
[10] M. Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.
[11] M. Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5-6):503–535, 2009.
[12] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.
[13] M. Katz, N. Lipovetzky, D. Moshkovich, and A. Tuisov. Adapting novelty to classical planning as heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27, pages 172–180, 2017.
[14] N. Lipovetzky. Planning for novelty: Width-based algorithms for common problems in control, planning and reinforcement learning. *arXiv preprint arXiv:2106.04866*, 2021.
[15] N. Lipovetzky and H. Geffner. Searching for plans with carefully designed probes. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 21, pages 154–161, 2011.
[16] N. Lipovetzky and H. Geffner. Width and serialization of classical planning problems. In *ECAI 2012*, pages 540–545. IOS Press, 2012.
[17] N. Lipovetzky and H. Geffner. Width-based algorithms for classical planning: New results. In *ECAI 2014*, pages 1059–1060. IOS Press, 2014.
[18] N. Lipovetzky and H. Geffner. Best-first width search: Exploration and exploitation in classical planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.
[19] N. Lipovetzky and H. Geffner. A polynomial planning algorithm that beats LAMA and FF. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27, pages 195–199, 2017.
[20] B. C. Muppasani, V. Pallagani, and B. Srivastava. Solving the Rubik's Cube with a PDDL planner. In *ICAPS 2024 System's Demonstration track*, 2024.
[21] M. Ramirez, N. Lipovetzky, and C. Muise. Lightweight Automated Planning ToolKiT. http://lapkt.org/, 2015. Accessed: 2020.
[22] S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.
[23] G. Röger and M. Helmert. The more, the merrier: Combining heuristic estimators for satisficing planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 20, pages 246–249, 2010.
[24] G. Rosa and N. Lipovetzky. Count-based novelty exploration in classical planning - extended version. *arXiv preprint arXiv:2408.13719*, 2024.
[25] J. Seipp, F. Pommerening, S. Sievers, and M. Helmert. Downward lab, 2017.
[26] J. Seipp, T. Keller, and M. Helmert. Saturated cost partitioning for optimal classical planning. *Journal of Artificial Intelligence Research*, 67: 129–167, 2020.
[27] A. Singh, N. Lipovetzky, M. Ramirez, and J. Segovia-Aguas. Approximate novelty search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 349–357, 2021.
[28] A. L. Strehl and M. L. Littman. An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.
[29] A. Taitler, R. Alford, J. Espasa, G. Behnke, D. Fišer, M. Gimelfarb, F. Pommerening, S. Sanner, E. Scala, D. Schreiber, et al. The 2023 international planning competition, 2024.