Combining Neural Networks and Tree Search for Task and Motion Planning in Challenging Environments

Chris Paxton[1,2], Vasumathi Raman[1], Gregory D. Hager[2], Marin Kobilarov[1,2]

I. Introduction

Robots in the real world have to deal with complex scenes involving multiple actors and complex, changing environments. Both its continuous motion in the physical world and the discrete goals it must accomplish are pertinent to correctly completing a complex task. In particular, self-driving cars are faced with a uniquely challenging task and motion planning problem that incorporates logical constraints with multiple interacting actors in a scene that includes other cars, pedestrians, and bicyclists.

Current methods for task and motion planning (TAMP) succeed at solving many sequential path planning and spatial reasoning problems [8], but the combined discrete and continuous state space tends to explode in size for complex problems. The addition of linear temporal logic constraints makes the search problem even more difficult, though there has been recent progress in this direction [6]. On the other hand, recent work in Deep Reinforcement Learning (DRL) has shown promise in challenging domains including autonomous driving[1, 9], and has been combined with Monte Carlo Tree Search (MCTS) for game playing [7], where it was able to achieve master-level performance. However, the question remains open whether these approaches can be integrated to produce reliable robot behavior.

We achieve the best of both worlds by using neural networks to learn both low-level control policies and high-level action selection priors, and then using these multi-level policies as part of a heuristic search algorithm to achieve a complex task. We formulate task and motion planning as a variant of Monte Carlo Tree Search over high-level options, each of which is represented by a learned control policy, trained on a set of Linear Temporal Logic (LTL) formulae [2]. LTL is an expressive language that has been used to concisely and precisely specify a wide range of system behaviors for robots. This approach allows us to efficiently explore the relevant parts of the search space to find high quality solutions when other methods would fail to do so. Fig. 1 shows some scenarios to which our algorithm was applied. For more details, see the full version of the paper [5].

II. Approach

For this work, we assume the existence of a simulator for the environment. One of our goals is to achieve robust behavior in complex scenes containing multiple other entities with relatively few simulations. In a dynamic environment with many actors and temporal constraints, decomposing the problem into reasoning over goals and trajectories separately is infeasible.

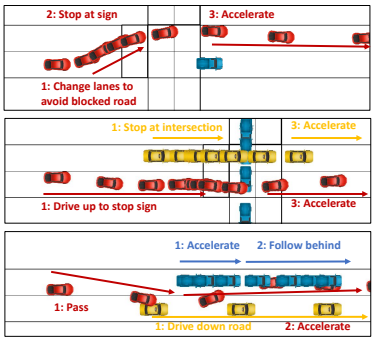

Fig. 1: Solutions to simulated self driving car problems containing an intersection and multiple vehicles.

Instead, we use learned policies together with an approach based on a variant of MCTS with two specializations. We learn two types of policies: a policy $\pi^*_{\mathcal{U}}(\cdot, o)$ for each high-level option o that maps from arbitrary feature values to controls:

$$\pi^*_{\mathcal{U}}(\phi(xw), o) = \arg\max_u(V^*(\delta(xw, uo)))$$

We also compute a second policy over options, $\pi^*_{\mathcal{O}}$:

$$\pi^*_{\mathcal{O}}(\phi(xw)) = \arg\max_o(V^*(\delta(xw, \pi^*_{\mathcal{U}}(\phi(xw), o)o)))$$

Planning is a variant of Monte Carlo Tree Search. We choose the next option to explore from a particular world state $s)i-1$ according to:

$$Q(s_i, o_i) = Q^*(s_i, o_i) + C\frac{P(s_i, o_i)}{1 + N(s_i, o_i)}$$

where $Q^*(s_i, o_i)$ is the average value of option o_i from simulated play, $N(s_i, o_i)$ is the number of times option o_i was observed from s_i, $N(s_i)$ is the number of times s_i has been visited, and $P(s_i, o_i)$ is the predicted value of option o_i from state s_i. The goal of this term is to encourage useful exploration while focusing on option choices that performed well according to previous experience; it grants a high weight to any terms that have a high prior probability from our learned model. We use MCTS with Progressive Widening to limit the number of new nodes added to the search tree. Information from the execution is used to expected reward

[1]Zoox, Inc., Menlo Park, CA, USA {chris.paxton,vasu,marin}@zoox.com
[2]The Johns Hopkins University, Baltimore, MD, USA cpaxton@jhu.edu, hager@cs.jhu.edu

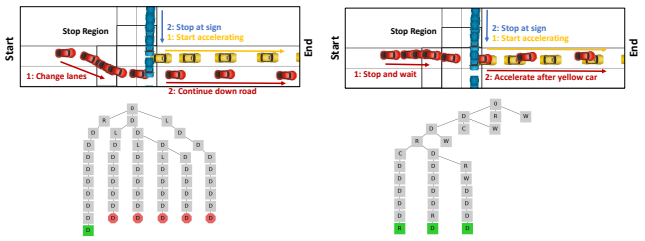

Fig. 2: Comparison of MCTS on a test problem with a stopped car. Letters indicate option being executed: 0 = root, D = default "stay in lane" policy, W = wait, C = Finish/complete level, R = lane change to the right. On the left, we see tree search with a manually defined preference; on the right, we see the tree using the high-level policy acquired through DQN. Green leaves indicate success; red leaves indicate failure. The right side finds a higher-reward solution faster.

following each high-level decision, allowing us to integrate low and high level decision making.

Each discrete option is associated with an LTL formula φ_o which establishes conditions that must hold while applying that option. We can evaluate $u_i = \pi_{\mathcal{U}}(o, \phi(x_i w_i))$ to get the next control as long as φ_o holds. In addition, we have a shared set Φ of LTL formulae that constrain the entire planning problem. At every step, we check whether a sampled trajectory satisfies all associated LTL formulae to ensure it meets safety conditions.

We apply our approach to the problem of planning for a self-driving car passing through an all-way stop intersection. Our scenarios take place at the intersection of two two-lane, one-way roads. Stop signs are described as "stop regions": areas on the road that vehicles must come to a stop in before proceeding. Other vehicles follow a manually defined driving policy, designed for good performance under expected driving conditions. High level policies correspond to lane changes, stopping at stop signs, following a car, or continuing in the same lane.

The reward function is a combination of a a cost term based on the current continuous state and a bonus based on completing intermediate goals or violating constraints (e.g. being rejected by the DRA corresponding to an LTL formula). The cost term penalizes the control inputs, acceleration and steering angle rate, as well as other terms determined from the state of the simulated vehicle. We add a terminal penalty of -100 for trajectories that hit obstacles or violate constraints.

All control policies were represented as multilayer perceptrons with a single hidden layer of 32 fully connected neurons. We used the ReLu activation function on both the input and hidden layer, and the tanh activation function on the outputs. Outputs mapped to steering angle rate $\dot{\psi} \in [-1, 1]$ rad/s and acceleration $a \in [-2, 2]$ m/s^2. Control policies were trained according to the Deep Direct Policy Gradients algorithm [3]. We then performed Deep Q learning [4] on the discrete set of options to learn our high-level options policy. High-level policies were trained on a challenging road environment with 0 to 6 randomly placed cars with random velocity, plus a 50% chance of a stopped vehicle ahead in the current lane.

III. Results and Conclusions

We generated 100 random worlds in a new environment containing 0-5 other vehicles. We also test in the case with an additional stopped car in the lane ahead. For cases with the learned or simple action sets, we performed 100 iterations of MCTS to a time horizon of 10 seconds and select the best path to execute. Fig. 2 shows how the algorithm works in practice with different methods for choosing the high-level policy. With the learned high-level policy, we see excellent performance on the test set for simple problems and three failures in complex problems. These failures represent cases where there was a car moving at the same speed in the adjacent lane and a stopped car a short distance ahead, and there was no good option to explore. Our system avoids these situations where it is possible. When it predicts that such a situation will arise, our planner would give us roughly 2 seconds of warning to execute an emergency stop and avoid collision.

Our approach allows off-the-shelf DRL techniques to better generalize to challenging new environments, and allows us to verify their behavior in these environments by checking LTL constraints during execution. We use learned neural nets to prune possible solutions when performing task planning. In the future, we will extend this work to use stochastic control policies, and will also apply our system to real robots. For more details, see the full version of the paper [5].

References

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT Press, 2001. ISBN 978-0-262-03270-4.

[3] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] Chris Paxton, Vasumathi Raman, Gregory D. Hager, and Marin Kobilarov. Combining neural networks and tree search for task and motion planning in challenging environments. *CoRR*, abs/1703.07887, 2017. URL `http://arxiv.org/abs/1703.07887`.

[6] Erion Plaku and Sertac Karaman. Motion planning with temporal-logic specifications: Progress and challenges. *AI Communications*, (Preprint): 1–12.

[7] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[8] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence*, 2015.

[9] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. *arXiv preprint arXiv:1612.01079*, 2016.