Hierarchical Deep Reinforcement Learning through Scene Decomposition for Autonomous Urban Driving

Peggy (Yuchun) Wang [1] Maxime Bouton [2] Mykel J. Kochenderfer [3]

Abstract

Reinforcement learning combined with utility decomposition techniques have recently demonstrated the ability to scale existing decision strategies to autonomous driving environments with multiple traffic participants. Although these techniques are promising, it is not clear how their performance would generalize past the demonstrations on a limited set of scenarios. In this study, we investigated the possibility of fusing existing micro-policies to solve complex tasks. Specifically, we applied this method to autonomous urban driving by developing a high-level policy composed of low-level policies trained on urban micro-scenarios using hierarchical deep reinforcement learning. To demonstrate this, we solved for low-level micro-policies on a simple two-lane left-lane change scenario and a simple single-lane right-turn merging scenario using Deep Q-Learning. We used utility decomposition methods to solve for a policy on a higher level composite scenario given as a two-lane right turn merging scenario. We achieved promising results using utility decomposition compared to the baseline policy of training directly on the complex scene. In the future, we plan on developing a city-level policy composed from multiple micro-policies by continuing to develop an algorithm that efficiently and accurately decomposes scenes.

1. Introduction

One of the challenges facing autonomous driving in urban environments is decision making under uncertainty in many different traffic scenarios. Factors such as environmental dynamics, interactions with other drivers, and generalization of road scenarios are difficult. Particularly challenging is generalization of many different policies given different road topologies.

In practice, most autonomous vehicles use a state machine to switch between predefined behaviors (Schwarting et al., 2018). However, these rule-based approaches make it difficult to generalize when dealing with a scenario that is not defined in their state machines. Additionally, the rule-based approaches are not scalable as there is no guarantee that every possible state that will ever be encountered will be encoded in the state machine, which means that when encountering states not encoded in the state machine, the agent will not know what to do.

Currently, literature on an alternative to a rule-based approach to decision making include game theoretic approaches with multiple agents (Fisac et al., 2018) or solving through Deep Reinforcement Learning (DRL) on simple scenarios such as lane-changing scenarios (Wang et al., 2018). However, these approaches all contain limitations. Although game theoretic approaches perform well in regards to solving for the model of another agent, it does not scale to multiple agents. An advantage of using DRL to solve for policies is that it handles continuous spaces as opposed to only discrete spaces handled by the rule-based approaches. Nonetheless, DRL faces the same general problem as the rule-based approaches - it is not scalable because the solver would have to be run on every single scenario that could ever exist. Moreover, DRL also is expensive to train and compute, and would require a large amount of time to train, especially if it needs to be trained on every possible scenario.

To address this issue, in this study we chose to focus on the use of utility decomposition on complex scenarios. To the best of our knowledge, this is the first work that has been done on this problem of planning generalization.

We investigate the possibility of developing a high-level policy composed of low-level policies trained on micro-scenarios using hierarchical deep reinforcement learning (DRL). For example, if we have a policy for a left turn at a T intersection, one for a crosswalk, one for a round-about, and

[1]Department of Computer Science, Stanford University [2]Department of Aeronautics and Astronautics, Stanford University, not enrolled in CS234 [3]Department of Aeronautics and Astronautics, and by courtesy, Department of Computer Science, Stanford University, not enrolled in CS234. Correspondence to: Peggy (Yuchun) Wang <peggy.yuchun.wang@cs.stanford.edu>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

one for lane change, we then use the knowledge from these micro-policies to adapt to any driving situation. A double lane round-about could perhaps be seen as a composition of a single-lane round-about policy and a lane change policy.

A key limitation of our approach is the method of decomposition. In this study, we handcrafted a composition scenario that is able to decompose into micro-scenarios. However, in the future, we would like to investigate algorithms that are able to automatically decompose a complicated scenario into a predefined set of micro-scenarios.

The remainder of this paper presents a utility decomposition method using two low-level scenarios. To demonstrate this, we solved for low-level micro-policies on a simple two-lane road left lane-change micro-scenario and a simple single-lane right-turn intersection scenario using Deep Q-Learning Networks (DQN). We then used utility decomposition methods to solve for a policy on a high-level composite scenario represented as a two-lane right-turn merging scenario. We compared our policy generated using utility decomposition with the baseline policy of directly training on the composite scenario using DQN. Our utility decomposition policy has comparable performance to the baseline policy and shows that complex policies can effectively be approximated using utility decomposition.

2. Related Work

2.1. Deep Reinforcement Learning for Autonomous Vehicle Policies

In recent years, work has been done using Deep Reinforcement Learning to train policies for autonomous vehicles, which are more robust than rule-based scenarios. For example, Wang et al. has developed a lane-change policy using DRL that is robust to diverse and unforeseen scenarios (Wang et al., 2018). Moreover, Wolf et al. has used DRL to to learn maneuver decisions based on a compact semantic state representation (Wolf et al., 2018). Chen et al. has used DRL to select the best actions during a traffic light passing scenario (Chen et al., 2018).

2.2. Policy Decomposition

In recent years, advances have been made in the area of policy decomposition. Zhang et al. has decomposed a policy for a block stacking robot by decomposing a complex action to simpler actions (Zhang et al., 2018). Liaw et al. has trained a meta-policy using Trust-Region Policy Optimization (TRPO) based on simpler basis policies (Liaw et al., 2017).

2.3. Utility Decomposition

Utility decomposition methods are similar to policy composition methods, except that they decompose the state-value function (also called the utility function) rather than the policy. The concept of value decomposition was first described as Q-decomposition by Russell and Zimdars (Russell & Zimdars, 2003), where multiple lower-level state-action values functions are summed together. Recently, methods such as Value Decomposition Networks (Sunehag et al., 2017), and QMIX (Rashid et al., 2018) build upon this idea. Value Decomposition Network methods generate a state-action value function summed from from low-level value functions generated from DRL networks. QMIX fuses low-level value functions by using DRL to train weights and biases for each low-level value function before summing them together.

Bouton et al. applied utility decomposition to the field of autonomous driving (Bouton et al., 2018). The low-level value functions were trained using DRL on a single agent and then fused together. A deep neural network was used to train a correction factor before summing the low-level value functions to create an value function approximation for multiple agents. This method was applied to autonomous driving and was able to approximate the value function of crosswalks with multiple agents from fusing the value function of crosswalks with a single agent. Although these techniques are promising, it is not clear how their performance would generalize past the demonstrations on a limited set of scenarios.

3. Background

We modeled the urban traffic scenarios as a sequential decision making problem, in which we optimized for the highest reward. We formulated the environment and dynamics as a fully-observable Markov Decision Process (MDP) (Kochenderfer, 2015).

3.1. Reinforcement Learning

We formulated the problem using a reinforcement learning framework (Sutton & Barto, 2018), where the agent sequentially interacts with the environment over a series of timesteps. We modeled the environment as a Markov Decision Process (MDP). We formally defined a MDP as a 5-element tuple of (S, A, T, R, γ), where S is the state space, A is the action space, T is the state transition function, R is the reward function, and γ is the discount factor. At each timestep t, the agent chose an action $a_t \in A$ based on observing state $s_t \in S$. The agent then receives a reward $r_t = R(s_t, a_t)$. In the next timestep t+1, the environment transitions to a state s_{t+1} based on the probability given by the transition function $Pr(s_{t+1}|s_t, a_t) = T(s_{t+1}, s_t, a_t)$. The agent's goal was to maximize the expected cumulative discounted reward given by $\sum_{t=0}^{\infty} \gamma^t r_t$.

A policy π is defined as a function mapping from states to probability of distributions over the action space, where $\pi : S \rightarrow Pr(A)$. The agent probabilistically chooses an action based on the state. Each policy π is associated with a state-action value function Q^π, representing the expected discounted value of following the policy π. An optimal state-action value function of a MDP $Q^*(s,a)$ satisfies the Bellman Equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s', s, a) max_{a'} Q^*(s', a') \tag{1}$$

where s is the current state, a is the action taken, s' is the next state reachable from action a, and a' is the next action in state s'. An optimal policy $\pi^*(s) = argmax_a Q^*(s,a)$ takes the argmax of the actions over the value function. The utility of the given state was defined as $U(s) = max_a Q^*(s,a)$.

3.2. Utility Decomposition

Utility decomposition, sometimes called Q-decomposition (Russell & Zimdars, 2003), involves combining the action value functions of simple tasks to approximate the action value function of a more complex task, assuming that the simple tasks are a substructure of the more complex task. This hierarchical structure of tasks solved using reinforcement learning is an example of a Hierarchical Reinforcement Learning (HRL). In the general case, the agent's $Q^*(s,a) = f(Q_1^*(s,a), ..., Q_n^*(s,a))$, where Q^* is the optimal state-action value function of the agent, and Q_n is the state-action value function each of the agent's subtasks. Examples of the functions used in utility decomposition and HRL include Q-decomposition (Russell & Zimdars, 2003), where $Q^*(s,a) = \sum_{i=1}^{n} Q_i^*(s,a)$ and Value Decomposition Networks (VDN) (Sunehag et al., 2017), where $Q(s,a) = \sum_{i=1}^{\infty} Q_i(s,a)$. In this study, when we have a single composite scenario comprised of two micro-scenarios, we can represent the value function of the combined scenario Q_{comp} as:

$$Q_{comp}(s,a) = Q_1(s,a) + Q_2(s,a).$$

4. Approach

We used the AutoViz.jl driving simulator (`https://github.com/sisl/AutoViz.jl`) developed by the Stanford Intelligent Systems Lab (SISL) for traffic, roadways, and driver model simulations. We first solved for low-level policies offline on micro-scenarios using Deep Q-Learning on AutoViz.jl. We then decomposed the complex merging scenario (Figure 3) into two micro-scenarios (Figures 9 and 10) using the spatial road representations and developed a value function of the complex scenario using after fusing the value functions of the low-level policies. Lastly, we extracted the policy from taking the argmax of the fused value function.

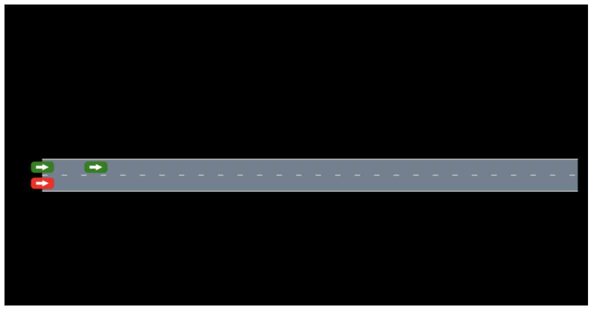
Figure 1. Starting state of the AutoViz.jl simulation of the lane-change micro-scenario. The ego vehicle is red and the obstacle cars are green. The ego car needs to drive from that starting point to the goal position of the end of the left lane without crashing into the obstacle cars.

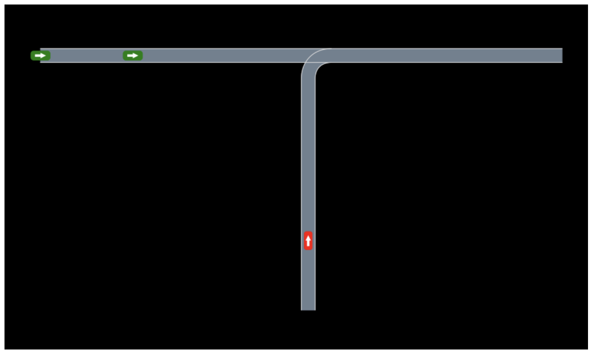
Figure 2. Starting state of the AutoViz.jl simulation of the right-turn merging micro-scenario. The ego vehicle is red and the obstacle cars are green. The ego car needs to drive from the starting point to the goal position of the end of the lane on the far right without crashing into the obstacle cars.

4.1. Modeling Driving Scenarios as Markov Decision Process

The simulation environment was modeled as an MDP. The state included the roadway structure, the position in Frenet frame, and Euclidean velocity of every car in the scene. Transition functions were given by the simulation moving forward one time step deterministically. We developed the reward function (Algorithm 1) as receiving a normalized reward of +1 at the goal position, and defined a collision and off-road position as receiving a reward of -1. If the ego vehicle reached either one of the three previous states, the state becomes a terminal state. If the car was on the road but not yet at the goal state, our reward is -0.01 times the normalized distance to the goal. We discretized the actions by defining the action space as combinations of longitu-

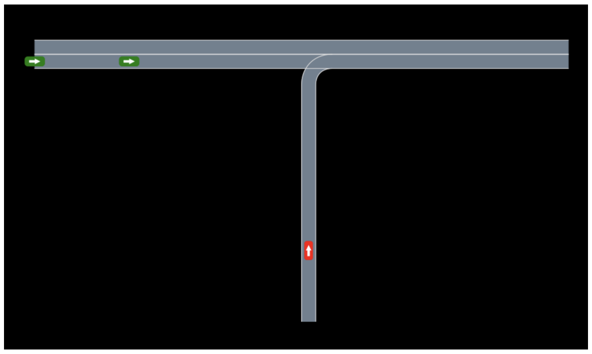

Figure 3. Starting state of the AutoViz.jl simulation of the two-lane right-turn combined merging scenario. The ego vehicle is red and the obstacle cars are green. The ego car needs to drive from that starting point to the goal position of the end of the leftmost lane.

Algorithm 1 Reward Function

```
Require: state
Ensure: reward
  if isCollision(state) or isOffRoad(state) then
    reward = -1.
  else if reachGoal(state) then
    reward = 1.
  else
    reward = -0.01 * distanceToGoal(state)
  end if
  return reward
```

dinal acceleration between $-2.0m/s^2$ and $2.0m/s^2$ in a $1.0m/s^2$ interval and and lateral steering angles accelerations between $-1.0rad/s^2$ and $1.0rad/s^2$ in a $0.1rad/s^2$ interval.

We modeled the position of the car based on the Frenet frame instead of the Euclidean frame, where each Frenet frame corresponded to each lane. Frenet.s referred to the longitudinal position of the car along the lane starting from the lane origin and Frenet.t referred to the latitudinal position of the car starting from the center of the lane.

4.2. Deep Q-Learning

We used the DeepQLearning.jl package from JuliaPOMDP to develop our Deep Q-Network (DQN). We defined a DQN with 2 hidden layers of 32 units each and used a Rectified Linear Units (Relu) activation function for each layer. We passed in a 1×4 state vector for each vehicle. Our output of the DQN is a policy for the scenario being trained on. Additional hyperparameters for our DQN may be found in the Appendix.

We implemented the input state representation passed into the neural network as an $n \times 4$ dimensional vector, where there are n total number of cars in the scene. Each car has 4 elements representing its state: the state of its s (longitudinal) position in the Frenet frame, its velocity in Euclidean space, and a one-hot vector representation of the lane it is currently in. We normalized the Frenet s position by the total road length and the speed by the maximum speed to facilitate training and increase time of convergence. The ego car is always represented by the first four elements of the $n \times 4$ dimensional vector. In our scenarios, since $n = 3$, we pass in a 12-dimensional vector to the DQN.

4.3. Utility Decomposition

We assumed that the combined two-lane merging scenario (Figure 3) may be fully decomposed into a two=lane change micro-scenario (Figure 1) and a single-lane merging micro-scenario (Figure 10). This assumption meant that utility decomposition methods could be used to solve this problem, specifically that

$$Q^*_{comp}(s,a) = f(Q^*_1(s,a), Q^*_2(s,a)),$$

where Q^*_{comp} was the optimal state-action value function for the combined two-lane merging scenario, Q^*_1 was the optimal state-action value function for the lane-change scenario, and Q^*_2 was the optimal state-action value function for the single-lane merging scenario.

Under this assumption, we could approximate the optimal state-action value function Q^*_{comp} as a linear combination of the value functions of the decomposed micro-scenarios, where $Q^*_{comp}(s,a) \approx Q^*_1 + Q^*_2$. We estimated Q^*_1 and Q^*_2 by training a DQN to estimate the state-action values trained on the corresponding micro-scenarios. We call these estimates $\tilde{Q}^*_1$ and $\tilde{Q}^*_2$, respectively. Therefore, the estimate of the value function Q^*_{comp} may be represented as $\tilde{Q}^*_{comp}$, where

$$\tilde{Q}^*_{comp}(s,a) = \tilde{Q}^*_1(s,a) + \tilde{Q}^*_2(s,a).$$

We then extracted an estimate of the optimal policy $\tilde{\pi}^*_{comp}$ using $\tilde{Q^*}_{comp}(s,a)$, where

$$\tilde{\pi}^*_{comp} = argmax_a \tilde{Q}^*_{comp}(s,a).$$

Algorithm 2 Utility Decomposition Algorithm

```
Require: simplePolicy1, simplePolicy2, state
Ensure: combinedPolicy
  simpleState1, simpleState2 $\leftarrow$ decompose(state)
  QCompNetwork = actionValues(simpleState1,
      simplePolicy1) + actionValues(simpleState2,
      simplePolicy2)
  combinedPolicy = argmax(QCompNetwork)
  return combinedPolicy
```

4.4. Decomposing Complex Scenarios into Micro-scenarios

We developed a utility decomposition algorithm shown in Algorithm 2. We decomposed a complex state by mapping every position on the full scenario to a position on each micro-scenario. For our two-lane right-turn merging scenario, we mapped the starting lanes of the ego and obstacle cars to the right lane of the two-lane micro-scenario, and we mapped the leftmost lane to the left lane of the two-lane micro-scenario. We also mapped the two horizontal lanes on the full scenario onto the single horizontal lane on the right-turn merging micro-scenario, and we mapped the vertical lane of the ego vehicle onto the vertical lane of the micro-scenario. To adjust for different lengths of the road, we normalized the road lengths when we decomposed into the micro-scenarios.

5. Results

5.1. Experiments

We developed a series of experimental scenarios using the AutoViz.jl simulator. The codebase is currently hosted at `https://github.com/PeggyYuchunWang/Deep-HRL-for-Scene-Decomp`.

The AutoViz.jl simulation environment was first set up in Julia. We then created a simple two-lane change micro-scenario, where the world was composed of a straight two-lane road with two other obstacle cars (green) in addition to the ego vehicle (red). This starting state is shown in Figure 1. The ego vehicle needs to drive from the start of the right lane to the goal of the end of the left lane without crashing into the obstacle vehicles or going off-road. We also created a simple single-lane right-turn merging micro-scenario shown in Figure 2. The ego vehicle needs to drive from the start of the vertical lane to the goal of the end of the horizontal lane without crashing into the obstacle vehicles or going off-road.

We modeled the world as an MDP, where we created a class called DrivingMDP for which the state representation of the MDP, reward function, discretized action spaces, and transition functions were implemented. We used the POMDPs.jl framework to create the MDP. We also implemented a lookahead function to prevent the car from going off-road and crashing into an obstacle. The lookahead function was then masked with the action space to create a safe action space for faster training. Deep Q-Learning then was used with two hidden layers on the DrivingMDP model to learn a successful policy of changing lanes. We assumed that the obstacle cars are going at constant speed and direction using a constant driver model, and that the cars start at an urban speed of 10.0 m/s. We also assumed the world was deterministic.

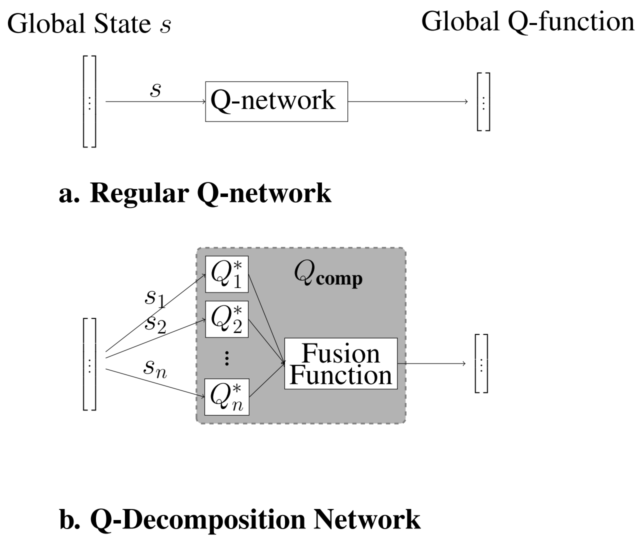

Figure 4. Architecture Comparison of a Regular Q-Network and a Q-Decomposition Network. (a) shows the architecture of a Regular Q-Network. (b) shows the architecture of a Q-Decomposition Network. Figure revised from (Bouton et al., 2018).

5.2. Description of Results

We successfully trained micro-policies using Deep Q-Learning for a simple lane-change micro-scenario (Figure 9 in Appendix) and a right-turn merging micro-scenario (Figure 10 in Appendix). We clearly see that the policy of the right-turn micro-scenario is better than a naive constant policy of 0.0 longitudinal and latitudinal acceleration, because the ego vehicle will crash into the green obstacle vehicle using the naive policy. The crash in the naive policy is shown in the Appendix in Figure 11. Therefore, we see that the network successfully learns a more optimal policy than the naive policy.

5.2.1. Baseline

Our baseline policy was a policy trained using DQN on the full scenario, since such a policy is a close estimate of the optimal policy on that scenario. Therefore, in a situation of infinite samples, our baseline should be converge to the optimal policy for that scenario and would be theoretically better than our Q-decomposition policy. This is true because Russel and Zimdars showed that if we trained two policies separately and then sum them together to get an approximation of an optimal policy, the Q-decomposition policy would be suboptimal because it is an approximation of the optimal value function (Russell & Zimdars, 2003).

We successfully trained a baseline policy using DQN directly on the full scenario. Using the baseline policy (Figure 7), the ego vehicle reached the goal position in 7 timesteps with an evaluation reward of .973. The average rewards over

Table 1: Results

Policy Name	Evaluation Reward	Timesteps
Baseline DRL	0.973	7
Q-Decomposition	0.968	9
Lane Change	0.960	8
Right Turn Merge	0.970	8

Figure 5. Evaluation reward and number of timesteps taken to reach goal for different scenarios. Baseline DRL refers to the policy achieved by training DQN on the composite scenario (Figure 7) and Q-Decomposition refers to the policy extracted from a fusion of the state-action value functions of the micro-scenarios (Figure 8). Lane change refers to the DQN policy trained on the two-lane change micro-scenario (Figure 9 in Appendix). Right-turn merge refers to the DQN policy trained on the right-turn single-lane micro-scenario (Figure 10 in Appendix).

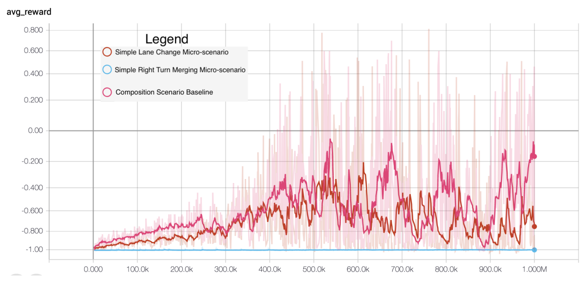

Figure 6. Average Reward for Scenarios trained using DRL

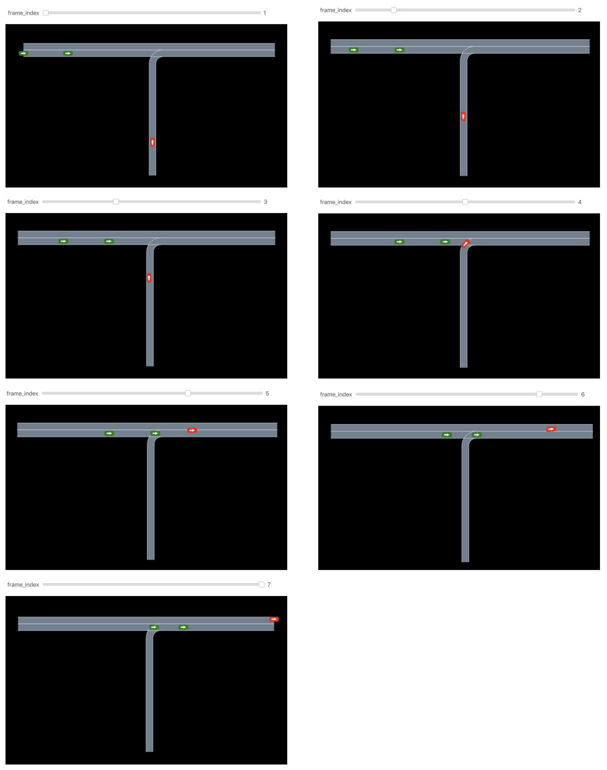

Figure 7. Baseline Policy, starting from top left frame 1 to bottom left frame 7

time for each of the scenarios trained using DQN are shown in Figure 6.

5.2.2. Q-DECOMPOSITION

We then implemented Q-decomposition using Algorithm 2, where we passed in the micro-policy of the two-lane change micro-scenario, the micro-policy of the single-lane change right-turn micro-scenario, and the initial state of the full composed scenario. The architecture of the Q-decomposition Network is shown in Figure 4. The visualization of the micro-policies may be found in the Appendix. After summing up the Q-networks functions of each of the micro-scenarios, we extracted a policy of the composed scenario by taking the argmax of the combined Q-network. We see that using the composed policy (Figure 8), the ego vehicle successfully reached the goal in 9 timesteps with an evaluation reward of .968.

Our results are shown in Figure 5. All of the four policies enabled the agent to successfully reach the goal in its corresponding scenario. We compared the evaluation reward and number of timesteps taken to reach the goal in each of the four policies. The policies are the baseline policy (Figure 7), Q-decomposition policy (Figure 8), lane-change micro-policy (Figure 9 in Appendix), and right-turn merge micro-policy (Figure 10 in Appendix).

We also showed the results of our training data for the three policies trained using DQN over 1 million iterations. Average rewards for the baseline policy, lane-change micro-policy, and right-turn merge micro-policy over time are shown in Figure 6. The evaluation reward and loss over time are shown in the Appendix.

5.3. Discussion of Results

We see that our baseline policy is slightly better than the policy using Q-decomposition, as shown in the 7 timesteps as compared to the 9 timesteps and .973 evaluation reward compared to the .968 evaluation reward. This is expected as the policy trained using DQN on that specific scenario will perform well. If we consider the baseline policy as a close to optimal policy, we see that the policy extracted using Q-decomposition is very close to the to the optimal policy in terms of performance. Q-decomposition is also computationally more efficient than training DQN on the composed scenario, since it does not require retraining of the entire Q-network and rather just sums the Q-networks of the simpler policies.

We also see that utility decomposition is less expensive to compute. Once the micro-policies are extracted, Q-

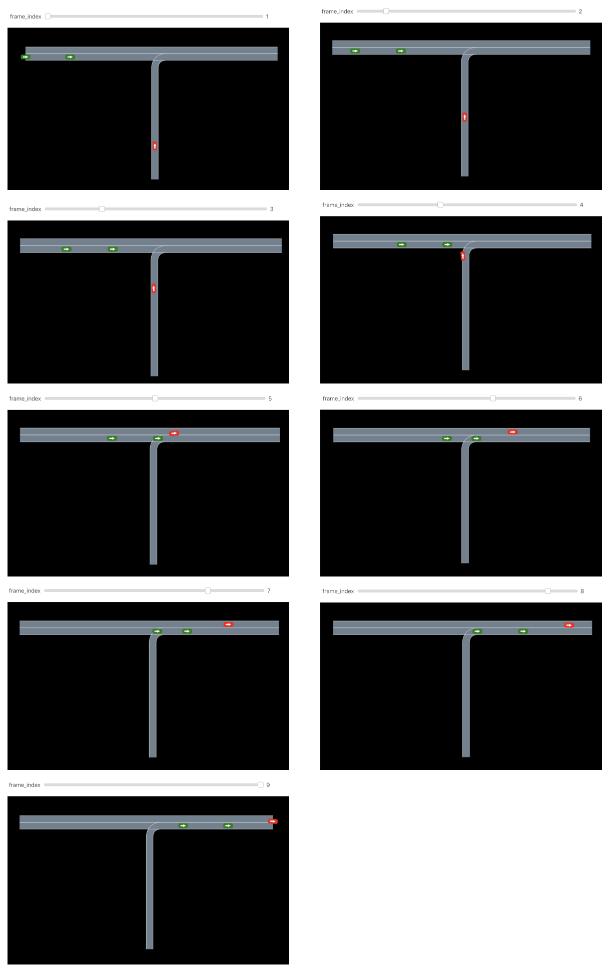

Figure 8. Visualization of the Q-decomposition policy. Starting from top left frame 1 to bottom left frame 9.

decomposition fuses them together without any additional training. However, using the baseline approach of solving for a close-to-optimal policy requires training for 1 million iterations, taking 45 minutes on a Macbook Pro.

This shows the tremendous power of Q-decomposition, where we are able to approximate the optimal policies online of many complex scenarios from simple micro-scenarios trained offline, even if we have not seen the more complex scenario before. This has many applications especially in the field of autonomous driving. A key limitation of the baseline policies trained using DQN or rule-based approaches is that they only work for the scenarios they were trained on and perform poorly on other scenarios. They are not able to generalize. Q-decomposition is able to address this limitation by ensuring that we will always be able to generalize from every scenario, if we have a decomposition function that decomposes a complex scenario onto trained micro-scenarios. We then would be able to develop a city-wide policy by composing scenarios from a set of micro-policies.

6. Conclusions and Future Work

Utility decomposition methods efficiently find approximate solutions to decision making problems when the complex problem can be broken down into simpler problems. In this study, we have shown that once a set of solutions can be computed on a set of micro-scenarios, the micro-policies may be combined to solve a harder problem of a complex road scenario. Although these methods have been applied to other tasks, in this study we created a novel technique to generalize utility decomposition to autonomous driving policies using scene decomposition.

Our ultimal goal for this project is to compose a general city-level policy based on several micro-policies. To accomplish this goal, we need to learn several other low-level policies on micro-scenarios such as roundabout scenarios, left-turn scenarios, and stop intersection scenarios. We also plan to investigate an efficient scene decomposition algorithm that is able to automatically decompose a high-level scene into a micro-scenario with efficiency and high degrees of accuracy.

Additionally, to achieve the scene decomposition algorithm, we will develop a formalism for state decomposition for urban driving and investigate efficient state representation, such as using spatial or topical representation of scenarios. We also want to investigate how to these policies will interact with different driver models, a stochastic world, and multiple agents. We also will investigate how to generalize based on partial observability instead of full observability.

Acknowledgements

A tremendous thank you to my mentor Maxime Bouton for all his help and input, as well as conceiving the idea of this interesting project. A tremendous thanks as well to Professor Mykel Kochenderfer for allowing me to work on this independent project in conjunction with CS191W and providing help and mentorship. Thanks as well to Mary McDevitt for giving revision suggestions, especially during finals week.

References

Bouton, M., Julian, K., Nakhaei, A., Fujimura, K., and Kochenderfer, M. J. Utility decomposition with deep corrections for scalable planning under uncertainty. *CoRR*, abs/1802.01772, 2018. URL http://arxiv.org/abs/1802.01772.

Chen, J., Wang, Z., and Tomizuka, M. Deep hierarchical reinforcement learning for autonomous driving with distinct behaviors. pp. 1239–1244, 06 2018. doi: 10.1109/IVS.2018.8500368.

Fisac, J. F., Bronstein, E., Stefansson, E., Sadigh, D., Sastry, S. S., and Dragan, A. D. Hierarchical game-theoretic planning for autonomous vehicles. *CoRR*, abs/1810.05766, 2018. URL `http://arxiv.org/abs/1810.05766`.

Kochenderfer, M. J. *Decision making under uncertainty: theory and application*. MIT press, 2015.

Liaw, R., Krishnan, S., Garg, A., Crankshaw, D., Gonzalez, J. E., and Goldberg, K. Composing meta-policies for autonomous driving using hierarchical deep reinforcement learning. *CoRR*, abs/1711.01503, 2017. URL `http://arxiv.org/abs/1711.01503`.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N., and Whiteson, S. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. *CoRR*, abs/1803.11485, 2018. URL `http://arxiv.org/abs/1803.11485`.

Russell, S. J. and Zimdars, A. Q-decomposition for reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 656–663, 2003.

Schwarting, W., Alonso-Mora, J., and Rus, D. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):187–210, 2018. doi: 10.1146/annurev-control-060117-105157.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V. F., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. Value-decomposition networks for cooperative multi-agent learning. *CoRR*, abs/1706.05296, 2017. URL `http://arxiv.org/abs/1706.05296`.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Wang, P., Chan, C., and de La Fortelle, A. A reinforcement learning based approach for automated lane change maneuvers. *CoRR*, abs/1804.07871, 2018. URL `http://arxiv.org/abs/1804.07871`.

Wolf, P., Kurzer, K., Wingert, T., Kuhnt, F., and Zllner, J. Adaptive behavior generation for autonomous driving using deep reinforcement learning with compact semantic states. 06 2018. doi: 10.1109/IVS.2018.8500427.

Zhang, A., Lerer, A., Sukhbaatar, S., Fergus, R., and Szlam, A. Composable planning with attributes. *CoRR*, abs/1803.00512, 2018. URL `http://arxiv.org/abs/1803.00512`.

Contributions

- Peggy (Yuchun) Wang
  - Implemented full project and algorithms in Julia codebase
  - Wrote and edited paper and poster
- Maxime Bouton (PhD student mentor, not in CS234)
  - Provided project vision and ideas
  - Suggested literature review and code package resources
  - Mentored and discussed ideas about algorithms, simulation, and implementation
  - Gave input on edits for paper and poster
- Prof. Mykel J. Kochenderfer (Faculty advisor, not in CS234)
  - Faculty advisor
  - Provided mentorship and opportunity to work on independent project in conjunction with CS191W

Appendix

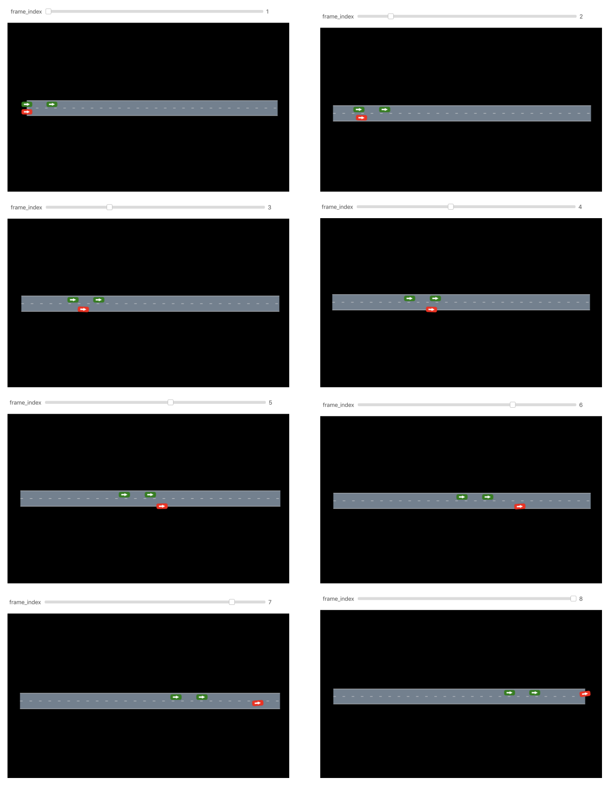

Figure 9. Visualization of the left lane-change micro-policy. Starting from top left frame 1 to bottom right frame 8.

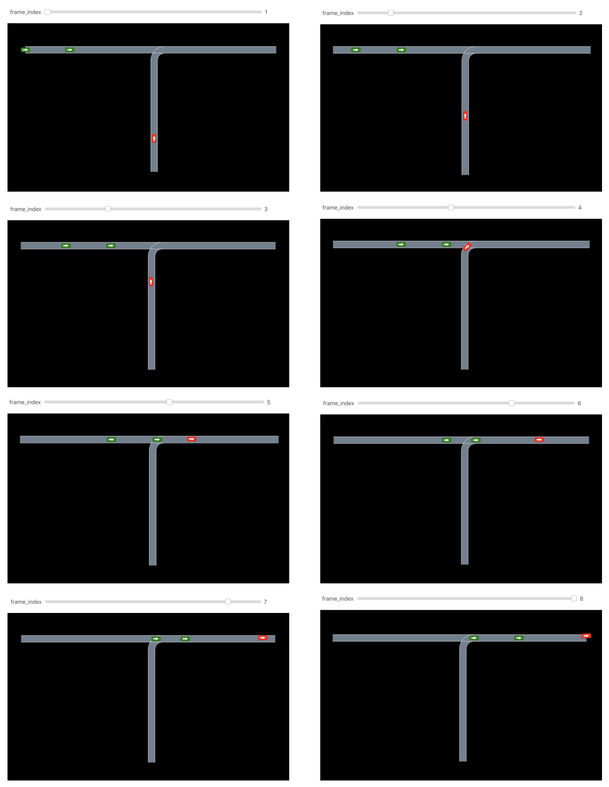

Figure 10. Visualization of the right-turn merge micro-policy. Starting from top left frame 1 to bottom right frame 8.

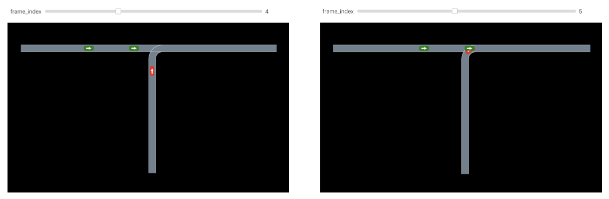

Figure 11. Visualization of the constant velocity policy crash for the right-turn merging micro-scenario.

Table 2: DQN Hyperparameters

Hyperparameter	Value
Fully Connected Layers	2
Hidden Units	32
Activation functions	Rectified linear units
Replay buffer size	400,000
Target network update frequency	3,000 episodes
Discount factor	0.9
Number of training steps	1,000,000
Learning rate	0.001
Prioritized replay	$\alpha = 0.6$, $\beta = 1 \times 10^{-6}$
Exploration fraction	0.5
Final ϵ	0.01

Figure 12. Hyperparameters of the Deep Q-Learning Network

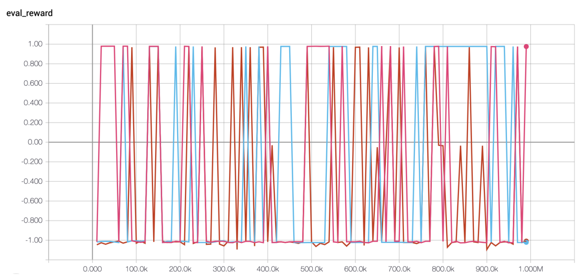

Figure 13. Evaluation Reward for Scenarios trained using DQN

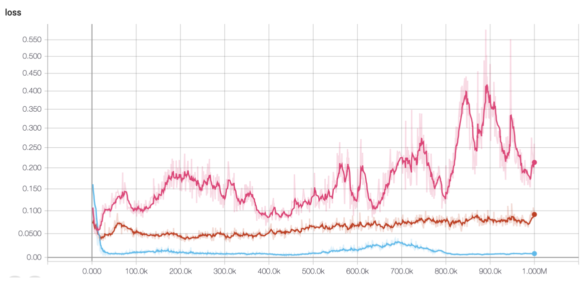

Figure 14. Loss for Scenarios trained using DQN