Learning to Correct Mistakes: Backjumping in Long-Horizon Task and Motion Planning: Supplementary Document

Yoonchang Sung[1*], **Zizhao Wang**[1*], **Peter Stone**[1,2]
[1]The University of Texas at Austin [2]Sony AI

1 Introduction

In this appendix, we provide additional materials and evaluations in support of the main paper. We firstly present the architecture of our models and training details in Section 2. We secondly report a set of additional evaluation results in Section 3, such as the backjumping prediction accuracy (Section 3.1), varying training data size (Section 3.2), batch sampling results (Section 3.3), empircal evaluation with additional baselines (Section 3.4), effect of the sample size (Section 3.5), effect of the sampling budget (Section 3.6), computation time (Section 3.7), and analysis on the overhead of the model inference (Section 3.8). We lastly show the pseudo-code of the proposed algorithms in Section 4.

Unless stated otherwise, all evaluations are conducted in the following settings. The planning code is executed on one core of Intel(R) Xeon(R) Gold 6342 CPU. Numbers represent the mean $\pm 95\%$ confidence interval computed by solving 100 problems.

Table 1: Architectures for imitation learning (**IL**) and plan feasibility (**PF**) used for all tasks.

Modules	Name	Parameter
GNN	node feature size,	128
	edge feature size,	128
	global feature size,	128
	node network,	[128, 128]
	edge network,	[128, 128]
	global network,	[128, 128]
IL	# of attention blocks	3 for packing, 2 for NAMO
	attention size	256
	# of attention heads	8
	attention residual network	[256]
	RNN hidden size	256
	# of recurrent layers	3 for packing, 2 for NAMO
	object feature size	256
	MLP1	[128]
	MLP2	[128, 128]
PF	# of attention blocks	3 for packing, 2 for NAMO
	attention size	256
	# of attention heads	8
	attention residual network	[256]
	RNN hidden size	256
	# of recurrent layers	3 for packing, 2 for NAMO
	object feature size	256
	MLP1	[128]
	MLP2	[128, 128]

[*]Equal contribution.

2 Architecture and Training Details

We list the architectures of our imitation learning (**IL**) and plan feasibility (**PF**) models in Table 1 and training hyperparameters in Table 2. Note that both **IL** and **PF** models use the same GNN architecture (Table 1). For each MLP network, we represent its architecture, *e.g.*, [128, 128] is an MLP of 2 hidden layers with 128 neurons in each layer. For all activation functions, ReLU is used.

Table 2: Hyperparameters shared across all methods and tasks.

Name	**Task**	
	Packing	NAMO
number of tasks for training	500	250
number of tasks for testing	100	50
optimizer	Adam	
learning rate	1e-4	
batch size	32	

3 Additional Results

3.1 Backjumping prediction accuracy

In Table 3, we show how closely our backjumping prediction $\hat{k}^\star$ finds the ground-truth $k^\star$ in the packing task, where each method is trained with data collected from 500 problems. We consider three cases: (1) $\hat{k}^\star = k^\star$ (*i.e.*, correct prediction) where the backjump correctly guides the search to directly modify the culprit, (2) $\hat{k}^\star < k^\star$ (we call LT) where the backjump still has a chance to reach the culprit without backjumping but overshoots (*i.e.*, the culprit exists in the descendants of $\hat{k}^\star$), thus having lower planning efficiency compared with $\hat{k}^\star = k^\star$, and (3) $\hat{k}^\star > k^\star$ (we call GT) where the backjump undershoots, and thus, the search cannot reach the culprit without backjumping. For example, that $\hat{k}^\star \leq k^\star$ at around 70% of the time implies that the chance of finding the culprit without backjumping occurs more frequently than missing the culprit, leading to improved planning efficiency.

Notice that our method involves sampling that approximates a continuous domain in a discrete way. Because of this, both the following situations are possible in the LT case: (1) The search may have a new opportunity to find a feasible solution by sampling a different value for the variable that used to be the culprit before backjumping but is no longer a culprit. (2) The search may make a new culprit variable by sampling a wrong value, and thus, further backtracking or backjumping is required to find a feasible solution.

Table 3: The accuracy of $\hat{k}^\star$ prediction in the packing task. Numbers represent the mean $\pm$ 95% confidence interval computed by solving 100 problems. The number with $*$ represents the best-performing method within the same data size. The bolded numbers are those whose performance is not statistically significantly different from the one with $*$.

Metric	**IL RNN**	**IL Attn**	**PF RNN**	**PF Attn**
Correct prediction percentage (%)	**39.3 $\pm$ 1.0**	**39.2 $\pm$ 1.5***	44.2 $\pm$ 0.5	43.0 $\pm$ 0.4
LT percentage (%)	**33.0 $\pm$ 2.2**	**30.0 $\pm$ 2.0**	**28.6 $\pm$ 2.2***	**30.3 $\pm$ 2.3**
GT percentage (%)	**27.6 $\pm$ 1.2**	**30.1 $\pm$ 2.1**	**27.2 $\pm$ 1.8**	**26.6 $\pm$ 1.9***
LT distance	**1.76 $\pm$ 1.10**	**1.76 $\pm$ 1.06**	**0.71 $\pm$ 1.12***	**0.80 $\pm$ 1.20**
GT distance	**1.58 $\pm$ 0.91**	**1.59 $\pm$ 0.93**	**1.37 $\pm$ 0.69**	**1.34 $\pm$ 0.65***
Prediction backjumping distance	**2.58 $\pm$ 1.66**	**2.48 $\pm$ 1.62***	**2.56 $\pm$ 1.54**	**2.67 $\pm$ 1.62**
Ground-truth backjumping distance	2.44 $\pm$ 1.56			

The mean and standard deviation of the LT distance and GT distance (*i.e.*, $|\hat{k}^\star - k^\star|$ for both distances) are shown in the fourth and fifth rows, and for each method, it can be seen that the prediction $\hat{k}^\star$ has a relatively small deviation from the $k^\star$ labels. Finally, as shown in the last two rows, compared to backtracking which always reevaluates the variable at one level higher, our methods backjump to higher levels (*i.e.*, $k^d - \hat{k}^\star$) and avoid evaluating irrelevant variables. For reference, we show the ground-truth backjumping distance (*i.e.*, $k^d - k^\star$); one can see that the prediction distances of all methods are close to this ground-truth distance.

3.2 Varying training data size

In Table 4, we show the efficiency of our methods when varying the training data size (*i.e.*, data collected by varying the number of problems). All methods trained with the reported data sizes outperform backtracking significantly. With more data, the performance of each method improves, except for **PF** trained with more than 2000 problems. **PF** collects sufficient data even in a small data regime as it gathers one feasibility likelihood label for each node visited in the search tree. Thus, further increasing data size does not significantly affect the **PF** performance.

Table 4: The number of nodes visited in the search tree in the packing task when varying the data size (measured by the number of problems).

Data size	Backtracking	**IL RNN**	**IL Attn**	**PF RNN**	**PF Attn**
100	4414 ± 879	$\mathbf{3490 \pm 698}$	$\mathbf{3615 \pm 831}$	$\mathbf{3487 \pm 704}^*$	$\mathbf{3592 \pm 788}$
500		$\mathbf{2464 \pm 464}$	$\mathbf{2638 \pm 602}$	$\mathbf{2205 \pm 313}$	$\mathbf{2062 \pm 297}^*$
2000		$\mathbf{2439 \pm 529}$	$\mathbf{2404 \pm 464}^*$	$\mathbf{3084 \pm 672}$	$\mathbf{3468 \pm 729}$
4500		$\mathbf{1769 \pm 372}^*$	$\mathbf{2151 \pm 507}$	$\mathbf{2243 \pm 498}$	$\mathbf{2350 \pm 484}$

3.3 Batch sampling results

We conduct additional evaluation of batch sampling on the NAMO task (in Table 5). The results match with our results on the packing task, *i.e.*, although batch sampling still outperforms backtracking by far, forgetting empirically exceeds batch sampling in both tasks.

Table 5: The comparison between the forgetting method (**F**) and the batch sampling (**BS**) method, measured by the number of nodes visited in the search tree.

Task	Backtracking	**IL RNN**	**IL Attn**	**PF RNN**	**PF Attn**
Packing (**F**)	4414 ± 879	$\mathbf{2464 \pm 464}$	$\mathbf{2638 \pm 602}$	$\mathbf{2205 \pm 313}$	$\mathbf{2062 \pm 297}^*$
Packing (**BS**)	13541 ± 4205	$\mathbf{4464 \pm 1160}$	7073 ± 2040	$\mathbf{4556 \pm 749}$	$\mathbf{4311 \pm 690}^*$
NAMO (**F**)	$(21 \pm 10) \times 10^4$	$\mathbf{543 \pm 187}$	$\mathbf{425 \pm 153}^*$	$\mathbf{529 \pm 188}$	2615 ± 710
NAMO (**BS**)	$(44 \pm 11) \times 10^4$	16019 ± 5384	9558 ± 3411	$\mathbf{4327 \pm 1559}^*$	84286 ± 55595

3.4 Empirical evaluation with additional baselines

We implement additional baselines, which backjump predetermined steps at dead-ends, to compare with our methods. We report the number of nodes visited in the search tree and the wall clock time, measured in seconds, obtained by the forgetting algorithm for solving each packing problem with 10 objects. We choose the packing task because the performance gap between backtracking and our method in packing is much smaller than that in NAMO, and thus the packing task yields a more contrasting comparison.

Table 6: The number of nodes visited in the search tree and the wall clock time, measured in seconds, obtained by the forgetting algorithm for solving each packing problem with 10 objects.

# fixed backjumping steps	# visited nodes	Wall clock time
1 (*i.e.*, backtracking)	4414 ± 879	260.6 ± 58.1
2	3093 ± 509	120.0 ± 21.1
3	3323 ± 637	140.0 ± 26.9
4	2892 ± 557	154.0 ± 26.6
5	5408 ± 1106	391.6 ± 76.1
6	8360 ± 1570	577.9 ± 109.6
Always backjumping to root	11212 ± 2002	685 ± 120

The results show that when the fixed step is set to 4, the baseline algorithm performs the best among all baselines. Its confidence interval even overlaps with some of our methods, but we observe that our best results (*i.e.*, 1889 ± 328 of **IL RNN** from Table 8 and 2062 ± 297 of **PF Attn** from Table 4) still show the performance statistically significantly better than all baselines. Nonetheless, the rest of the results in the appendix support our claim that the performance of our methods can further be improved by varying training data size and/or sampling budget in training.

3.5 Effect of the sample size

We analyze the effect of the sample size (denoted by N in the paper) on the noisy data and the trained model performance so that users can determine appropriate N for their applications.

Since our method uses sampling, the planner may mistakenly think it hits a dead end even if a feasible action (*e.g.*, placement) is available. This corresponds to a false negative, which contributes to noisy labels and informs a sufficient number of samples not to miss a feasible action. The ideal sample size varies depending on a problem domain. Thus, as an example, we show false-negative ratios in the packing task (Table 7) where the goal is to find a placement for the 10-th object when there are 9 objects in the cabinet.

Table 7: False-negative ratios resulted when using different sample sizes.

Sample size	10	30	50	70	90
False-negative ratio	0.72	0.50	0.34	0.21	0.10

3.6 Effect of the sampling budget

We conduct additional evaluations on the performance of learning models in the packing task when varying the number of samples (*i.e.*, N in the paper) used in training. Specifically, we set the number of samples to be 10, 30, and 50 in training while that to be 30 in testing.

Table 8: The ratio of model inference time over the total wall clock time (in %) for solving a single problem.

# samples in training / in testing	**IL RNN**	**IL Attn**	**PF RNN**	**PF Attn**
10 / 30	$\mathbf{1889 \pm 328}^*$	$\mathbf{2153 \pm 369}$	3276 ± 597	2932 ± 450
30 / 30	$\mathbf{2464 \pm 464}$	$\mathbf{2638 \pm 602}$	$\mathbf{2205 \pm 313}$	$\mathbf{2062 \pm 297}^*$
50 / 30	$\mathbf{5752 \pm 1446}$	$\mathbf{6672 \pm 1819}$	$\mathbf{4859 \pm 1100}$	$\mathbf{4267 \pm 1035}^*$

The results show that **IL** performs the best for $10/30$, **PF** performs the best for $30/30$, and both perform the worst for $50/30$. It is expected that at the extreme (*i.e.*, $N = 1$) in training, **IL** would predict to backjump to near the root as, most of the time, a feasible placement is not found with $N = 1$. This would behave similarly to a baseline, always backjumping to a root; thus, its performance is expected to be poor. For a given number of samples in testing, we can treat the sample size in training as a hyperparameter, and one can tune it for their applications for better performance.

3.7 Computation time

We report the wall clock time for solving each problem, measured in seconds, in Table 9. The wall clock time is roughly proportional to the number of nodes visited in the search tree.

Table 9: The wall clock time for solving each problem, measured in seconds.

Task	Backtracking	**IL RNN**	**IL Attn**	**PF RNN**	**PF Attn**
Packing	202.0 ± 45.4	$\mathbf{97.4 \pm 19.3}$	162.1 ± 32.7	$\mathbf{94.7 \pm 22.0}^*$	$\mathbf{122.0 \pm 23.8}$
NAMO	61754.3 ± 24608.4	$\mathbf{66.0 \pm 19.9}$	$\mathbf{67.8 \pm 19.9}$	$\mathbf{58.7 \pm 18.3}^*$	11648.3 ± 12342.7

3.8 Analysis on the overhead of the model inference

We report the ratio of model inference time over the total wall clock time (in %) for solving a single problem.

Table 10: The ratio of model inference time over the total wall clock time (in %) for solving a single problem.

Task	**IL RNN**	**IL Attn**	**PF RNN**	**PF Attn**
Packing	$\mathbf{0.3 \pm 0.2}^*$	2.1 ± 0.6	7.7 ± 2.7	1.2 ± 0.3
NAMO	12.5 ± 4.1	$\mathbf{3.9 \pm 1.4}$	12.4 ± 3.8	$\mathbf{2.1 \pm 0.6}^*$

The results show the overhead of the querying backjumping model is relatively cheap, consisting of less than 8% for the packing task. Even though querying takes 12.5% of the wall clock time for NAMO tasks, it is worthwhile considering that using backjumping reduces the total wall clock time from 6×10^4s to around 60s. The results also show that the difference between the overhead of **IL** and that of **PF** is marginal.

We also report the average time to determine the dead end when querying the model, measured in milliseconds.

Table 11: The average time to determine the dead end when querying the model, measured in milliseconds.

Task	**IL RNN**	**IL Attn**	**PF RNN**	**PF Attn**
Packing	267.9 ± 217.5	$\mathbf{56.5 \pm 40.2}$	163.1 ± 112.4	$\mathbf{16.9 \pm 6.5}^*$
NAMO	823.4 ± 462.2	$\mathbf{292.7 \pm 176.3}$	800.8 ± 433.9	$\mathbf{138.1 \pm 68.3}^*$

Even though the attention method is queried for every previous step while the RNN is only queried once, it is still much faster than RNN because its time-series computation can be parallelized on GPUs, while the RNN needs to finish the time-series computation sequentially.

4 Pseudo-code

We present the pseudo-codes for both batch sampling and forgetting algorithms.

Algorithm 1: Batch sampling algorithm

```
Input : Time limit ($T$), number of samples per level ($N$)
$\mathcal{P} \leftarrow \emptyset$ // Initialize the empty plan.
$\mathcal{V}_k \leftarrow$ sampleValues $(c_k, N)$ $\forall k \in \{0, ..., K-1\}$ // Sample $N$ values per level $k$.
$k \leftarrow 0$
$\mathcal{V}'_k \leftarrow \mathcal{V}_k$
while not isTimeLimitExceeded $(T)$ do
    while $0 \leq k \leq K-1$ do
        while not isEmpty $(\mathcal{V}'_k)$ do
            $v \leftarrow$ selectValue $(\mathcal{V}'_k)$
            if isConsistent $(v, \mathcal{P})$ then // Check if $v$ is consistent with $\mathcal{P}$.
                $\mathcal{P}$.append $(v)$
                $k \leftarrow k+1$ // Move to the next level.
                $\mathcal{V}'_k \leftarrow \mathcal{V}_k$
                break
            else
                $\mathcal{V}'_k \leftarrow$ removeElements $([v], \mathcal{V}'_k)$ // Remove $v$ from $\mathcal{V}'_k$.
                if isEmpty $(\mathcal{V}'_k)$ then // The dead-end is met $(k = k^d)$.
                    $\hat{k}^\star \leftarrow$ predictBackjump (model inputs) // Model inputs are
                      specified in Section 4.
                    $\mathcal{P} \leftarrow$ removeElements $([\hat{k}^\star, ..., k-1], \mathcal{P})$ // Remove the last $k - \hat{k}^\star$
                      elements from $\mathcal{P}$.
                    $k \leftarrow \hat{k}^\star$ // Backjump to level $\hat{k}^\star$.
                    break
                end
            end
        end
    end
    if $k = 0$ then
        $\mathcal{V}_k \leftarrow$ sampleValues $(c_k, N)$ $\forall k \in \{0, ..., K-1\}$ // Sample a new batch of $N$
          values.
        $\mathcal{V}'_k \leftarrow \mathcal{V}_k$
    else
        return $\mathcal{P}$ // Plan is found.
    end
end
return no solution
```

Algorithm 2: Forgetting algorithm

```
Input : Time limit ($T$), number of samples per level ($N$)
$\mathcal{P} \leftarrow \emptyset$ // Initialize the empty plan.
$k \leftarrow 0$
while not isTimeLimitExceeded ($T$) do
    while $0 \leq k \leq K-1$ do
        $\mathcal{V}_k \leftarrow$sampleValues ($c_k, N$) // Sample $N$ values.
        while not isEmpty ($\mathcal{V}_k$) do
            $v \leftarrow$selectValue ($\mathcal{V}_k$)
            if isConsistent ($v, \mathcal{P}$) then // Check if $v$ is consistent with $\mathcal{P}$.
                $\mathcal{P}$.append ($v$)
                $k \leftarrow k+1$ // Move to the next level.
                break
            else
                $\mathcal{V}_k \leftarrow$removeElements ($[v], \mathcal{V}_k$) // Remove $v$ from $\mathcal{V}_k$.
                if isEmpty ($\mathcal{V}_k$) then // The dead-end is met ($k = k^d$).
                    $\hat{k}^\star \leftarrow$predictBackjump (model inputs) // Model inputs are
                     specified in Section 4.
                    $\mathcal{P} \leftarrow$removeElements ($[\hat{k}^\star, ..., k-1], \mathcal{P}$) // Remove the last $k - \hat{k}^\star$
                     elements from $\mathcal{P}$.
                    $k \leftarrow \hat{k}^\star$ // Backjump to level $\hat{k}^\star$.
                    break
                end
            end
        end
    end
    if $k \neq 0$ then
        return $\mathcal{P}$ // Plan is found.
    end
end
return no solution
```