

A Proof of Theorem 4.1

PROOF: Denote by \mathcal{S} the set of all states and by H_n^* the set of states whose value is h^* after n updates: $H_n^* := \{s \mid s \in \mathcal{S}, G_n(s) = h^*(s)\}$. For convenience, we start counting the update iterations with $n = 0$. The initial lookup-table G_0 is arbitrarily initialized. We show by induction that $H_n^* \supseteq \{s \mid s \in \mathcal{S}, h^*(s) \leq n\}$.

Induction basis: After iteration $n = 0$, all goal states have the value 0, so, $H_0^* \supseteq \{s \mid s \in \mathcal{S}, h^*(s) = 0\}$.

Induction step: s is a state with $h^*(s) = n$. Then s has a successor s' with $h^*(s') = n - 1$. By induction hypothesis, we have $H_{n-1}^* \supseteq \{s \mid s \in \mathcal{S}, h^*(s) \leq n - 1\}$ and $s' \in H_{n-1}^*$. Thus, there is a path P from s' to the goal with G_{n-1} decreasing by 1 in each step. A GBFS run on s generates s' when expanding s , and afterwards follows P (or another path of the same length) resulting in n expansions. Hence $H_n^* \supseteq \{s \mid s \in \mathcal{S}, h^*(s) = n\}$. The same argument applies by induction assumption to all states t where $h^*(t) < n$: GBFS follows a direct path to the goal. So we have $t \in H_n^*$, and hence $H_n^* \supseteq \{s \mid s \in \mathcal{S}, h^*(s) \leq n\}$ as desired.

In all updates, GBFS run on a dead-end state s proves that s is a dead-end. Thus, all states s with infinite h^* value also satisfy $h^*(s) = G_n(s)$, for all n . This proves the claim. \square

B Adapting NN Training

B.1 Adaptations for h^{IL}

Only minor changes are needed to compare with h^{IL} , which like our heuristic functions is based on per-instance learning. Ferber, Helmert, and Hoffmann (2020) generated training data for up to 400 hours, on a single CPU core. Then they trained for up to 48 hours using 4 CPU cores and 12 GB of memory. This exceeds our resource limits by far, and also Ferber et al. state themselves that, in many domains, a fraction of the training data is sufficient. Thus we adapted their resource limits to our setting, as follows.

For each benchmark instance, we generate training data on a single core for 56 hours. We train 10 heuristic functions. Each heuristic function is trained on two cores for up to 2.8 hours. Supervised learning has to keep the training data in memory, thus we use Ferber et al.’s original memory limit of 12 GB. We use validation as described above, and evaluate the resulting heuristic function h^{IL} in our experiments.

B.2 Adaptations for h^{HGN}

STRIPS-HGN is designed for good performance with short training time, and in their original work Shen, Trevizan, and Thiébaux (2020) train the networks for only 10 minutes on small-sized problem instances. To provide a fair comparison, we adapted the training procedure of STRIPS-HGN to account for the extra training time and the source of training data used by the other learning approaches. Precisely, for each domain, we trained 10 different STRIPS-HGN networks simultaneously for up to 28 hours using 4 cores and 3.8 GB per core. We split the training time between data generation (10 hours) and network training (1.8 hour per network). Initially, we tried out different training parameters for STRIPS-HGN. We observed that, for Blocksworld, Scanalyzer and Transport, the original training time of 10 minutes

Hard Tasks with Validation

Domain	h^{Boot}	$h^{\text{Boot+}}$	h^{BExp}	$h^{\text{BExp+}}$	h^{AVI}	$h^{\text{AVI+}}$	h^{FF}	$h^{\text{FF+}}$
blocks	0	+0	0	+0	0	+0	62	+9
depots	8	+16	4	+13	13	+1	36	+31
grid	88	+11	95	+4	70	+10	53	24
npuzzle	0	+0	0	+0	0	+0	33	-2
pipes-nt	23	+6	19	+10	8	+3	27	+37
rovers	3	+34	1	+5	6	+0	14	+82
scanaly.	3	+5	0	+3	61	+6	98	+1
storage	27	+5	13	+6	16	+6	14	-5
transport	0	+0	0	+0	2	+30	0	+26
visitall	28	+4	0	+0	0	+0	74	+4

Table 2: The coverage (in %) for h^{Boot} , h^{BExp} , and h^{AVI} with and without using the preferred operators of h^{FF} on the hard tasks.

and a shorter data generation time of 2 hours leads to more robust performance. We hence used this setup for these three domains. In all cases, we use the validation states to select the best STRIPS-HGN network per domain.

We generate the training data for STRIPS-HGN as follows. We sample, with replacement, a moderate or hard instance, perform the same backward walk as h^{Boot} and h^{BExp} for n steps (see below), and solve the generated task using A^* instead of GBFS (as in the original STRIPS-HGN). We repeat this procedure until time is up. We discard every task solved within 5 minutes as they are too easy. We also discard tasks not solved after 30 minutes as it is unlikely that we will find a solution. From the solved tasks, we use the states along the (optimal) plans as training data.

The random walk length n here is uniformly chosen from $\{\underline{n} \leq n \leq \bar{n}\}$ where \underline{n} and \bar{n} are initially 50 and 500, respectively. Whenever our procedure generates an easy task, it updates the lower bound \underline{n} to $(\underline{n} + 3n)/4$; whenever our procedure generates a timed-out task, it updates the upper bound \bar{n} to $(\bar{n} + n)/2$. The number of state-value pairs obtained following this procedure ranges from 78 for Transport to 1563 for VisitAll. All the mentioned parameters were tuned so as to optimize h^{HGN} ’s performance in GBFS.

C Further Results

C.1 Preferred Operators

All our techniques can be enhanced by the preferred operators of h^{FF} . For this purpose, we execute a GBFS with two open lists. The first open list uses simply the predictions of the NN, the second open list uses the same predictions, but stores only states which are reached by a preferred operator of h^{FF} . Table 2 shows that the enhanced versions dominate the base versions. For every technique, using the preferred operators leads to significant improves in multiple domains.

C.2 Coverage Over Time

Given our comparatively large search time limit of 10 hours, Figure 1 shows coverage as a function of runtime. We show results for moderate tasks in four domains; the data is qualitatively similar in the other domains and tasks.

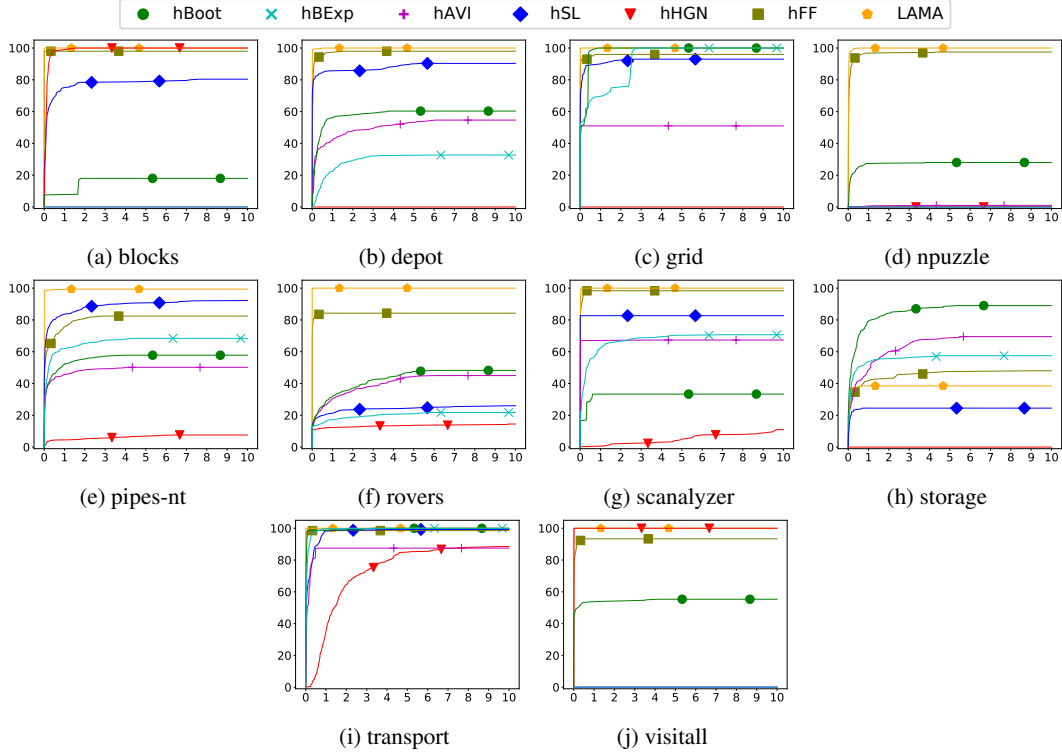


Figure 1: Coverage (%) as a function of the search time (in hours), on all moderate task for four domains.

Comparing different NN heuristics, the general finding is that coverage superiority persists over any time limit. With few exceptions, an approach that is better after 30 minutes is still better after 2 or more hours. The picture with respect to LAMA is different, as LAMA (like all state-of-the-art model-based heuristic search planners) tends to solve a task either quickly or not at all. With its comparatively fast heuristic functions, LAMA quickly runs up against the memory limit. The NN heuristic functions in contrast are very slow (run on a single core!), and thus require some time to “catch up” with LAMA. They still solve additional tasks even after very long run-times, and relative performance differences become more pronounced over time. In particular, the advantages over LAMA in Storage grow as a function of the run-time limit.

C.3 Informedness

We compare the informedness across the different approaches by comparing the number of expansions they require to solve a task. Figure 2 shows the distribution of expansions per domain, for commonly solved moderate tasks. In each domain we ignore algorithms that solve less than 10% of the tasks, as otherwise the set of commonly solved tasks would become too small.

Again, the primary conclusion from these results is that *the techniques are highly complementary* – at a glance, just consider how the different colors in Figure 2 move to and fro in the plots. Comparing neural network heuristic functions against each other, h^{HGN} is only well informed in the

3 domains in which it yields high coverage. The comparison between h^{IL} and our RL methods is similar as for coverage, exhibiting performance differences in the same domains (which is expected as the per-state runtime of these heuristic functions is very similar). Finally, the NN heuristic functions are quite competitive with h^{FF} and LAMA in terms of informedness. In Depots, Grid, and VisitAll the lowest number of expansions is achieved by a NN heuristic function (a different one in each case); and in Pipesworld-NoTankage, Rovers, and Scanalyzer, the best NN heuristic function is basically on par with h^{FF} .

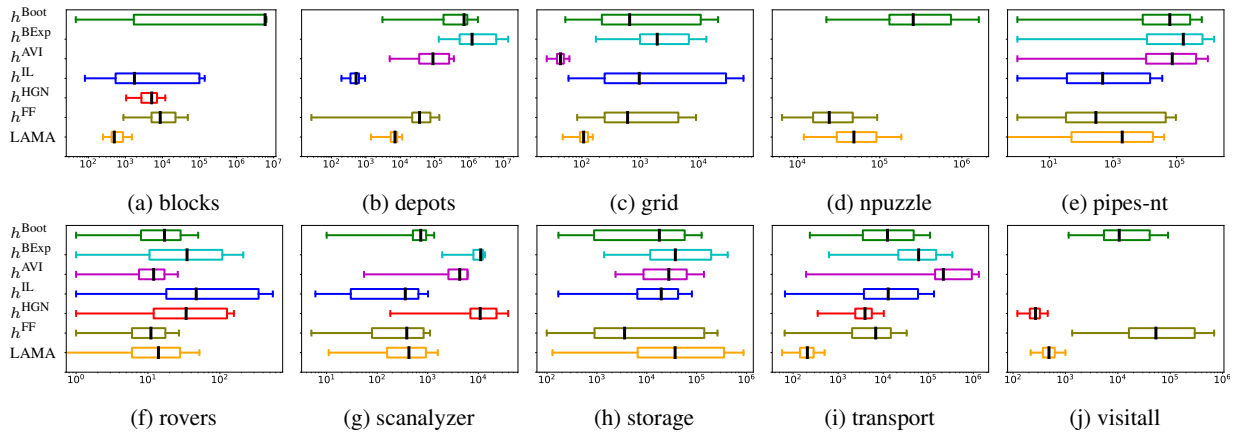


Figure 2: Expansions on commonly solved moderate tasks, removing algorithms with coverage $< 10\%$. In each plot, the line within the body indicates the median, the body of the box plot indicates the 25 and 75 percentile, and the whiskers show the 5 and 95 percentiles.