Supplementary Material: Utilising Uncertainty for Efficient Learning of Likely-Admissible Heuristics

Ofir Marom, Benjamin Rosman
University of the Witwatersrand, Johannesburg, South Africa

1 Practical Algorithms

In the main paper we introduce two conceptual algorithms, *GenerateTask* and *LearnHeuristic*. We later discuss considerations required for a practical implementation. In this section we introduce two new algorithms *GenerateTaskPrac* (Algorithm 3) and *LearnHeuristicPrac* (Algorithm 4) that implement these.

Algorithm 3: GenerateTaskPrac practical implementation of *GenerateTask*.

Input: { nn_{WUNN}, `// a weight uncertainty neural network`
ϵ, *MaxSteps*, K `// as described in the paper`
}
$s' = s_g$
$numSteps = 0$
$s'' = NULL$ `// used to store the previously observed state`
while *(numSteps < MaxSteps)* **do**
  $numSteps = numSteps + 1$
  initialise a dictionary $states\langle\mathcal{S}, \mathbb{R}^+\rangle$
  foreach $s \in \mathcal{E}^{rev}(s')$ **do**
    if $s'' \neq NULL$ *and* $s'' = s$ **then**
      continue loop `// don't include the state that takes you back to the previously observed state`
    end
    $x = F(s)$
    compute $\sigma_e^2(x)$ from nn_{WUNN} using K samples
    $states[s] = \sigma_e(x)$
  end
  sample from softmax distribution derived from *states.Values* to obtain some pair $(s, \sigma_e(x))$
  if $\sigma_e^2(x) \geq \epsilon$ **then**
    $\mathcal{T} = \langle\mathcal{S}, \mathcal{O}, \mathcal{E}, \mathcal{C}, s, s_g\rangle$
    return($\mathcal{T}$)
  end
  $s'' = s'$
  $s' = s$
end

2 Pattern Databases

In the main paper we discuss that for the 24-puzzle, 24-pancake and 15-blocksworld domains we use pattern databases (PDBs) as features to the network. In this section we detail the patterns used for each domain. The PDBs are described from the reference point of the goal state of each domain.

Algorithm 4: LearnHeuristicPrac practical implementation of *LearnHeuristic*.

Input : { *NumIter*, *NumTasksPerIter*, *NumTasksPerIterThresh*, α_0, Δ, ϵ, β_0, γ, κ, ϵ, *MaxSteps*, *MemoryBufferMaxRecords*, *TrainIter*, *MaxTrainIter*, *MiniBatchSize*, t_{max}, μ_0, σ_0^2, q, K `// as described in the paper`

}

initialise a WUNN nn_{WUNN} using priors μ_0 and σ_0^2 `// used to obtain` σ_e^2

initialise a FFNN nn_{FFNN} `// used to obtain` $\hat{y}$ `and` σ_a^2

initialise a list $memoryBuffer\langle(F(\mathcal{S}), \mathbb{R}^+)\rangle$

$y^q = -\infty$ `// stores quantile` q `of observed cost-to-goals`

$\alpha = \alpha_0$ `// set to the initial admissibility probability`

$\beta = \beta_0$ `// set to the initial prior strength factor`

$updateBeta = TRUE$ `// controls whether` β `is updated`

define a function $h(\alpha, \mu, \sigma)$ that that returns y^α where $P(y^\alpha \leq y \mid y \sim \mathcal{N}(\mu, \sigma^2)) = \alpha$ `//` h `is computed from the inverse CDF of a normal distribution`

for $n \in 1 : NumIter$ **do**

    $numSolved = 0$ `// counts number of solved tasks`

    for $i \in 1 : NumTasksPerIter$ **do**

        $\mathcal{T} = GenerateTask(nn_{WUNN}, \epsilon, MaxSteps, K)$

        try solve $\mathcal{T}$ within t_{max} seconds using IDA* with $\max(h, 0)$ as the heuristic to obtain a plan π. When planning with h for each state visited s, pass α, $\hat{y}(x)$ and $\sigma_t^2(x)$ as the parameters respectively where $\sigma_t^2(x) = \sigma_a^2(x)$ if $\hat{y}(x) < y^q$ else $\sigma_t^2(x) = \epsilon$ and where $x = F(s)$.

        if *plan* π *was found* **then**

            $numSolved = numSolved + 1$ `// count solved tasks`

            foreach $s_j \in \pi$ **do**

                if $(s_j \neq s_g)$ **then**

                    compute y_j, the cost-to-goal from s_j

                    $x_j = F(s_j)$

                    $memoryBuffer.Add((x_j, y_j))$

                end

            end

        end

    end

    trim *memoryBuffer* to keep the most recently added $MemoryBufferMaxRecords$ records

    if $numSolved < NumTasksPerIterThresh$ **then**

        $\alpha = \max(\alpha - \Delta, 0.5)$ `// we cannot solve enough tasks so reduce admissibility probability`

        $UpdateBeta = FALSE$ `// we update` α `so we don't update` β `because we want to keep the strength of the prior the same as before and try solve tasks with lower admissibility probability`

    else

        $UpdateBeta = TRUE$ `// update` β `because we are not updating` α

    end

    train nn_{FFNN} using entire *memoryBuffer* for *TrainIter* iterations

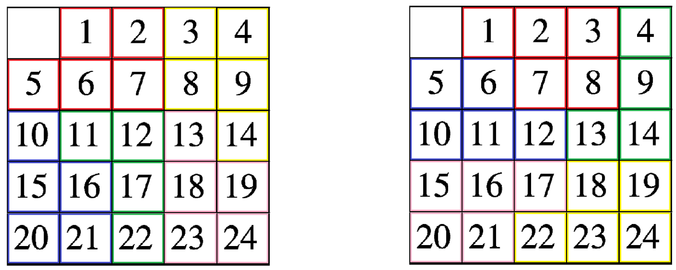
    train nn_{WUNN} from *memoryBuffer* for *MaxTrainIter* iterations using a minibatch size of *MiniBatchSize* per iteration. If after any iteration $\sigma_e^2(x_i) < \kappa\epsilon$ for all (x_i, y_i) in *MemoryBuffer* then stop early. Else complete *MaxTrainIter* iterations and if $UpdateBeta = TRUE$ then $\beta = \gamma\beta$ `// either reduce the epistemic uncertainty on all states in the memory buffer or reduce the importance of the prior`

    update y^q with quantile q of the cost-to-goal observations in *memoryBuffer*

end

2.1 24-puzzle

For the 24-puzzle domain we use two sets of disjoint 5-5-5-4 PDBs.



(a) First set of disjoint PDBs. (b) Second set of disjoint PDBs.

Figure 1: Disjoint PDBs for 24-puzzle.

For the first set:

- 1, 2, 5, 6, 7
- 3, 4, 8, 9, 14
- 10, 15, 16, 20, 21
- 11, 12, 17, 22
- 13, 18, 23 ,24

For the second set:

- 1, 2, 3, 7, 8
- 5, 6, 10, 11, 12
- 15, 16, 17, 20, 21
- 4, 9, 13, 14
- 18, 19, 22, 23, 24

2.2 24-pancake

For the 24-pancake domain we use two sets of location-based disjoint 5-5-5-4 PDBs.

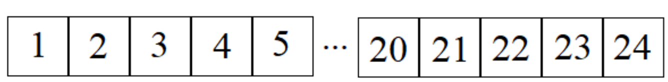


Figure 2: Goal state for 24-pancake.

For the first set:

- 1, 2, 3, 4, 5
- 6, 7, 8, 9, 10
- 11, 12, 13, 14, 15
- 16, 17, 18, 19, 20
- 21, 22, 23, 24

For the second set:

- 1, 2, 3, 19
- 4, 5, 6, 7, 8
- 9, 10, 11, 12, 13
- 14, 15, 16, 17, 18
- 20, 21, 22, 23, 24

2.3 15-blocksworld

For the 15-blocksworld domain we use 12 4-block PDBs.

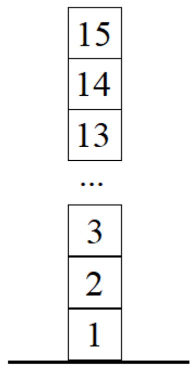


Figure 3: Goal state for 15-blocksworld.

The 12 4-block PDBs are:

- 1, 2, 3, 4
- 5, 6, 7, 8
- 9, 10, 11, 12
- 12, 13, 14, 15
- 3, 4, 5, 6
- 7, 8, 9, 10
- 11, 12, 13, 14
- 1, 2, 14, 15
- 2, 3, 4, 5
- 6, 7, 8, 9
- 10, 11, 12, 13
- 1, 13, 14, 15

3 Extensions

3.1 Multiple Goal States

In the main paper we make the assumption that all domains have a unique goal state. This assumption can be relaxed as follows:

- when running *GenerateTask* sample a goal state s_g from all possible goal states $\mathcal{S}_g$ and then proceed to generated a task starting from this goal state.
- We now need a heuristic that incorporates the goal state as well i.e. $h(s, s_g)$. This can be achieved by extending the feature representation to distinguish between the different goal states i.e. $x = F(s, s_g)$.

We note that as the size of $\mathcal{S}_g$ increases, more time will be required to learn a suitable heuristic because the model needs to learn from enough sampled goal states to generalise across the domain. Clearly, the way the goal states is encoded would be an important factor for how effectively the model can generalise across the domain.

3.2 Lifted Domains

The domains used in the paper differ only in their start states and in section 3.1 we discuss how to extend the framework to multiple goal states. However, we may wish to learn a heuristic that applies to a lifted domain - for example a heuristic that works well for an arbitrary n-puzzle, or a heuristic for the entire domain of a game like Sokoban where the size of the maze as well as the locations of walls, boxes, storage locations changes in each task.

Conceptually, this is straightforward to incorporate into our proposed framework as it requires only a judicious choice for the feature function F and model $\mathcal{M}$. In practice, learning general and transferable skills is an active area of research in the field of Artificial Intelligence. For example, previous work has shown that by using a particular choice of representation together with convolutional neural networks, a heuristic function can be learned that generalises across the Sokoban domain [1]. Then so long as the network architecture can be augmented with a mechanism to effectively model epistemic and aleatoric uncertainty we can incorporate it into our proposed framework.

4 Detailed Results

We include tables as in the paper but with the addition of the standard deviation, in brackets, for each statistic. We note that the standard deviations for the statistics of the suboptimality experiments in some domains is very high (in some cases, higher than the means) and that the empirical distribution of these statistics is highly skewed to the right. Finding techniques to reduce the run variance is an area of future research for learning likely-admissible heuristics under our framework.

Table 7: Detailed suboptimality results for 15-puzzle.

α	Time	Generated	Subopt	Optimal
0.95	74.6 (81.50)	78,787,262 (76,296,444)	2.21% (2.17%)	67.80% (15.08%)
0.9	26.72 (25.44)	29,342,747 (27,830,962)	2.46% (2.41%)	65.20% (16.95%)
0.75	8.71 (11.40)	9,357,055 (12,682,566)	2.97% (2.72%)	59.00% (18.35%)
0.5	5.06 (5.56)	5,284,645 (6,160,880)	3.35% (2.62%)	52.30% (15.96%)
0.25	4.85 (6.99)	5,107,840 (7,728,056)	4.45% (3.03%)	38.30% (16.96%)
0.1	3.89 (5.02)	4,285,483 (5,996,957)	5.32% (3.39%)	30.70% (16.11%)
0.05	3.80 (4.67)	4,189,753 (5,696,733)	5.63% (3.03%)	25.3% (11.27%)
N/A	2.25 (3.09)	3,071,956 (4,749,797)	10.75% (3.23%)	10.90% (5.54%)

5 Hardware

All experiments were run on an Intel i7-5500U 2.40Ghz CPU with 8GB RAM.

Table 8: Detailed efficiency results for 15-puzzle.

LengthInc	Solved Train		Solved Test	
1	100%	(0.00%)	38.59%	(3.56%)
2	95.10%	(2.27%)	48.20%	(3.32%)
4	61.80%	(10.57%)	51.40%	(16.21%)
6	36.20%	(6.01%)	39.61%	(3.97%)
8	19.20%	(3.49%)	35.16%	(3.56%)
10	11.80%	(6.78%)	31.83%	(12.04%)
GTP	93.30%	(2.65%)	60.59%	(12.32%)

Table 9: Detailed suboptimality results for 24-puzzle.

α	Time		Generated		Subopt		Optimal	
0.95	2,664.53	(3,062.59)	1,233,965,823	(1,322,894,069)	2.15%	(0.65%)	28.80%	(11.36%)
0.9	1,371.29	(1,292.45)	628,101,474	(539,128,139)	2.64%	(0.66%)	20.00%	(9.55%)
0.75	549.22	(549.44)	274,003,465	(259,744,778)	3.67%	(0.68%)	5.20%	(4.66%)
0.5	189.37	(131.53)	99,244,234	(70,575,145)	4.64%	(0.54%)	1.20%	(1.60%)
0.25	121.18	(66.23)	66,147,586	(35,047,947)	5.18%	(0.58%)	0.00%	(0.00%)
0.1	86.62	(50.78)	46,988,530	(28,554,357)	5.81%	(0.51%)	0.00%	(0.00%)
0.05	83.56	(37.19)	40,046,361	(18,032,980)	6.26%	(0.55%)	0.00%	(0.00%)
N/A	25.39	(20.83)	11,719,659	(9,552,893)	11.34%	(0.85%)	0.00%	(0.00%)

Table 10: Detailed suboptimality results for 24-pancake.

α	Time		Generated		Subopt		Optimal	
0.95	364.58	(85.68)	104,132,601	(27,914,343)	1.09%	(0.09%)	76.00%	(1.26%)
0.9	198.56	(37.55)	54,089,822	(10,731,888)	1.27%	(0.07%)	72.40%	(1.96%)
0.75	54.24	(12.35)	13,001,211	(1,713,639)	1.85%	(0.13%)	59.20%	(2.99%)
0.5	20.42	(3.88)	4,530,281	(820,070)	2.17%	(0.11%)	53.20%	(3.25%)
0.25	11.66	(2.03)	2,511,066	(396,223)	3.53%	(0.29%)	37.20%	(6.52%)
0.1	8.30	(3.75)	1,621,775	(843,061)	3.83%	(0.72%)	30.80%	(7.33%)
0.05	4.96	(2.65)	871,908	(441,617)	4.03%	(0.79%)	30.80%	(7.65%)
N/A	0.85	(1.30)	210,622	(338,283)	10.58%	(4.73%)	8.40%	(12.86%)

Table 11: Detailed suboptimality results for 15-blocksworld.

α	Time		Generated		Subopt		Optimal	
0.95	55.51	(11.49)	115,691,631	(7,070,181)	0.02%	(0.03%)	99.60%	(0.80%)
0.9	53.79	(11.78)	112,390,208	(9,764,255)	0.07%	(0.06%)	98.40%	(1.50%)
0.75	50.52	(11.89)	101,109,757	(15,266,842)	0.23%	(0.20%)	95.60%	(3.67%)
0.5	38.01	(15.44)	69,663,441	(19,064,929)	0.98%	(0.49%)	84.80%	(7.22%)
0.25	43.97	(34.56)	63,963,572	(44,432,088)	4.28%	(2.01%)	50.80%	(13.00%)
0.1	35.83	(21.90)	50,951,658	(25,855,679)	9.70%	(5.87%)	34.40%	(11.20%)
0.05	28.50	(19.06)	42,499,655	(28,081,066)	13.36%	(9.05%)	24.00%	(12.46%)
N/A	20.88	(22.20)	31,178,090	(32,600,259)	7.07%	(3.50%)	38.40%	(13.71%)

6 Code

A C# implementation for all the domains described in the paper can be found here:
https://github.com/OfirMarom/LearnHeuristicWithUncertaintly

Table 12: Detailed training runtime in hours.

Domain	plan with $\hat{y}$		plan with y^{α}	
15-puzzle	1.32	(0.16)	2.67	(0.17)
24-puzzle	6.03	(0.36)	20.52	(1.32)
24-pancake	2.34	(0.19)	15.85	(1.19)
15-blocksworld	4.38	(0.47)	6.54	(0.27)

References

[1] E. Groshev, A. Tamar, S. Srivastava, and P. Abbeel. Learning generalized reactive policies using deep neural networks. *arXiv:1708.07280*, 2017.