Supplementary Material

Additional Discussion

Divergence-based Conditional Probability

$P(\pi \mid G)$ is unchanged from the previous definition in the paper, that is

$$P(\pi \mid G) := \frac{z(\pi) \cdot w(\pi)}{\sum_{\pi' \in \mathcal{O}} z(\pi') \cdot w(\pi')}$$

Thus, the full definition of the equation for $P(q \mid G)$ is

$$\begin{aligned} P(q \mid G) &= \sum_{\pi \in \mathcal{O}} P(q \mid \pi) P(\pi \mid G) \\ &= \sum_{\pi \in \mathcal{O}} \frac{P(q \mid \pi) \cdot z(\pi) \cdot w(\pi)}{\sum_{\pi' \in \mathcal{O}} z(\pi') \cdot w(\pi')} \end{aligned}$$

This definition is useful to more explicitly highlight the connection of this KL divergence-based conditional probability with the sampling and weighting steps described in the IRPL model.

Given for each fact $q_i \in O^F$ an associated Bernoulli distribution $P^i_G(q_i) := P(q_i \mid G)$, and a hard assignment $P^i_O(q_i) = 1$. Let $P_G(O^F) = \prod_{i:q_i \in O^F} P^i_G(q_i)$ and $P_O(O^F) = \prod_{i:q_i \in O^F} P^i_O(q_i) = 1$.

The KL divergence is simplified as follows:

$$\begin{aligned} D_{\mathrm{KL}}(P_O(O^F) \parallel P_G(O^F)) &= P_O(O^F) \log \frac{P_O(O^F)}{P_G(O^F)} \\ &= \log \frac{1}{P_G(O^F)} \\ &= \log \frac{1}{\prod_{i:q_i \in O^F} P^i_G(q_i)} \\ &= - \sum_{i:q_i \in O^F} \log P^i_G(q_i) \end{aligned}$$

GBFS Solvers

We evaluate several ***greedy best first search*** (GBFS) planners that expand nodes with the lowest heuristic value, breaking ties by unit cost. When using multiple heuristics, we apply ***heuristic alternation*** (Röger and Helmert 2010), which avoids order bias and ensures fair comparison.

Table 1 shows that our heuristics, pcp and $rpop$, improve coverage over a baseline using the ***Landmark heuristic*** (Richter, Helmert, and Westphal 2008), while maintaining similar expansion speed. Used alone, they underperform compared to h^{lm} as their $P(O \mid G)$ estimates resemble a more exploratory breadth-first search, as indicated by Claim 1, favoring nodes near the start and failing to reach deep plans in time. When used with landmarks, *pcp* and *rpop* improve the performance of the GBFS solvers, unlike using a breadth-first search second open list for exploration, that instead reduces the performance compared to the baseline h^{lm} variant.

Why not use $P(G \mid O)$ to estimate heuristics?

From our theoretical section, we discuss both maximizing $P(O \mid G)$, as well as $P(G \mid O)$, but experimentally we only estimate $P(O \mid G)$. The main reason for this is that our experimental contributions are modifications and extensions of existing techniques both in Planning and Goal Recognition. Previous work in GR as planning generally estimates $P(O \mid G)$ (Ramirez and Geffner 2010; Masters and Sardiña 2021; Wilken et al. 2024), and then derives the Bayesian posterior, as opposed to directly estimating $P(G \mid O)$. In planning, however, we would need to estimate multiple posteriors for different observations, such that their values can be ranked. As such, we cannot treat the Bayesian denominator as a normalizing constant, and must instead estimate $P(O)$, or $P(O|\neg G)$ from the underlying distribution of the $P(O \mid G)$. We believe that obtaining effective estimates and derivations for these quantities is better framed as future work, as it lies outside the core focus of our current study and may require new techniques.

We also describe how maximizing according to $P(G \mid O)$ is more "goal-directed" than $P(O \mid G)$. As such, our current contributions can provide a foundation to understand whether estimating $P(G \mid O)$ would be desirable if our estimates are inaccurate, or if we should prefer using $P(O \mid G)$ in such cases. This aspect, however, also requires further study, making it more suitable for future work.

Number of sampled relaxed plans for RPOP

We note that altering the number of sampled relaxed plans in the *RPOP* information extraction phase between 10, 100, 1000, and 10000 samples does not improve coverage meaningfully with this technique in empirical tests, so we did not focus on evaluating this aspect in performance benchmarks. We hypothesize that the quality of estimates obtained from relaxed plans, which approximate an inherently biased distribution rather than that of the non-relaxed problem, are not informative enough to gain any benefit from larger sample counts. It mainly affects the computational cost of the information extraction phase, as such we kept it at 100 to have a meaningful number of samples, while retaining low computation times.

We do note that in several problems, the *rpop-UTP* version of the heuristic re-weights sampled plans by placing most of the weight (>90%) on a single plan. Given the lower coverage but greater Agile score of this variant, it seems to induce a more "high-risk high-reward" behaviour: it prioritises relaxed plans that are estimated to lead to solutions with fewer node generations, solving problems earlier when they prove valid, but potentially failing more when they do not.

Magnitude of Changes in Number of Expansions

Figure 1 compares the magnitudes of the changes in number of expanded nodes to solve problems between BFWS($f5$) and our proposed variants. Results support our discussion in the main paper, whereby both our heuristics positively impact (reduce) the number of expansions on average across tested problems.

Sampling Supporter Actions

Extracting *fact observation probabilities* relies on obtaining sets of supporter actions that constitute relaxed plans, and evaluating the proportion of such relaxed plans that achieve each fact. We provide the algorithm for sampling supporter actions as presented in (Wilken et al. 2024) in Alg. 1. In red we highlight portions of the original algorithm we modified in our implementation in Alg. 2. First, we directly insert all goal condition facts into the set C of facts that must be achieved (line 5), and thus search for achievers for the entire goal statement in a single pass, as opposed to separately seeking achievers for each goal fact and merging achievers for different goal facts together at a later stage. This is done to improve overall computational cost, and account for actions that achieve more than one relevant fact for the entire goal statement. Our second modification involves selecting the set of candidate supporter actions. Rather than all supporter actions in the earliest layer of the RPG in which they occur, we select the set of valid supporter actions with minimum h-add heuristic value (line 13).

Algorithm 1 Supporter Action Sampling - Original.

```
1:  function SAMPLERELEVANTACTIONS(g_i, RPG, s_0, N)
2:      count ← {} ▷ Map from action to count in samples
3:      samples ← [ ]      ▷ List of generated supporter sets
4:      for i ∈ range(0, N) do
5:          C ← g_i                          ▷ Facts to be supported
6:          found ← ∅
7:          sups ← ∅
8:          for t = RPG.levels to 0 do
9:              newC ← ∅
10:             while |C| > 0 do
11:                 p ← C.pop()
12:                 psups ← ∅
13:                 for t_2 = 0 to t do
14:                     for all a ∈ RPG.level(t_2) do
15:                         if |add(a) ∩ {p}| > 0 then
16:                             psups ← psups ∪ {a}
17:                         end if
18:                     end for
19:                     if |psups| > 0 then
20:                         break
21:                     end if
22:                 end for
23:                 psups ← MINCOUNT(psups, count)
24:                 a ← RANDOM(psups)
25:                 found ← found ∪ {p}
26:                 C ← C \ {p}
27:                 sups ← sups ∪ {a}
28:                 count[a] ← count[a] + 1
29:                 for all n ∈ pre(a) do
30:                     if n ∉ s_0 ∧ n ∉ found ∧ n ∉ C then
31:                         newC ← newC ∪ {n}
32:                     end if
33:                 end for
34:                 for all r ∈ add(a) do
35:                     C ← C \ {r}
36:                     newC ← newC \ {r}
37:                 end for
38:             end while
39:             C ← C ∪ {newC}
40:         end for
41:         samples ← samples.add(sups)
42:     end for
43:     return samples
44: end function
```

Algorithm 2 Supporter Action Sampling - Ours.

```
1: function SAMPLERELEVANTACTIONS(G, RPG, s_0, N)
2:     count ← {} ▷ Map from action to count in samples
3:     samples ← [ ] ▷ List of generated supporter sets
4:     for i ∈ range(0, N) do
5:         for g_i ∈ G do C ← g_i ▷ Support all goal facts
6:         found ← ∅
7:         sups ← ∅
8:         for t = RPG.levels to 0 do
9:             newC ← ∅
10:            while |C| > 0 do
11:                p ← C.pop()
12:                psups ← ∅
13:                psups ← MINHADDSUPPORTERS(p, t, RPG)
14:                psups ← MINCOUNT(psups, count)
15:                a ← RANDOM(psups)
16:                found ← found ∪{p}
17:                C ← C \ {p}
18:                sups ← sups ∪{a}
19:                count[a] ← count[a] + 1
20:                for all n ∈ pre(a) do
21:                    if n ∉ s_0 ∧ n ∉ found ∧ n ∉ C then
22:                        newC ← newC ∪ {n}
23:                    end if
24:                end for
25:                for all r ∈ add(a) do
26:                    C ← C \ {r}
27:                    newC ← newC \ {r}
28:                end for
29:            end while
30:            C ← C ∪ {newC}
31:        end for
32:        samples ← samples.add(sups)
33:    end for
34:    return samples
35: end function
```

Experimental Considerations

Measuring Agile score

The Agile score is calculated for all planners using the overall process runtime provided by the Lab environment (Seipp et al. 2017). This is done to obtain less biased estimates, without relying on the search runtime measured by the planning libraries themselves, which may vary in when they start or stop measurements.

BFWS Variants

- BFWS($f5$): BFWS solver with evaluation function $f5 := \langle w_{\#r,\#g}, \#g\rangle$, where $\#r$ is the $\#r$ partition function from (Lipovetzky and Geffner 2017), and $\#g$ is the goal count heuristic. Remaining ties are broken by path length.
- BFWS($f5$)-pcp: BFWS($f5$) where third ties are broken by pcp, and remaining ties are broken by path length.
- BFWS($f5$)-$rpop_r$: BFWS($f5$) where third ties are broken by $rpop_r$, and remaining ties are broken by path length. BFWS($f5$)-$rpop_r$-UTP adopts $rpop_r$ with an additional UTP sample relaxed plan weighting function.
- BFWS($f5$)-Landmarks: BFWS solver with evaluation function $f5 := \langle w_{\#r,lm}, lm\rangle$, where lm is the Landmarks heuristic (Richter, Helmert, and Westphal 2008). Remaining ties are broken by path length.
- BFWS($f5$)-Landmarks-$rpop_r$: BFWS($f5$)-Landmarks where third ties are broken by $rpop_r$, and remaining ties are broken by path length. A key consideration is that it automatically reverts to using $\#g$ instead of the landmarks heuristic if it detects more than 100 facts in the goal condition. This is a practical consideration that is done to limit the number of problem partitions created by the planner, as traces used by $rpop_r$ start only at the most recent heuristic improvement. If there are too many improvements, then $rpop_r$ is restarted too often and does not inform the search. Since the number of landmarks is greater or equal to the number of goal facts, switching helps reduce this problem in such cases.
- BFWS($f5$)$_t$: BFWS($f5$) solver that substitutes the normal open list with a trimmed open list (Rosa and Lipovetzky 2024). This open list variant limits the maximum depth allowed of the binary heap, limiting memory usage. The maximum binary heap depth allowed by the open list is a hyperparameter, and is set to 18, following results in (Rosa and Lipovetzky 2024). All BFWS$_t$($f5$) variants are the same as variants described above, but also adopt a trimmed open list.
- BFWS($f5$)$_t$-RPOP$_r$-Dual: A ***dual-strategy*** solver, where a frontend solver attempts to solve the problem and, if it fails, it falls back to a backend solver. The frontend solver is BFWS($f5$)$_t$-RPOP$_r$, the backend is the backend solver of *Dual-BFWS* (Lipovetzky and Geffner 2017). This configuration is the same as *BFNoS-Dual* (Rosa and Lipovetzky 2024) albeit substituting the BFNoS frontend with our proposed BFWS($f5$)$_t$-RPOP$_r$. As such, like *BFNoS-Dual*, it adopts both time and memory thresholds to signal the fallback of the frontend solver. We set the time thresholds to 1500 sec and the memory threshold to 6000 MB. We use as a reference the 1600 sec and 6000 MB thresholds of *BFNoS-Dual*, but reduce the time threshold due to the faster solution times of our proposed solver compared to BFNoS.

Sources and commands for Benchmark Planners

BFWS, Dual-BFWS, Approximate BFWS. Run on LAPKT (Ramirez et al. 2015).

```
BFWS --grounder FD -d <domain>
    -p <problem> --search_type BFWS-f5

BFWS --grounder FD -d <domain>
    -p <problem> --search_type DUAL-BFWS

Approximate_BFWS --grounder Tarski
    -d <domain> -p <problem>
    --seed <seed>
```

BFNoS-Dual Run on LAPKT-BFNoS (Ramirez et al. 2025).

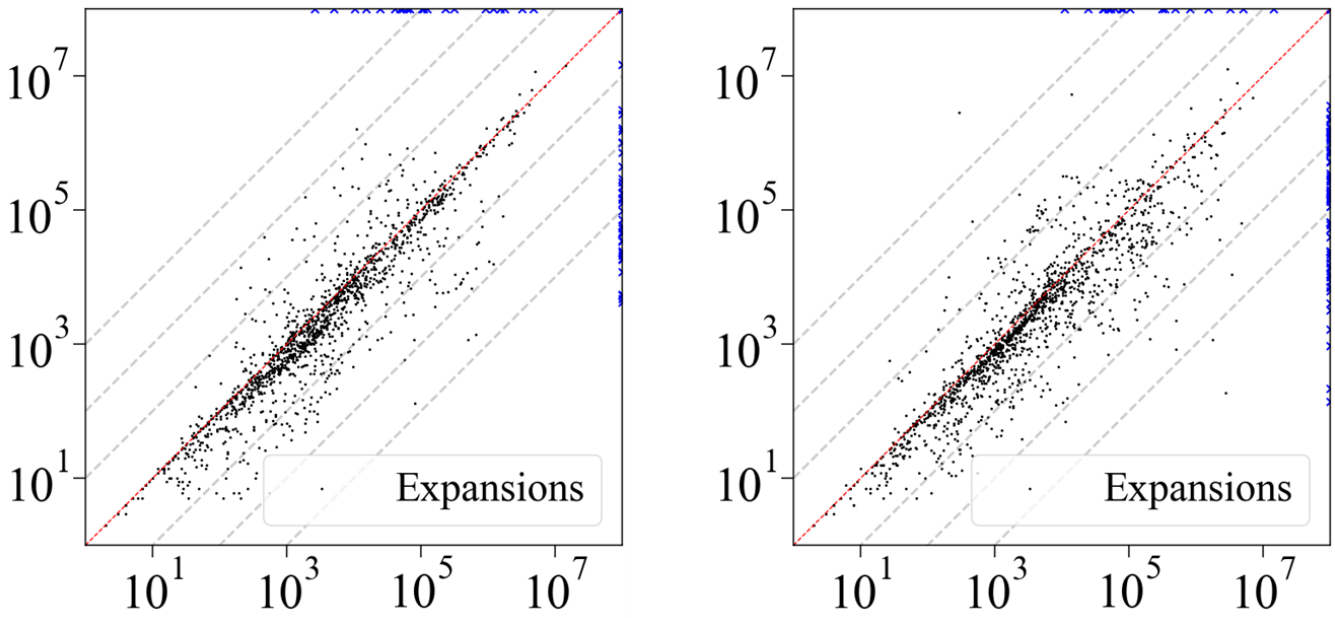
```
BFWS --grounder FD
    -d <domain> -p <problem>
    --search_type BFNOS
    --fallback_backend
    --backend_type DUAL-BFWS
    --time_limit 1600
    --memory_limit 6000
    --tol_seed <seed>
```

LAMA Run on the Fast Downward planning system (Helmert 2006).

```
--alias lama-first <domain> <problem>
```

Scorpion-Maidu We run a "first" version of Scorpion Maidu (Corrêa et al. 2023b), which halts after finding a solution rather than improving the plan, from the IPC-2023 branch of the code base (Corrêa et al. 2023a).

```
<domain> <problem>
--evaluator 'hlm=lmcount(
lm_factory=lm_reasonable_orders_hps(
    lm_rhw()),
transform=adapt_costs(one),pref=false)'
--evaluator
    'hff=ff(transform=adapt_costs(one))'
--search 'lazy(alt([single(hff),
single(hff, pref_only=true),
single(hlm),
single(hlm, pref_only=true),
type_based([hff, g()]),
novelty_open_list(novelty(width=2,
consider_only_novel_states=true,
reset_after_progress=True),
break_ties_randomly=False,
handle_progress=move)],
boost=1000),preferred=[hff,hlm],
cost_type=one,reopen_closed=false)'
```



(a) BFWS vs. BFWS-pcp

(b) BFWS vs. BFWS-rpop$_r$

Figure 1: Num Expansions (x-axis vs. y-axis) (lower is better). Comparing BFWS($f5$) with BFWS($f5$)-pcp and BFWS($f5$)-rpop$_r$, BFWS($f5$)-rpop$_r$ was ran with seed 0 for this comparison. Blue crosses on the border indicate problems not solved by one planner. The majority of data points are below the red line. This indicates a meaningful reduction in the average number of expansions across all tested instances.

Domain	hlm	pcp	rpop	hlm, gn	hlm, pcp	hlm, rpop
agricola-sat18-strips	12	0	0±0.0	12	3	12±0.5
airport	26	24	24±0.0	26	28	29±0.0
assembly	0	5	0±0.0	0	5	0±0.0
barman-sat14-strips	0	0	0±0.0	0	0	0±0.0
blocks	35	18	24±0.0	35	35	35±0.0
caldera-sat18-adl	9	4	2±0.0	8	10	9±0.5
cavediving-14-adl	6	6	7±0.0	7	7	7±0.0
childsnack-sat14-strips	0	0	0±0.0	0	0	0±0.0
citycar-sat14-adl	0	0	0±0.0	0	0	0±0.0
data-network-sat18-strips	0	1	0±0.0	0	1	0±0.0
depot	13	9	6±0.0	13	13	13±0.5
driverlog	19	11	10±0.0	18	19	18±0.0
elevators-sat11-strips	0	0	0±0.0	0	0	0±0.0
flashfill-sat18-adl	8	3	4±0.0	7	12	11±0.0
floortile-sat14-strips	0	0	0±0.0	0	0	2±0.0
folding	0	1	0±0.0	7	8	7±0.0
freecell	80	78	15±0.0	80	80	80±0.0
ged-sat14-strips	20	1	0±0.0	20	20	20±0.0
grid	4	2	4±0.0	4	4	5±0.0
gripper	20	6	6±0.0	20	20	20±0.0
hiking-sat14-strips	2	2	2±0.0	2	2	7±0.0
labyrinth	9	9	10±0.0	9	9	9±0.0
logistics00	28	12	14±0.0	28	28	28±0.0
maintenance-sat14-adl	9	0	0±0.0	9	11	9±0.0
miconic	150	50	40±0.0	150	150	150±0.0
movie	30	30	30±0.0	30	30	30±0.0
mprime	18	27	24±0.0	18	24	22±0.0
mystery	14	16	14±0.0	14	16	14±0.0
nomystery-sat11-strips	7	6	4±0.0	7	14	11±0.0
nurikabe-sat18-adl	11	7	4±0.0	10	11	9±0.6
openstacks-sat14-strips	20	0	0±0.0	20	19	20±0.0
organic-synthesis-split-sat18-strips	4	4	4±0.0	3	3	4±0.0
parcprinter-sat11-strips	0	6	0±0.0	12	12	12±0.0
parking-sat14-strips	0	0	0±0.0	0	0	0±0.0
pathways	5	5	4±0.0	5	6	4±0.0
pegsol-sat11-strips	20	15	16±0.0	19	19	19±0.0
pipesworld-notankage	29	25	19±0.0	32	38	31±1.4
pipesworld-tankage	15	13	11±0.0	20	22	23±0.6
psr-small	50	43	49±0.0	44	45	49±0.0
quantum-layout	18	8	1±0.0	18	18	18±0.0
recharging-robots	11	13	6±0.0	13	14	13±0.0
ricochet-robots	1	2	4±0.6	1	2	4±0.0
rovers	17	7	5±0.0	17	16	16±0.0
rubiks-cube	4	6	5±0.9	8	8	7±0.0
satellite	9	8	4±0.5	8	9	10±0.8
scanalyzer-sat11-strips	20	3	3±0.6	20	20	20±0.0
schedule	71	22	18±0.5	59	65	62±0.0
settlers-sat18-adl	3	0	0±0.0	1	1	2±0.6
slitherlink	2	1	1±0.0	3	4	3±0.0
snake-sat18-strips	20	1	0±0.5	3	3	3±0.0
sokoban-sat11-strips	6	3	2±0.0	8	7	7±0.0
spider-sat18-strips	13	3	0±0.0	13	13	16±0.7
storage	17	14	17±0.0	15	16	17±0.0
termes-sat18-strips	10	0	3±0.0	9	9	10±0.0
tetris-sat14-strips	20	0	0±0.0	19	18	19±0.5
thoughtful-sat14-strips	5	6	4±0.0	5	5	5±0.0
tidybot-sat11-strips	18	9	3±0.0	18	18	19±0.0
tpp	23	6	6±0.0	21	21	21±0.6
transport-sat14-strips	2	0	0±0.0	0	0	0±0.6
trucks-strips	4	2	4±0.0	4	6	8±0.0
visitall-sat14-strips	20	0	0±0.0	20	20	20±0.0
woodworking-sat11-strips	1	1	1±0.0	1	1	1±0.0
zenotravel	20	10	8±0.0	20	20	20±0.0
Coverage (1831)	1004	564	444 ±2.83	993	1038	1042 ±2.39

Table 1: Comparative performance analysis across GBFS variants, employing one alternating open list per heuristic. Values for rpop variants represent the mean and include the standard deviation across 3 measurements, using seeds from 0 to 2.

Domain	BFWS($f5$)	BFWS($f5$)-pcp	BFWS($f5$)-rpop$_r$	BFWS$_t$($f5$)	BFWS$_t$($f5$)-pcp	BFWS$_t$($f5$)-rpop$_r$
agricola-sat18-strips	10	12	12±0.6	11±0.8	12±0.6	13±0.5
airport	47	48	47±0.6	47±0.5	48±0.5	48±0.7
assembly	30	30	30±0.5	29±1.1	30±0.0	30±0.5
barman-sat14-strips	20	20	20±0.0	20±0.0	20±0.0	20±0.0
blocks	35	35	35±0.0	35±0.0	35±0.0	35±0.0
caldera-sat18-adl	15	16	18±0.6	18±0.5	19±0.0	19±0.5
cavediving-14-adl	7	7	7±0.0	7±0.0	7±0.0	7±0.0
childsnack-sat14-strips	0	1	0±0.0	0±0.5	1±0.0	0±0.0
citycar-sat14-adl	5	5	5±0.6	20±0.0	15±1.3	20±0.0
data-network-sat18-strips	11	12	17±0.8	17±1.1	17±0.8	20±0.5
depot	22	22	22±0.0	22±0.0	22±0.0	22±0.0
driverlog	20	20	20±0.0	20±0.0	20±0.0	20±0.0
elevators-sat11-strips	20	20	20±0.0	20±0.0	20±0.0	20±0.0
flashfill-sat18-adl	12	12	16±0.5	13±0.5	13±0.6	17±0.6
floortile-sat14-strips	2	1	2±0.0	1±0.0	1±0.0	2±0.0
folding	8	5	8±0.8	8±0.0	5±0.0	8±0.8
freecell	80	80	80±0.0	80±0.0	80±0.0	80±0.0
ged-sat14-strips	20	20	20±0.0	20±0.0	20±0.0	20±0.0
grid	5	5	5±0.0	5±0.0	5±0.0	5±0.0
gripper	20	20	20±0.0	20±0.0	20±0.0	20±0.0
hiking-sat14-strips	11	11	9±1.5	15±0.7	17±1.6	12±1.4
labyrinth	15	15	15±0.0	15±0.5	15±0.0	15±0.0
logistics00	28	28	28±0.0	28±0.0	28±0.0	28±0.0
maintenance-sat14-adl	17	16	16±0.5	17±0.0	16±0.0	17±0.5
miconic	150	150	150±0.0	150±0.0	150±0.0	150±0.0
movie	30	30	30±0.0	30±0.0	30±0.0	30±0.0
mprime	30	35	34±0.0	35±0.0	35±0.0	35±0.0
mystery	19	19	18±0.5	19±0.0	19±0.0	19±0.5
nomystery-sat11-strips	16	14	15±0.8	14±0.9	14±0.6	15±0.8
nurikabe-sat18-adl	16	17	16±0.8	16±0.0	17±0.0	16±1.5
openstacks-sat14-strips	20	19	20±0.0	20±0.0	20±0.6	20±0.0
organic-synthesis-split-sat18-strips	5	5	5±0.0	5±0.0	5±0.0	6±0.0
parcprinter-sat11-strips	9	9	16±0.0	8±0.8	9±0.6	16±0.0
parking-sat14-strips	20	20	20±0.0	20±0.0	20±0.0	20±0.0
pathways	24	27	29±0.5	25±0.5	26±1.3	29±0.8
pegsol-sat11-strips	19	20	19±0.0	19±0.0	20±0.0	19±0.0
pipesworld-notankage	50	50	50±0.0	50±0.0	50±0.0	50±0.0
pipesworld-tankage	44	44	44±0.8	43±0.6	43±1.0	44±0.8
psr-small	47	48	48±0.0	49±0.0	49±0.0	49±0.0
quantum-layout	20	20	20±0.0	20±0.0	20±0.0	20±0.0
recharging-robots	14	14	14±0.5	13±0.5	14±0.0	13±0.6
ricochet-robots	1	3	20±0.0	1±0.0	3±0.0	20±0.0
rovers	39	39	40±0.6	40±0.0	40±0.5	39±0.8
rubiks-cube	5	5	5±0.0	5±0.0	5±0.0	5±0.0
satellite	28	29	30±0.0	35±0.6	32±0.8	34±0.6
scanalyzer-sat11-strips	20	20	20±0.5	20±0.0	20±0.0	20±0.0
schedule	149	149	150±0.0	149±0.9	150±0.6	150±0.0
settlers-sat18-adl	10	11	10±1.1	11±0.5	12±0.6	14±0.6
slitherlink	4	5	4±0.5	4±0.0	5±0.6	4±0.5
snake-sat18-strips	18	18	19±1.0	19±0.5	19±0.5	18±0.0
sokoban-sat11-strips	15	15	14±0.6	15±0.5	15±0.6	16±1.1
spider-sat18-strips	14	15	12±1.4	14±0.0	15±0.0	12±1.4
storage	29	29	30±0.5	30±0.6	30±0.9	30±0.0
termes-sat18-strips	9	10	8±0.5	10±0.5	10±0.0	9±0.5
tetris-sat14-strips	20	20	20±0.0	20±0.0	20±0.0	20±0.0
thoughtful-sat14-strips	20	20	20±0.0	20±0.0	20±0.0	20±0.0
tidybot-sat11-strips	20	20	20±0.5	20±0.0	20±0.0	20±0.5
tpp	29	29	30±0.6	30±0.5	29±0.6	30±0.0
transport-sat14-strips	20	20	20±0.0	20±0.0	20±0.9	20±0.0
trucks-strips	8	8	9±0.8	9±0.9	8±0.5	10±1.1
visitall-sat14-strips	20	20	20±0.0	20±0.0	20±0.0	20±0.0
woodworking-sat11-strips	20	20	20±0.0	20±0.0	20±0.0	20±0.0
zenotravel	20	20	20±0.0	20±0.0	20±0.0	20±0.0
Coverage (1831)	1510	1526	1560±5.0	1557±3.7	1558±2.6	1599±3.8
% Score (100%)	76.77	77.63%	80.20%±0.35	79.90%±0.24	79.99%±0.19	82.94%±0.32

Table 2: Comparative performance analysis across the full set of benchmark domains. *% score* is the average of the % of instances solved in each domain. Values for solvers with randomized components represent the mean and include the standard deviation across 5 measurements, using seeds from 0 to 4.

Domain	Dual-BFWS	Apx-BFWS (Tarski)	LAMA-First	Scorpion Maidu	BFNoS-Dual	BFWS$_t$-hlm RPOP$_r$	BFWS$_t$-hlm RPOP$_r$-UTP	BFWS$_t$-hlm RPOP$_r$-Dual
agricola-sat18-strips	13	18±0.6	12	12	15±0.0	14±1.0	14±0.6	14±0.6
airport	46	47±0.6	34	38	46±0.6	47±0.0	47±0.0	47±0.0
assembly	30	30±0.0	30	30	30±0.0	30±0.0	30±0.6	30±0.0
barman-sat14-strips	20	20±0.0	20	20	20±0.0	20±0.0	20±0.0	20±0.0
blocks	35	35±0.0	35	35	35±0.0	35±0.0	35±0.0	35±0.0
caldera-sat18-adl	18	19±0.6	16	16	16±0.0	18±0.6	18±0.6	16±0.0
cavediving-14-adl	8	8±0.6	7	7	8±0.0	7±0.6	7±0.0	8±0.0
childsnack-sat14-strips	9	5±1.5	6	6	8±0.0	0±0.0	0±0.0	8±0.6
citycar-sat14-adl	20	20±0.0	5	7	20±0.0	19±0.6	20±0.0	19±0.6
data-network-sat18-strips	16	19±0.0	13	16	15±0.6	18±0.0	18±0.0	18±0.0
depot	22	22±0.0	20	22	22±0.0	22±0.0	22±0.0	22±0.0
driverlog	20	20±0.0	20	20	20±0.0	20±0.0	20±0.0	20±0.0
elevators-sat11-strips	20	20±0.0	20	20	20±0.0	20±0.0	20±0.0	20±0.0
flashfill-sat18-adl	17	15±1.0	14	15	17±0.0	17±0.6	15±0.6	18±0.6
floortile-sat14-strips	2	2±0.0	2	2	2±0.0	1±0.6	1±0.6	2±0.0
folding	5	5±0.6	11	11	9±0.0	8±0.6	9±1.0	8±0.6
freecell	80	80±0.0	79	80	80±0.0	80±0.0	80±0.0	80±0.0
ged-sat14-strips	20	20±0.0	20	20	20±0.0	20±0.0	20±0.0	20±0.0
grid	5	5±0.0	5	5	5±0.0	5±0.0	5±0.0	5±0.0
gripper	20	20±0.0	20	20	20±0.0	20±0.0	20±0.0	20±0.0
hiking-sat14-strips	18	20±0.0	20	20	20±0.0	20±0.0	20±0.0	20±0.0
labyrinth	5	18±0.6	1	0	15±0.6	15±0.0	15±0.0	15±0.0
logistics00	28	28±0.0	28	28	28±0.0	28±0.0	28±0.0	28±0.0
maintenance-sat14-adl	17	17±0.0	11	13	17±0.0	17±0.0	17±0.0	17±0.0
miconic	150	150±0.0	150	150	150±0.0	150±0.0	150±0.0	150±0.0
movie	30	30±0.0	30	30	30±0.0	30±0.0	30±0.0	30±0.0
mprime	35	35±0.0	35	35	35±0.0	35±0.0	35±0.0	35±0.0
mystery	19	19±0.0	19	19	19±0.0	19±0.0	19±0.0	19±0.0
nomystery-sat11-strips	19	14±1.0	11	19	19±0.0	15±0.6	17±0.6	19±0.0
nurikabe-sat18-adl	14	18±0.6	9	11	16±0.6	16±0.6	15±0.0	16±0.6
openstacks-sat14-strips	20	20±0.0	20	20	20±0.0	20±0.0	20±0.0	20±0.0
organic-synthesis-split-sat18-strips	12	8±0.6	14	14	12±0.0	5±0.0	4±0.5	12±0.0
parcprinter-sat11-strips	16	10±0.6	20	20	20±0.0	16±0.0	16±0.6	20±0.0
parking-sat14-strips	20	20±0.0	20	20	20±0.0	20±0.0	20±0.0	20±0.0
pathways	30	29±1.5	23	25	30±0.0	30±0.6	29±0.6	30±0.0
pegsol-sat11-strips	20	20±0.0	20	20	20±0.0	20±0.0	20±0.0	20±0.0
pipesworld-notankage	50	50±0.0	43	45	50±0.0	50±0.0	50±0.0	50±0.0
pipesworld-tankage	42	45±0.6	43	43	43±0.6	48±1.0	47±1.0	48±1.0
psr-small	50	50±0.0	50	50	50±0.0	49±0.0	49±0.0	50±0.0
quantum-layout	20	20±0.0	20	20	20±0.0	20±0.0	20±0.0	20±0.0
recharging-robots	11	14±1.0	13	13	14±1.0	14±0.0	14±0.0	14±0.6
ricochet-robots	20	18±0.0	14	18	20±0.0	20±0.0	20±0.0	20±0.0
rovers	39	40±0.6	40	40	40±0.6	39±1.0	40±0.0	39±1.0
rubiks-cube	6	5±0.6	20	20	5±0.0	5±0.0	5±0.0	5±0.0
satellite	32	34±0.0	36	36	32±1.0	34±0.6	33±0.6	33±1.2
scanalyzer-sat11-strips	20	20±0.6	20	20	20±0.0	19±0.0	19±0.6	19±0.0
schedule	150	150±0.0	150	150	149±0.6	150±0.0	150±0.0	150±0.0
settlers-sat18-adl	7	12±0.6	17	18	11±0.6	19±0.6	19±0.6	19±0.6
slitherlink	6	5±0.6	0	0	6±0.6	7±0.0	5±0.6	7±0.0
snake-sat18-strips	17	20±0.0	5	14	20±0.0	19±0.0	18±0.0	19±0.0
sokoban-sat11-strips	18	15±0.0	19	19	16±0.6	15±0.6	13±1.0	16±0.6
spider-sat18-strips	16	17±1.2	16	16	18±0.0	19±0.6	19±1.5	19±0.0
storage	30	30±0.0	20	25	30±0.0	30±0.6	30±0.5	30±0.0
termes-sat18-strips	10	5±2.0	16	14	10±0.6	10±0.6	10±0.6	10±0.0
tetris-sat14-strips	17	20±0.0	16	17	20±0.0	20±0.0	20±0.0	20±0.0
thoughtful-sat14-strips	20	20±0.0	15	19	20±0.0	20±0.0	20±0.0	20±0.0
tidybot-sat11-strips	18	20±0.0	17	20	20±0.0	19±0.0	19±0.0	19±0.0
tpp	30	30±0.0	30	30	30±0.0	30±0.0	30±0.0	30±0.0
transport-sat14-strips	20	20±0.0	17	18	20±0.0	20±0.0	20±0.0	20±0.0
trucks-strips	19	13±1.0	18	20	18±0.0	9±0.6	9±0.6	18±0.0
visitall-sat14-strips	20	20±0.0	20	20	20±0.0	20±0.0	20±0.0	20±0.0
woodworking-sat11-strips	20	13±1.5	20	20	20±0.0	20±0.0	20±0.0	20±0.0
zenotravel	20	20±0.0	20	20	20±0.0	20±0.0	20±0.0	20±0.0
Coverage (1831)	1607	1611±3.5	1535	1591	1641±0.6	1621±3.2	1616±2.1	1655±1.5
% Score (100%)	83.56%	83.83%±0.17	79.07%	82.92%	86.23%±0.06	84.59%±0.25	84.22%±0.13	86.99%±0.08
Agile score	1200.9	1233.7±0.24	1192.3	1206.4	1173.3±3.5	1229.4±3.4	1236.0±1.9	1232.6±2.9

Table 3: Comparative performance analysis across the full set of benchmark domains. *% score* is the average of the % of instances solved in each domain. Values for solvers with randomized components represent the mean and include the standard deviation across 3 measurements, using seeds from 0 to 2.

References

Corrêa, A. B.; Francès, G.; Hecher, M.; Longo, D. M.; and Seipp, J. 2023a. Scorpion maidu satisficing ipc2023-classical. https://github.com/ipc2023-classical/planner8/tree/ipc2023-classical.

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

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

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

Masters, P., and Sardiña, S. 2021. Expecting the unexpected: Goal recognition for rational and irrational agents. 297:103490.

Ramirez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. 1121–1126.

Ramirez, M.; Lipovetzky, N.; Singh, A.; and Muise, C. 2015. Lightweight Automated Planning ToolKiT. http://lapkt.org/. Accessed: 2025.

Ramirez, M.; Lipovetzky, N.; Singh, A.; Muise, C.; and Rosa, G. 2025. Lightweight Automated Planning ToolKiT - BFNoS Planners. https://github.com/grosa97/LAPKT-BFNoS. Accessed: 2025.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *AAAI*, volume 8, 975–982.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 20, 246–249.

Rosa, G., and Lipovetzky, N. 2024. Count-based novelty exploration in classical planning. In *Proceedings of the European Conference on Artificial Intelligence*, volume 392, 4181–4189.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward lab.

Wilken, N.; Cohausz, L.; Bartelt, C.; and Stuckenschmidt, H. 2024. Fact probability vector based goal recognition. In *ECAI 2024*. IOS Press. 4254–4261.