Towards efficient and scalable planning: Learning search heuristics for multi-agent planning frameworks

Ashwin Misra
Mujin Corp
United States
ashwin.misra@mujin-corp.com

Viraj Parimi
Massachusetts Institute of Technology
United States
vparimi@mit.edu

Mansi Agarwal
Carnegie Mellon University
United States
magarwa2@andrew.cmu.edu

Zachary B. Rubinstein
Carnegie Mellon University
United States
zbr@cs.cmu.edu

Stephen F. Smith
Carnegie Mellon University
United States
sfs@cs.cmu.edu

Abstract: The combinatorics of various search-based approaches to planning pose a solid barrier to scalable performance. Such approaches become increasingly complex and complicated, even in single-agent planning contexts, as the number of goals to be achieved increases. We propose that such complex combinatorics can be overcome by learning an abstract model of the planner's search that utilizes various state characteristics to learn the relative quality of various search decisions over time. An efficient machine learning framework consisting of a Long Short Term Memory Network is developed to accelerate the time-consuming search process and achieve a substantial computational speedup by learning the planner's reasoning from spatial and temporal global states and constraints. It generalizes well and enables efficient usage of multiple agents across multiple tasks.

Keywords: Multi-Agent Planning, Machine Learning, Long Short Term Memory Networks, Hierarchical Task Networks

1 Introduction

Future operations in remote and inaccessible areas (e.g., subterranean exploration, disaster response) will require teams of robots to plan and execute autonomously and collaboratively under challenging temporal and spatial constraints, which in turn presents challenging multi-agent planning problems. The motivating domain of interest in our work, that of maintaining a deep space habitat during those periods when humans do not occupy the habitat, is a particular case in point. In contrast to current near-earth operations (e.g., the International Space Station) that can be manually controlled in real-time from the ground, delays in the range of 20 minutes are observed when communicating from Earth to a spacecraft in the vicinity of Mars [1]. These delays virtually prohibit the feasibility of real-time control and necessitate much higher levels of autonomy in multi-robot planning and execution. Equipped with a set of onboard robotic and autonomous subsystems capable of sensing relevant conditions (e.g., pressure drops, mechanical failures) and carrying out necessary maintenance tasks (e.g., changing filters, replacing a power module, manufacturing a damaged part, tending to science experiments), the spacecraft must continually generate and maintain multi-robot plans that assign pending tasks to individual robots (or to robot sub-teams when joint actions are required), ensure collision-free movement, monitor their execution and re-plan when necessary. As is typical in task planning, we assume that the planner is provided with descriptions of the types of tasks that can be performed (in our case hierarchical descriptions) and that the multi-agent plan is produced by a

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.

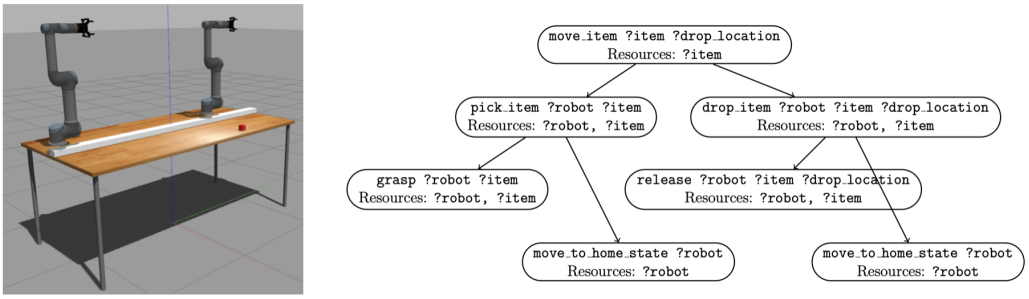

Figure 1: Domain setup with two robots on a discrete five-block rail, and an example decomposition tree

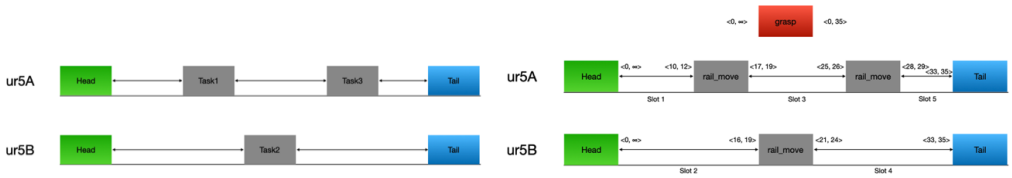

Figure 2: Pre-populated timelines and generated slots triggered by an incoming item

search process that generates and evaluates alternative actions that can be taken at each search state until a complete final plan is reached. To overcome the computational cost of this search process in complex multi-agent domains such as deep space habitat maintenance, we attempt to learn a model for a given domain that can be used to generate new multi-agent plans much more efficiently.

In this paper, we focus specifically on a recent Hierarchical Task Network planning framework T-HTN [2] for multi-agent planning and scheduling that was developed to support the continual management of a team of agents. This framework combines the temporal flexibility of timeline-based planners and the structure of a hierarchical task network. This leads to a more extensive search space as states across the planning horizon need to be considered. However, T-HTN takes advantage of problem structure by designating particular objects as resources and assuming that all actions will be allocated and scheduled on specific resources. As it is a constraint-based search mechanism with spatio-temporal constraints, a global schedule of activities is maintained over time. During the search process, multiple feasible options compliant with these constraints are generated and evaluated based on an objective function (like minimizing makespan), and the best one is selected. Let us understand this by an example, let us take an example of moving a red cube from its given location to an edge in our example/evaluation domain 1, which would be referenced as the base of our experiments in this article.

Considering this incoming high-level request, a path decomposition tree is created for all potential possible alternatives for low-level tasks which is described in example 1. All the alternative decompositions are collected through recursively expanding OR nodes and depth-first search. This high-level task has corresponding tokens that form a timeline and have empty 'slots' consisting of where each token can be placed. Let us say the first sub-task is a grasp and there are pre-existing rail move actions on the timeline. Moving on to the selection process, the planner checks all the feasible slots 2 (slots which satisfy all temporal and spatial constraints), for an objective function, which in this case can be to reduce makespan. It will do so by scheduling the task on each of these slots and keeping track of the makespan of each schedule, this is called a backtracking search.

This search process becomes increasingly complex with the number of pre-existing tasks (more rail moves), decomposed sub-tasks (grasp, rail move, ungrasp, move to home), and agents (more robots). However, we believe this complexity can be mitigated by learning search heuristics as a

Item	Spatial Features	Temporal Features
Slot	Start location End location	Early Start Time, Late Start Time Early Finish Time, Late Finish Time
Incoming Task	Pick-up location Drop-off location	Duration

Table 1: Dataset: Slot and Task Description

variable ordering problem. We hypothesize that an abstract model of the planner's search can be learned that utilizes state descriptions of the planning state to recognize the relative importance of various search decisions for iterative planning. This model can aid the planner as a surrogate for the explicit combinatorial search and substantially speed up the computation time required to generate a partial solution. In our example, the spatial constraints can be dependent on the pick and place locations, the robot locations, etc. and the temporal constraints can be like grasping before ungrasping etc. Such constraints are always available in such planning problems and we hypothesize that these constraints hold the essence of the reasoning behind the planner's slot-searching process.

Recent and past research have combined Reinforcement Learning, Deep Learning, and Meta-Learning have combined learning in different aspects of planning to improve backtracking efficiency. Reinforcement Learning-based scheduling approaches [3, 4] require a very large amount of data to train the agent and can be computationally expensive. Other planning and learning algorithms [5] combine tree search with online learning to guide the exploration-exploitation trade-off during the search process, which is very sensitive to the choice of parameters, such as the depth of the decision tree and the learning rate for the entropy estimates. Poor choices of these parameters can result in sub-optimal or even inefficient search behavior. Secondly, using online learning to update the entropy estimates can be computationally expensive, particularly in large search spaces. Most of the research in this area has been focused on substituting the planner with the learner, or computationally expensive guiding mechanisms which are not very practical. We argue that a computationally inexpensive learning model is needed that does not substitute the deterministic nature of planning with a probabilistic nature, but instead aids the planner in making quicker decisions on past performance through global awareness.

To summarize, We propose Learn2Plan [6], a novel memory-driven learning framework that supports hierarchical planners to enable multi-agent coordination efficiently. We designed Learn2Plan to learn constraints from spatiotemporal descriptions of the world state and understand the characteristic behavior of hierarchical planners.

2 Methodology

Our broad methodology as described in algorithm 1, learns the slot order heuristic of a planner through the constraints and ordering for different incoming tasks relative to the already populated tasks on the timeline. The LSTM structure is important to understand that there are inherent differences for a sub-task that starts at a different location and comes around the beginning of a high-level task as compared to one that is at the end of a high-level task. Once for each request(sub-task), a slot is chosen, this slot is sent to the planner to be validated and then added to a task network. This ensures that even if the predictions are wrong in some cases, the planner will make sure to check that particular slot for feasibility.

We consider pick-and-place operations as tasks for our given domain 1. This is because these operations encompass a wide range of activities, including cargo stowage, object retrieval, and manipulation of parts on board. Each request corresponds to picking a box up and keeping it in the desired position. We use the domain as described above, and we limit to problems with five incoming requests and five discrete rail block divisions, with only two slots per request investigated.

Our sampled data from the planner outputs the state descriptions and ground truths. We have focused on high variance in our data set to capture the different nature of tasks with varied constraints.

Algorithm 1 Learn2Plan

procedure SLOTORDERHEURISTIC(T, S)
  $S \leftarrow$ Initialize an empty slot list
  for all $t \in T$ **do**
    $D_a \leftarrow S_a, t$
    S_{order} = LSTM(D_a)
    if $S_{order} \neq \varnothing$ **then**
      for all $s \in S_{order}$ **do**
        if s is validated by the planner and consistent with constraints **then**
          $s \leftarrow$ task t
          Add s to $TaskNetwork$
        end if
      end for
    end if
  end for
  return S
end procedure

Dataset $S = \{\boldsymbol{z}_i\}_{i=1}^n = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$ is sampled from a distribution $\mathcal{D}$ over a domain $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. This distribution consists of 100,000 tasks with descriptions defined in Table 1, with all slots per request and corresponding makespan. The target variable being $\boldsymbol{y}_i = \{SelectedSlot, makespan\}$. For an incoming task T, it is divided into sub-tasks t.

$\mathcal{X}$ is the instance domain (a set), $\mathcal{Y}$ is the label domain (a set), and $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ is the example domain (a set).

$$\begin{aligned} \boldsymbol{x}_i &= \{RequestID, description_{state}, description_{task}, slots, makespan\} \\ RequestID &\in requestA, requestB, requestC, requestD, requestE \\ slots &= \text{All feasible slots per RequestID} \end{aligned}$$

Usually, $\mathcal{X}$ is a subset of $\mathbb{R}^d$ and $\mathcal{Y}$ is a subset of $\mathbb{R}^{d_o}$, where d is the input dimension (10), d_o is the output dimension (1). The label for our data is the selected slot and the corresponding makespan.

$$\boldsymbol{y}_i = \{SelectedSlot, makespan\}$$

$n = 100,000$ is the number of samples. Without specification, S and n are for the training set, where each sample contains the sequence of all requests required to complete a task. We use a 70/30 split between training and testing. We do not keep a cross-validation set because the data is well-curated and structured by the planner.

In our case, we take one primary loss and an auxiliary loss. The primary loss is a standard classification loss, a stable request-wise Binary Cross Entropy with Logits Loss. The auxiliary loss is a Root-Mean Squared Error for our linear variable of makespan.

$$\begin{aligned} \ell(f_{\boldsymbol{\theta}}, \boldsymbol{z}) &= \ell_{slot} + \ell_{makespan} \\ \ell_{slot} = L_{BCE} &= -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \\ \ell_{makespan} = RMSE &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \end{aligned}$$

where ℓ_{slot} is the classification loss for the predicted slot, and $\ell_{makespan}$ is a regression loss for the predicted makespan. $\hat{y}_i$ is the ground truth, which is the slot chosen by the planner and the makespan

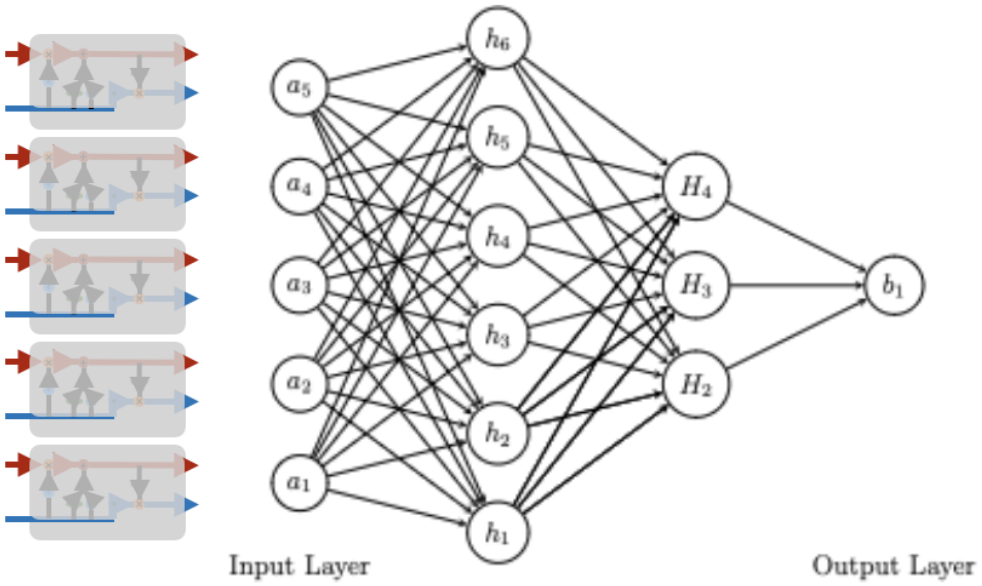

Figure 3: LSTM and Downstream Neural Network

value computed by the planner, y_i is the prediction by our learner. We predict the makespan but do not report it because it is not a part of the variable ordering process for slot matching. If we select the optimum slot for the planner to validate, it will automatically calculate the makespan for that slot and action. However, we consider it in the loss function because that is the objective function in our case and gives useful information about guiding the search process.

Empirical risk or training loss for a set $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{n}$ is denoted by $L_S(\boldsymbol{\theta})$,

$$L_S(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i). \tag{1}$$

The population risk or expected loss is denoted by $L_{\mathcal{D}}(\boldsymbol{\theta})$.

$$L_{\mathcal{D}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{D}} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}), \tag{2}$$

where $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$ follows the distribution $\mathcal{D}$.

As seen in Figure 3, We pass in five requests, each with two feasible slots. Every request corresponds to one LSTM cell, as it keeps in memory the spatio-temporal difference in the timelines and locations of each request corresponding with the global state. The encoder-decoder layers in the LSTM cell have five layers each, and the downstream linear neural network layer has 64 nodes in the hidden layer with ReLU activation. Our algorithm is defined in 1 which shows how we translate this architecture as a slot search problem, and short-circuit the search process.

3 Experimental Results

For evaluation, a data set of five requests and five blocks is used, with randomly sampled data consisting of ten rail blocks to test generalization. We use an Apple Mac M1 CPU with 16GB of RAM for computation, and all the times quoted from here on are from the same machine. We also prepare a Multi-layer perceptron baseline for comparison, with the same architecture but no LSTM cells.

We train the LSTM model for 1200 epochs until we find convergence. As shown in figure 4, we get expected trends in the loss function. It converges around 500 epochs and continues to perform well.

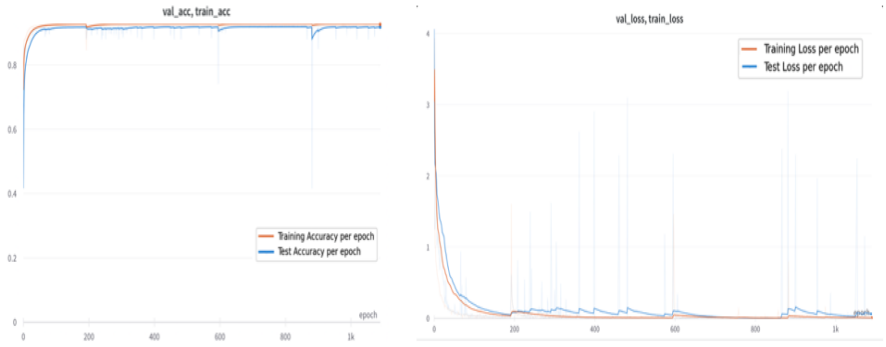

Figure 4: Accuracy curve for the train and test

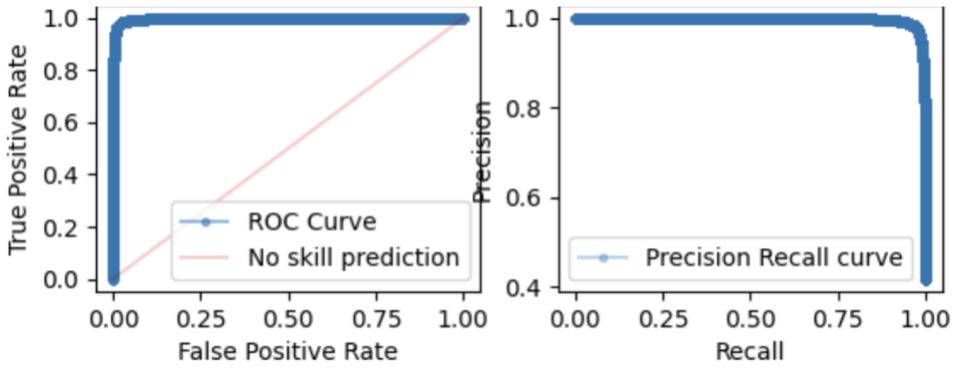

Figure 5: ROC and Precision-Recall Curve

3.0.1 Accuracy

We calculate accuracy as slot accuracy. It measures how often the LSTM model predicts the same slot as the T-HTN planner. As shown by Figure 4, we get a good test accuracy value of 90.2% around 1200 epochs, with high accuracy rates achieved in the early stages of the training. With the seq2seq modeling, the LSTM achieves better accuracy rates than the baseline. The limited dataset used in this article which are two slots per robot considered, does not do justice to the difference in accuracy with the baseline. As we keep increasing our sample space, the LSTM model accuracy will capture the spatial and temporal constraints in the memory and perform much better than the baseline.

We also calculate metrics similar to the MLP baseline to measure the model's performance, as shown in figure 6. ROC (Receiver Operating Characteristic) curve, AUC (Area Under the ROC Curve), and F1 score are standard evaluation metrics used in machine learning models to measure the model's ability to distinguish between positive and negative classes for classification problems.

All metrics show a positive trend, with an excellent F1 and AUC score. As shown in the Table 2. A high F1 score says our predictions have low false negatives and positives, and the model classifies well.

3.1 Computation Times

Our mission is to reduce the computation time for the slot-matching process. The novel thing about our approach is that no matter how many candidate slots are presented, the learner will take the

Table 2: Evaluation Metrics - Learn2Plan

Model	Accuracy	F1-Score	AUC-Score
Learn2Plan	90.2%	0.893	0.996
MLP baseline	86.3%	0.801	0.956

Table 3: Generalization accuracy

5 requests	MLP	Learn2Plan
10 rail blocks	76.11%	78.21%

same amount of time. However, the planner computation time will grow linearly with the number of candidate slots. For example, for a particular request, if ten slots have to be evaluated by the planner, following the Learn2Plan Heuristic will take approximately 1/10th of the computation time that the planner will take. We compare Learn2Plan with the planner T-HTN from which we have generated our training data, and POP-F [7] a forward chaining temporal planner used in task-planning and scheduling benchmarks. For every request, we compute the computation time improvement for our model (both LSTM and baseline) by the equation:

$$t_{improvement} = (t_{Planner})_{allslots} - t_{model}$$
$$t_{model} = (t_{model})_{modelinference} - (t_{Planner})_{modeloutputslot}$$

Where t is computation time. As shown in the table 4, we get computational speedups compared to both planners. The results are statistically significant, with p-scores of 0.012 and 0.014 for MLP and Learn2Plan, respectively, from a one-sample t-test. The MLP baseline achieves a 65.3% and 63.6% computational speedup compared to T-HTN and POP-F, respectively. The LSTM also fairs well with 14.6% and 15.3%, respectively, due to higher inherent complexity. However, we will achieve substantial improvement as we increase the number of slots per request, requests, and rail blocks. As accuracy is a feature for computational speedup, as it requires computing even to validate sub-optimal slots, LSTM will work reasonably well for complex cases with multiple slots.

3.2 Generalization

We generated sample data with five requests and ten discrete rail blocks to evaluate performance on unseen data. 10 Tasks were randomly sampled from this dataset and used with Learn2Plan to evaluate the generalizability of this approach. First, the mean accuracy is calculated in Table 3. We believe this is due to the difference in rail block locations, as it is from a different statistical distribution than that of 5 rail blocks. We also analyze feature importance, the spatial and temporal constraints, such as the pickup location, drop location, and the time bounds of the incoming item and the slots, have a high correlation. However, the LSTM model is also not dominated by specific features and understands the interplay of these features equitably, as seen in the last column.

The computation time is also measured to compare with the ten rail block test set that shows the numbers are relatively similar due to similar input and output sizes as shown in Table 4.

These results show that the model can reasonably generalize for uncertain requests.

4 Conclusion

As seen in our experimental results, an LSTM-based structure proves very well in learning an effective abstraction of the backtracking search of hierarchical planners. It substantially beats the

Table 4: Computation Metrics

5 requests	T-HTN	POPF	MLP	Learn2Plan
5 rail blocks	0.150	0.143	0.052	0.128
10 rail blocks	0.150	0.143	0.06	0.135

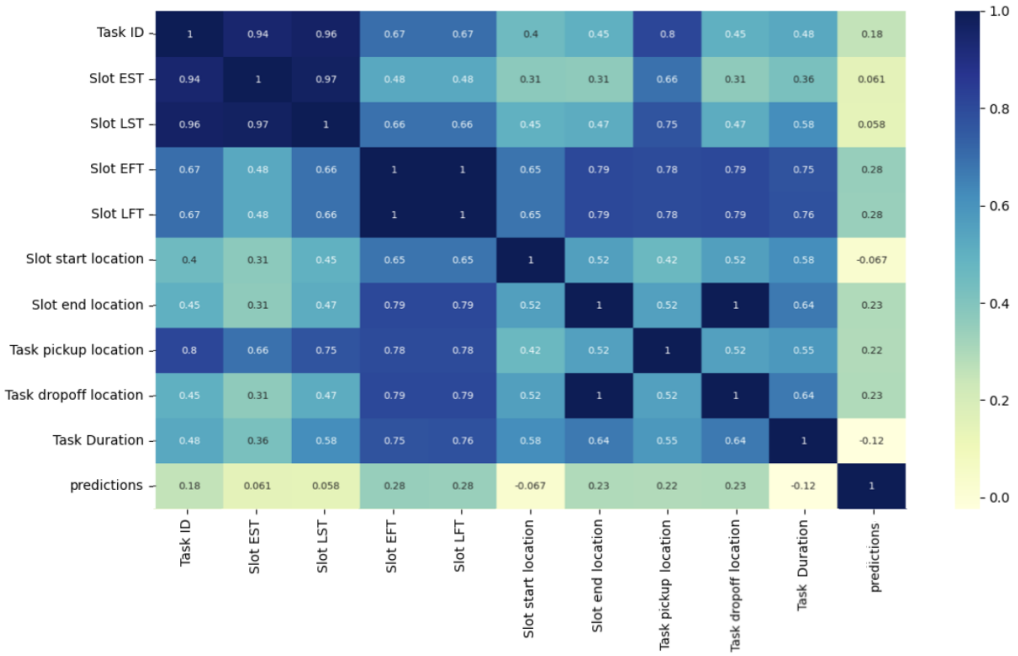

Figure 6: Correlation matrix of features

planner times in computational cost even in our restricted case of two requests, which can grow exponentially with more requests. It is observed that the feature heads pertinent to slot deadlines and durations have higher weights in the learning model, as well as spatial features like the location of pick up and drop off which is similar to exactly what a planner considers for decisions. It is able to generalize well and does not overfit to any spatial or temporal features of sample data, and proves that it is effective in understanding the nature of planning decisions due to the persistence in memory due to an LSTM cell understanding the inherent characteristic of requests and constraints. The work in the paper acts as a proof-of-concept for this methodology, and will follow on with wider analysis for more number of agents and requests. Hence, The future direction of this paper is expanding it to more diverse problems and benchmarks with various other state-f-the-art planners.

Acknowledgments

If a paper is accepted, the final camera-ready version will (and probably should) include acknowledgments. All acknowledgments go at the end of the paper, including thanks to reviewers who gave useful comments, to colleagues who contributed to the ideas, and to funding agencies and corporate sponsors that provided financial support.

References

[1] Moving around mars. https://mars.nasa.gov/mer/mission/timeline/surfaceops/navigation/#:~:text=Moving%20safely%20from%20rock%20to,about%2020%20minutes%20on%20average.

[2] V. Parimi, Z. B. Rubinstein, and S. F. Smith. T-htn: Timeline based htn planning for multiple robots. *HPlan 2022*, page 59.

[3] I. Sung, B. Choi, and P. Nielsen. Reinforcement learning for resource constrained project scheduling problem with activity iterations and crashing. *IFAC-PapersOnLine*, 53(2):10493–10497, 2020. ISSN 2405-8963. doi:https://doi.org/10.1016/j.ifacol.2020.12.2794. URL https://www.sciencedirect.com/science/article/pii/S2405896320335588. 21st IFAC World Congress.

[4] W. Song, Z. Cao, J. Zhang, C. Xu, and A. Lim. Learning variable ordering heuristics for solving constraint satisfaction problems. *Engineering Applications of Artificial Intelligence*, 109:104603, 2022. ISSN 0952-1976. doi:https://doi.org/10.1016/j.engappai.2021.104603. URL https://www.sciencedirect.com/science/article/pii/S0952197621004255.

[5] Z. N. Sunberg, Z. Yang, A. Mueller, F. Berkenkamp, and A. Schoellig. Adaptive entropy tree search for planning and learning. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4793–4802, 2018.

[6] A. Misra. Learn2plan: Learning variable ordering heuristics for scalable planning. 2023.

[7] A. Coles, A. Coles, M. Fox, and D. Long. Forward-chaining partial-order planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 20(1):42–49, May 2021. doi:10.1609/icaps.v20i1.13403. URL https://ojs.aaai.org/index.php/ICAPS/article/view/13403.