Utility Decomposition with Deep Corrections for Scalable Planning under Uncertainty

Maxime Bouton
Stanford University
Stanford, CA
boutonm@stanford.edu

Kyle Julian
Stanford University
Stanford, CA
kjulian3@stanford.edu

Alireza Nakhaei
Honda Research Institute
Mountain View, CA
anakhaei@honda-ri.com

Kikuo Fujimura
Honda Research Institute
Mountain View, CA
kfujimura@honda-ri.com

Mykel J. Kochenderfer
Stanford University
Stanford, CA
mykel@stanford.edu

ABSTRACT

Decomposition methods have been proposed in the past to approximate solutions to large sequential decision making problems. In contexts where an agent interacts with multiple entities, utility decomposition can be used where each individual entity is considered independently. The individual utility functions are then combined in real time to solve the global problem. Although these techniques can perform well empirically, they sacrifice optimality. This paper proposes an approach inspired from multi-fidelity optimization to learn a correction term with a neural network representation. Learning this correction can significantly improve performance. We demonstrate this approach on a pedestrian avoidance problem for autonomous driving. By leveraging strategies to avoid a single pedestrian, the decomposition method can scale to avoid multiple pedestrians. We verify empirically that the proposed correction method leads to a significant improvement over the decomposition method alone and outperforms a policy trained on the full scale problem without utility decomposition.

KEYWORDS

Deep Reinforcement Learning; Utility Decomposition; Multi-fidelity Optimization; Autonomous Driving

ACM Reference Format:
Maxime Bouton, Kyle Julian, Alireza Nakhaei, Kikuo Fujimura, and Mykel J. Kochenderfer. 2018. Utility Decomposition with Deep Corrections for Scalable Planning under Uncertainty. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 8 pages.

1 INTRODUCTION

It is difficult to handcraft decision making strategies for autonomous systems in complex environments. The burden is placed on the designer to anticipate the wide variety of possible situations and explicitly program how the autonomous agent should behave. There has been growing interest in automatically deriving suitable behavior by optimizing decision policies using reinforcement learning (RL) [9].

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden. © 2018 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

There are many domains where the agent must interact with multiple entities. For example, in the autonomous driving context, these other entities may include other vehicles on the road. RL can have difficulty scaling to such domains because the state space grows exponentially with the number of entities. One way to approximate solutions to problems with multiple entities is to decompose the state space into pairwise interactions and to learn the utility function with each entity individually. The solutions to the individual subproblems can be combined in real-time through an arbitrator that maximizes the expected utility [13]. Applications to aircraft collision avoidance with multiple intruders explores summing or using the minimum state-action values [4, 11]. While these approaches tend to perform well, they are still suboptimal because the solution to each subproblem assumes that its policy will be followed rather than the best policy considering all subproblems [14].

Another approach to scaling reinforcement learning algorithms is to leverage the representational power of deep neural networks in modeling the utility function. Although this approach can handle domains such as Atari games and complicated manipulation tasks, it often requires millions of training examples and requires long training times [8, 10]. To minimize training time, transfer learning can take advantage of prior knowledge. For example, an existing controller or a human expert can gather high reward demonstrations to train an autonomous agent on only these examples [17]. Another transfer learning approach uses a regularizing term to guide the agent towards regions where it has prior knowledge of the appropriate behavior [7]. Multi-fidelity optimization has been combined with model-based reinforcement learning [5], where a learning framework is proposed for efficiently deciding between sampling from an inexpensive low-fidelity simulator and a real world data.

This paper presents an approach for scaling RL algorithms to problems with multiple entities using utility decomposition with corrections represented by a deep neural network. Borrowing concepts from multi-fidelity optimization, we learn a correction model to improve an existing pairwise approximation of the optimal utility function. We first present how such approximate solutions can be easily found by decomposing sequential decision problems into individual subproblems. The value functions to the subproblems can be combined using utility fusion to approximate the utility function of the global task. We then introduce the concept of surrogate corrections with neural representation, originally from multi-fidelity

optimization, to improve this globally suboptimal policy. Finally, we empirically demonstrate the advantage of this technique on an autonomous driving scenario. It is shown that the policy resulting from the correction method is faster to learn and also gives better performance than a policy learned directly.

2 BACKGROUND

In this section we introduce a common mathematical framework for formulating sequential decision making problems as an optimization of a reward over time. A utility function is defined to represent the expected reward that th agent can accumulate from a given state while acting optimally. For large problems, the utility function can be approximated by a combination of utility functions of simpler sub-problems.

2.1 Markov Decision Processes

Sequential decision problems are commonly formulated as Markov decision processes (MDPs). An MDP can be formally defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ where $\mathcal{S}$ is a state space, $\mathcal{A}$ is an action space, T a state transition function, R a reward function, and γ a discount factor. At time t, an agent chooses an action $a_t \in \mathcal{A}$ based on observing state $s_t \in \mathcal{S}$. The agent then receives a reward $r_t = R(s_t, a_t)$. At time $t+1$, the environment transitions from s_t to a state s_{t+1} with a probability $\Pr(s_{t+1} \mid s_t, a_t) = T(s_{t+1}, s_t, a_t)$. The agent's objective is to maximize the accumulated expected discounted reward given by $\sum_{t=0}^{\infty} \gamma^t r_t$.

A policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ defines what action to execute at a given state. Each policy can be associated to a state-action value function $Q^\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, representing the expected discounted value of following the policy π. The optimal state value function of an MDP 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 of the environment and s' a next state reachable by taking action a. This paper focuses on approaches where the transition function is not directly available. Instead the agent has access to a generative model from which the next state is sampled. The Bellman equation can be defined more generally as an expectation over the next state:

$$Q^*(s,a) = \mathbb{E}_{s'}[R(s,a) + \gamma \max_{a'} Q^*(s', a')] \tag{2}$$

Once Q^* is computed, the associated optimal policy is given by $\pi^*(s) = \arg\max_a Q^*(s,a)$. Similarly, we define the utility of a given state as follows:

$$U^*(s) = \max_a Q^*(s,a) \tag{3}$$

In this paper we are interested in a single agent planning problem against multiple non-cooperative entities.

2.2 Utility Decomposition

Utility decomposition, which is sometimes called Q-decomposition [14], involves combining the utility functions associated with simple decision making tasks to approximate the solution to a more complex task. Each subtask i is formulated as an MDP and is first solved in isolation. The function Q_i^* represents the optimal value function to solve the subtask i. In non-cooperative multi-agent settings, these subtasks are pairwise interactions. Solving the global task is then achieved by fusing the utilities associated with each subtasks. More formally, it requires defining a function f such that:

$$Q^*(s,a) \approx f(Q_1^*(s_1, a), \ldots, Q_n^*(s_n, a)) \tag{4}$$

Q^* is the optimal value function associated with the global task. The state variable can also be decomposed and we can assume that each of the value functions Q_i^* uses subset of the information contained in s to solve the simpler subtask. One approach to this decomposition, presented by Russel [14], involves decomposing the reward function additively. Each term of the sum is then optimized individually by a sub-agent. To solve the global problem they then choose f to be the sum of the individual value functions. They show that using the tabular Q-learning algorithm the solution does not achieve optimality, although decomposition of the reward model has been demonstrated to work well empirically [13].

The simplicity of the approach makes it a very appealing technique to find an approximately optimal solution to a complex decision making problem. In many problems involving non-cooperative multiple agents, utility fusion can help to scale the solution. A common setting for utility decomposition occurs when an autonomous agent must make decisions involving multiple entities. One example is collision avoidance problems [4], where the autonomous agent must avoid multiple moving targets. In this setting, the decomposition approach consists in solving the task of avoiding a single target. When interacting with multiple entities, the agent computes the value associated with avoiding each entity i assuming it is the only entity to avoid. Solving for pairwise interactions rather than the global problem requires exponentially less computations since the size of the state space often grows exponentially with the number of entities to consider in the planning. The global utility is then computed by summing the individual value functions or by taking the minimum over each entity to avoid. Summing the value functions, as in Equation (5), implies that all targets to avoid are independent.

$$Q^*(s,a) \approx \sum_i Q_i^*(s_i, a) \tag{5}$$

It equally weighs the utility of a user in a safe zone of the environment and a user in a more dangerous zone. Instead, another strategy is to consider to take the minimum as follows:

$$Q^*(s,a) \approx \min_i Q_i^*(s_i, a) \tag{6}$$

In Equation (6), taking the minimum will consider the target with the lowest utility. Given a reward function penalizing for collisions, it will consider the target that action a is most likely to harm. This approach is more risk averse. A detailed example of the decomposition method for a collision avoidance problem for autonomous driving is discussed in Section 4. Equation (5) and Equation (6) will be referred to as the max-sum and max-min approaches respectively since the action to execute is obtained by $\arg\max_a f(Q_1^*(s_1, a), \cdots, Q_n^*(s_n, a))$.

A disadvantage of this approach is that it requires choosing a fusion function that will play the role of arbitrator between the different utilities considered. The choice of this function can greatly affect the performance of the global policy and has no guarantee of optimality. Instead it provides a low-fidelity approximation of

the optimal value function for little computational cost once the individual utility functions are computed.

3 DEEP CORRECTIONS

A method that leverages the decomposed utility functions to guide an RL algorithm to reach an optimal policy with less experience is introduced here.

3.1 Deep Q-Networks

In MDPs with discrete state and action spaces, the value function can often be represented by a table and computed using dynamic programming. In many reinforcement learning problems, the state space can be very large or even continuous, making it impossible to explicitly represent the value associated to every possible state. Instead, the value function is represented by a parametric model such as a neural network: $Q(s, a; \theta)$, referred to as a deep Q-network (DQN). This technique approximates the value of every possible state with a limited number of parameters θ.

Computing the parameters of the network can be formulated as a learning problem. An objective function to minimize, based on Equation (2), can be expressed as follows:

$$J(\theta) = \mathbb{E}_{s'}[(r + \gamma \max_{a'} Q(s', a'; \theta_-) - Q(s, a; \theta))^2] \quad (7)$$

The parameter θ_- defines a fixed target network. The loss function is computed and minimized using sampled experiences. An experience comes from interacting with the environment over one time step where the agent in state s takes action a, transitions to state s', and receives reward r. The action a is selected using a ϵ-greedy strategy. The original Q-learning algorithm uses a single Q-network in the objective function but fixing the parameters in a target network for several steps has been shown to help the convergence of the algorithm [10]. If Q satisfies the Bellman equation, then the loss should be zero. By taking the gradient of J with respect to the parameters θ, we obtain, given an experience (s, a, r, s'), the following update rule:

$$\theta \leftarrow \theta + \alpha(r + \gamma \max_{a'} Q(s', a'; \theta_-) - Q(s, a; \theta))\nabla_\theta Q(s, a; \theta) \quad (8)$$

where α is the learning rate, a hyperparameter of the algorithm.

This algorithm often requires many experience samples to converge and can be unstable and difficult to tune. Fortunately, there are several innovations to improve network training, such as double DQN, dueling network architectures, and prioritized experience replay [15, 19, 20]. These are the three improvements to the deep Q-learning algorithm that we used in this paper.

Although this algorithm can handle high dimensional state spaces, it requires a lot of experience samples to converge (on the order of millions for Atari games [10, 20]). Experience can sometimes be generated using simulators but for applications where simulation is expensive or inexistent, the deep Q-learning algorithm can become impractical. When applying reinforcement learning to accomplish a difficult task, a possibility is to decompose it into simpler subtasks for which you can afford to run algorithms such as deep Q-learning.

3.2 Policy Correction

Computing the optimal Q function with the deep Q-learning algorithm requires many experience samples mostly because the agent must go through a random exploration phase. By introducing prior knowledge into the algorithm, the amount of exploration required to converge to a good policy is drastically reduced. In this work, we consider that prior knowledge takes the form of a value function. No specific representation is needed. The decomposition method described above would be a simple and efficient way to generate prior knowledge. There are many techniques to approximate the value function, such as discretizing a continuous state space or using domain expertise such as physics-based models. If a controller already exists, a value function can be estimated using the Monte Carlo evaluation algorithm [18]. Given an approximately optimal value function, the objective is to leverage this prior knowledge to learn the optimal value function with as little computation as possible.

The method we propose is inspired by multi-fidelity optimization. Consider a setting with two models of different fidelities: an expensive high-fidelity model (f_{hi}) and a low-fidelity model (f_{lo}) providing an inexpensive but less accurate approximation. In multi-fidelity optimization, the goal is to maximize an objective function f approximated by these two simulators. However, only using the high-fidelity model would be too expensive since many queries are probably needed to optimize f. On the other hand, relying only on the low-fidelity model might lead to a poor solution. Instead, we could construct a parametric model that approximates f_{hi} but is inexpensive to compute. Such a surrogate model can represent the difference between the high-fidelity and the low-fidelity models [6, 12]:

$$f_{\text{hi}}(x) \approx f_{\text{lo}}(x) + \delta(x) \quad (9)$$

where δ is a surrogate correction learned using a limited number of samples from f_{hi}. Commonly used models to represent δ are Gaussian processes or parametric models. In this work we focused on additive surrogate correction, but a multiplicative correction or a weighted combination of both would be a valid approach as well [6].

Reinforcement learning can be reshaped into a multi-fidelity optimization problem. Let Q_{lo} be our low-fidelity approximation of the value function obtained from prior knowledge or from a decomposition method. The high-fidelity model that we want to obtain is the optimal value function. The surrogate correction for reinforcement learning can be described as follows:

$$Q^*(s, a) \approx Q_{\text{lo}}(s, a) + \delta(s, a; \theta) \quad (10)$$

In regular multi-fidelity optimization problems, samples from $f_{\text{hi}}(x)$ are used to fit the correction model. In reinforcement learning, $Q^*(s, a)$ is unknown, so a temporal difference approach is used to derive a learning rule similar to the Q-learning algorithm. We seek to minimize the same loss with the target network as presented in Section 3.1. The main difference lies in the representation of the Q function, which can be substituted by Equation (10). Only the corrective part is parameterized. As a consequence, when taking the gradients with respect to the parameters, the update rule becomes

$$\theta \leftarrow \theta - \alpha[R(s, a) + \gamma \max_{a'}(Q_{\text{lo}}(s', a') + \delta(s, a; \theta_-)) - Q_{\text{lo}}(s, a) - \delta(s, a; \theta)]\nabla_\theta \delta(s, a, \theta) \quad (11)$$

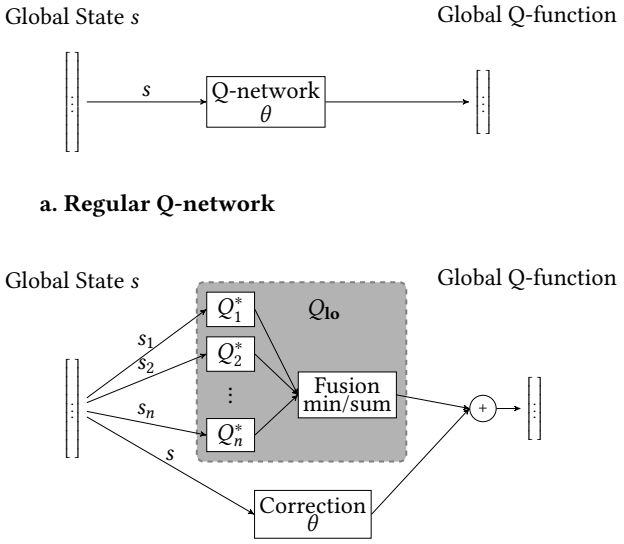
Instead of learning a Q-value representing the full problem, an additive correction is represented as a neural network. By using this representation of the corrective term, all the innovations (dueling, prioritized replay, double Q-network) used to improve the DQN algorithm can also be used to learn the correction. Although the algorithms are similar, learning to represent the correction to an existing value function should be much easier than learning the full value function.

An illustration of the deep correction architecture along with the decomposition method is presented in Figure 1. In order to use this approach jointly with a decomposition method we can substitute Q_{lo} by the results from utility fusion. For example substituting eq. (5) in eq. (10) would lead to the following approximation of the global Q function:

$$Q^*(s,a) \approx \sum_i Q_i^*(s_i,a) + \delta(s,a;\theta) \tag{12}$$

The value functions Q_i^* come from the subproblem solutions. In the collision avoidance setting, the value functions are the same, although other settings could use distinct value functions to represent Q_i^*. We can see from the update rule with correction that these functions are constant with respect to θ. In practice, if a neural representation is used for the subproblem solutions, then the weights of the subproblem networks are frozen during the training of the correction term.

The deep Q-learning algorithm used to learn the correction term does not have any theoretical convergence guarantees. However we demonstrate in section 4 that despite of the lack of guarantees, the deep Q-learning approach still learns good policies. Intuitively, starting with the policy from utility fusion significantly reduces the amount of exploration in DQN. If the low-fidelity value function is a good approximation of the optimal policy, the update rule leads the corrective term towards zero. This formulation is also flexible. A regularization term could be added to the loss function to minimize the impact of the corrective factor if one has trust in the low-fidelity policy.



a. Regular Q-network

b. Decomposition and Deep Correction

Figure 1: Regular DQN architecture (a) and architecture of the deep correction approach used with the decomposition method (b). The global state is decomposed and each substate is fed into networks pre-trained on the single entity problem. The output of the correction network is then added to the output of the utility fusion to approximate the global Q-function. More generally, the shaded gray area could be replaced by any existing value function Q_{lo}.

4 APPLICATION

This section is dedicated to applying the deep correction approach in addition to the decomposition method. It provides empirical results that demonstrates the advantage of learning the correction term.

4.1 Occluded Crosswalk

To illustrate the policy correction technique, we chose to focus on an autonomous driving scenario involving a crosswalk occluded by a physical obstacle as presented in Figure 2. The objective of the agent (the ego car), is to navigate through the crosswalk safely and efficiently. It must avoid potential pedestrians crossing the road. The agent must anticipate uncertainty and trade off between gathering information and reaching the goal position.

This problem can be modeled as a partially observable Markov decision process (POMDP). Contrary to an MDP, the state is not fully observable. In this crosswalk scenario, partial observability comes from the presence of a physical obstacle occluding the field of view of the sensors. Previous works propose modeling tactical decision making for autonomous driving as a POMDP [1–3]. One of the biggest challenges is to find a representation that makes the problem tractable. Bandyopadhyay suggests a discrete formulation to address a pedestrian collision avoidance problem [2]. Although the method provides an efficient policy, it is unlikely to scale in avoiding multiple pedestrians since the size of the state space would grow exponentially. Other approaches solve the problem in the continuous space and rely on sampling based methods, but suffer from the curse of dimensionality when trying to handle multiple road users [1, 3]. In this work, we suggest a similar formulation of the problem in the continuous domain. To scale the problem efficiently, we use the decomposition method presented in Section 2.2. We learn a corrective term to refine the approximation and improve performance.

The state of the environment consists of the position and velocity of the ego car as well as the position and velocity of the pedestrians. The autonomous vehicle measures its own position and velocity, the position and velocity of the pedestrians that are not occluded, and a constant value s_{absent} for the pedestrians that are not visible. Observing s_{absent} can mean either that a pedestrian is in the occluded area or that it is simply absent from the environment.

The action space consists in a set of strategic maneuvers such as hard breaking, moderate braking, keeping a constant speed, accelerating represented by a finite set of acceleration inputs:

$$\{-4\,\text{m/s}^2, -2\,\text{m/s}^2, 0\,\text{m/s}^2, 2\,\text{m/s}^2\}.$$

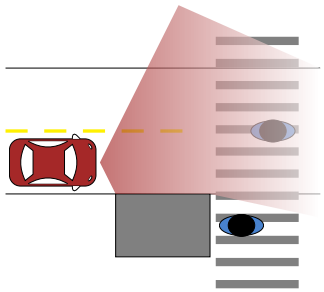


Figure 2: The autonomous vehicle in red must choose the acceleration to apply to navigate safely through the crosswalk. A gray obstacle occludes the vehicle's field of view. Several pedestrians might be present for the ego vehicle to avoid.

A transition model is defined that uses a point mass model for the ego vehicle and a constant speed model with random noise for the pedestrian. The pedestrian follows a desired speed of 1 m/s speed that can vary by a random amount in the set: $\{-1\,\text{m/s}, 0\,\text{m/s}, 1\,\text{m/s}\}$ at each time step.

The agent receives a reward when reaching a terminal state. Terminal states are defined by three possible outcomes:

- A collision for which the agent receives a penalty of −1.
- A time out, if the agent fails to pass the crosswalk under 20 s, for which the reward is 0.
- A success, if the agent reaches a goal position about 6 m after the crosswalk. A reward of +1 is received.

Time efficiency is not explicitly present in the reward function. Instead, a discount factor of 0.99 will encourage the vehicle to gather the final reward as fast as possible, since the value of the reward decays exponentially with time. Given the structure of the reward function, two evaluation metrics are defined. The first one is the collision rate across many simulations, and the second one is the average time to pass the crosswalk. These metrics account for safety and efficiency respectively. There is a natural trade-off between these two objectives, which makes this autonomous driving scenario a multi-objective optimization problem. The state variable as formulated consists of two dimensions for the ego vehicle and two dimensions for each pedestrians considered. As a consequence, the dimensionality grows exponentially with the number of agents considered. Many POMDP solvers would suffer from this representation [1–3]. Instead, the computational cost of the decomposition only increases linearly in the number of agents considered. One query of the individual utility functions per pedestrians considered is required to combine the utilities online.

We compare the performance of three different approaches to solving this occluded crosswalk scenario.

4.1.1 Deep Q-learning. The first approach uses deep Q-learning to solve the full problem. This scenario is naturally continuous and can be handled by a neural network representation of the value function. We used the hyperparameters from Table 1. A challenge for the occluded crosswalk problem is that the number of pedestrians present in the scene is unknown. For the sake of simplicity, the number of pedestrians was capped to ten. As a result, there are twenty-two state dimensions, with two dimensions for the ego car position and velocity and two dimensions for each pedestrian position and velocity.

When a pedestrian is not observed, its position and velocity are set to the constant value s_{absent}. This implementation limitation could be improved by using recurrent neural network to handle the partial observability, but this is left as future work. To address partial observability in the neural representation we use a k-Markov approximation that consists in approximating the POMDP structure by an MDP where the state consists of the last k observations: $s_t = (o_{t-k}, o_{t-k+1}, \cdots, o_t)$. The Q-network is given a history of the last four states [10].

4.1.2 Utility Fusion. The occluded crosswalk problem originally requires avoiding multiple pedestrians. A good decomposition of the utility for this problem considers the utility of avoiding a single pedestrian, similar to avoiding a single intruder aircraft [4]. For each pedestrian i in the environment, the ego vehicle observes a state s_i consisting of its own position and velocity as well as the position and velocity of the pedestrian. The agent then computes the optimal state-action value $Q^*(s_i, a)$ assuming i is the only user to avoid. The state-action value measures the expected accumulated reward of taking action a and then following the optimal policy associated with user i. By using Equation (4), the agent can approximate the solution of the global value function. In this work we considered two functions to combine the value functions as presented in Equation (5) and Equation (6).

4.1.3 Policy Correction. Finally, we demonstrate the deep correction method. By using the two policies described above as a low-fidelity approximation of the optimal value function, an additive corrective term is learned. The correction function is represented by a neural network, which is optimized through the approach presented in Section 3.2.

4.2 Experiments

All proposed solutions methods involve training a Q-network: on the full problem, on the single pedestrian problem, and on the corrective term. In order to fairly compare the three methods, a fixed training budget of one million experience samples was used for all three methods. For the policy correction technique, part of the sample budget was used to train the decomposed policy. The sample budget is divided equally between training a Q-network on the single pedestrian problem and training the corrective term. A hyperparameter search was executed to select the neural network architecture and the target network update frequency. Other parameters were set to common values found in the literature [10, 16, 20].

During training, the agent interacts with the environment following an ϵ-greedy policy. The value of ϵ is scheduled to decrease during training, and we found that the amount of exploration greatly influences the final policy. For the three approaches (training from scratch on the full problem, training on the problem with only one pedestrian, and training the correction factor), a random search over the exploration schedule was conducted. The exploration fraction values represents the fraction of total training time used to linearly decay ϵ from 1.0 to its final value. Each training sample

Table 1: Deep Q-learning parameters

Hyperparameter	Value
Neural network architecture	5 fully connected layers of 32 nodes
Activation functions	Rectified linear units
Replay buffer size	400 k experience samples
Target network update frequency	5 k episodes
Discount factor	0.99
Optimizer	Adam
Learning rate	1×10^{-4}
Prioritized replay [16]	α =0.7, β =1×10^{-3}
Exploration fraction	search between 0.0 and 0.9
Final ϵ	search between 0.0 and 0.1

Table 2: Environment Parameters

	Value	
Parameter	Evaluation	Training
Position sensor standard deviation	0.5 m	0.5 m
Velocity sensor standard deviation	0.5 m/s	0.5 m/s
Decision frequency	0.5 s	0.5 s
Simulation time step	0.1 s	0.5 s
Pedestrian velocity maximum noise	0.5 m/s	1.0 m/s

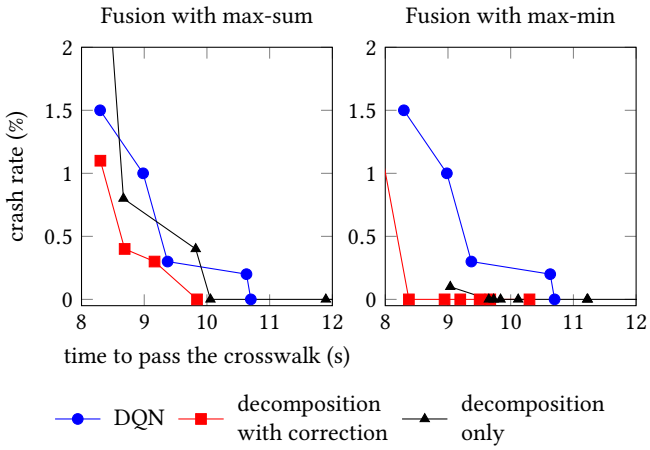


Figure 3: Pareto frontiers for the different approaches used to solve the crosswalk problem: DQN, utility decomposition, and utility decomposition with correction. Each point results from evaluating a policy generated by a given set of hyperparameters. The figure is zoomed in the region of optimality (bottom left). The two figures represents the max-sum (left) and the max-min (right) fusion techniques.

is initialized randomly, the ego vehicle starts at the same position with a velocity uniformly sampled between 6 m/s and 8 m/s. The initial number of pedestrians and their positions and velocities is also randomly sampled. From this initial state, the state is updated given the action of the agent and the dynamic model described in section 4.1.

All trained policies are evaluated in an evaluation environment that differs from the training environment. It has a finer time discretization, and the model followed by the pedestrian is less stochastic, as reported in Table 2. The weights of the networks are frozen during the evaluation. Each simulation is randomly initialized: random initial velocity for the ego vehicle, random number of pedestrians present with a random position and velocity. Each trained policy is evaluated on the two metrics of interest that are averaged over a thousand simulations: collision rate and time to pass the crosswalk. Since these two objectives are conflicting, Pareto optimality is used to decide if a policy is better than another. Formally, we say that a policy dominates another if it outperforms the other policy in one objective and is no worse in the other objective. From this definition, we can draw the Pareto frontier associated with all the generated policies from the hyperparameter search. The frontiers for the three approaches is represented in Figure 3.

We analyzed the benefit of the policy correction technique in terms of learning speed. Assuming that solving the single pedestrian problem does not require any training, we looked at the evolution of the performance of the corrected policy during training of the correction function. In our application, the single agent policy also requires training a deep Q-network. However this solution could come from an existing controller or from a model based planning approach that does not require sampling. In Figure 5, we froze the weights of the network at regular intervals during the training and evaluated the policy in the evaluation environment. The corresponding networks in Figure 3 are the fastest one guaranteeing safety (zero collisions over the thousand simulations). The metrics chosen for this experiments where the number of crashes, successes or time-outs which are the three possible outcomes of a simulation in the occluded crosswalk scenario. A simulation ends in a time-out if the ego vehicle is not able to cross in less than 20 s.

4.3 Results

The Pareto frontiers on Figure 3 show a domination of the policy correction method. If we keep one objective fixed, we can always find a policy computed with the correction method that outperforms the two other methods in the other objective. A reasonable approach would be to fix the safety level at 0 % of collisions over the thousand simulations and use the policy that minimizes the time to pass the crosswalk. Using the max-sum fusion, the fastest safe policy has an average time to cross of 10.06 s. By adding the correction term, the policy achieves an average time to cross of 9.84 s. Using max-min, the policy reaches an average time of 9.65 s, and 8.38 s with the correction. The deep Q-network policy has a time to cross of 10.70 s. For both utility fusion methods, max-sum and max-min, the addition of the corrective factor not only leads to an improvement in the policy but also outperforms the deep Q-network policy. The max-min decomposition method even dominates the Q-network trained on the complex environment. This result illustrates that the choice of a good function for combining the utilities can provide a good policy without training in the multi-pedestrian environment. However, Figure 3 shows that the max-sum decomposition without correction does not dominate the Q-network approach. In the scenario of interest, considering

Table 3: Best Hyperparameters

Model	Exploration Fraction	Final ϵ
DQN	0.50	0.01
max-min correction	0.00	0.01
max-sum correction	0.20	0.00

the agent with the minimum utilities favors risk averse behavior, resulting in safer policies for a given time to pass the crosswalk.

To gain intuition on the difference between these policies, we can visualize them on a two-dimensional slice of the state space in Figure 4. To generate this representation we ran simulations with the ego vehicle driving at a constant speed of 6.0 m/s (not reacting to the policy), and a single pedestrian fixed at a given position along the crosswalk. Although the policy is being tested against one pedestrian, it is still assuming that there might be multiple pedestrians. Figure 4 shows the actions returned by the policy at a given ego car position along the road and a given position of the pedestrian. The pedestrian is located at 25 m along the road and the ego car is moving along a horizontal line that intersects the crosswalk at 0 m. All plots show a red zone just on the left of the position (25, 0) representing a braking behavior when a pedestrian is in the middle of the road. The larger to the left the red area is, the more the car will anticipate the braking. Similarly, all the policies present a vertical red zone before 15 m along the road, indicating that the car will slow down regardless of pedestrian position. This behavior is natural since the presence of the obstacle hides a lot of information. A safe behavior is to slow down until the vehicle has more visibility. These plots can help us visualize areas of the state space where the policy might be suboptimal. For example, the two policies using the decomposition method without the correction present a braking area after the crosswalk although this area of the state space is safe, and nothing should prevent the car from accelerating. These suboptimal parts disappear or shrink when adding the correction term.

Finally, we looked at the evolution of the policies performance during training. Although they where trained with half the samples, the policies with the correction term converge. Since the policy resulting from the decomposition method already has performances close to the deep Q-network policy, only very little exploration is required to learn the corrective term. The max-min decomposition results in more stable training than the other policies, and the three metrics converge after about three hundred thousands episodes. The networks achieving these performances result from the hyperparameter search on the exploration schedule. The corresponding values are reported in Table 3. When training the corrective term, the amount of exploration to reach good performance is an order of magnitude lower than training DQN on the global problem.

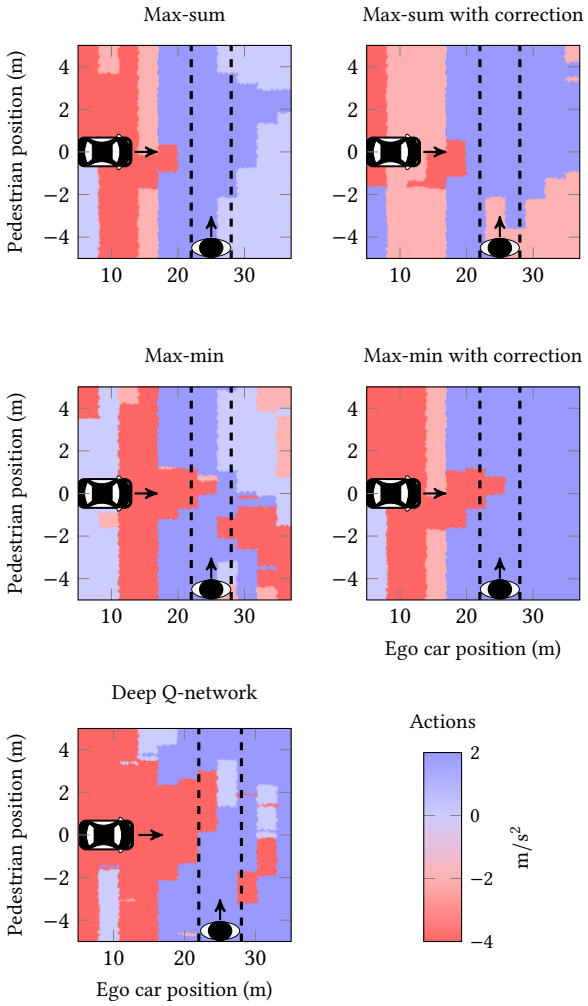


Figure 4: Visualization of the policies from the decomposition methods, the policy correction approach and using a deep Q-network. It assumes that the velocity of the car is 6 m/s and that the pedestrian is not moving. The color at a given point represents the action the ego car would take if it is at a certain position along the road and the pedestrian is at a certain position along the crosswalk.

5 CONCLUSIONS AND FUTURE WORK

Utility fusion methods efficiently find approximate solution to decision making problems where an agent interacts with multiple entities. Once a solution for interacting to a single entity has been computed, solutions can be combined to solve the global problem. Although computationally efficient, combining the utilities with simple functions, such as max-sum and max-min, leads to a suboptimal solution. To overcome this problem we presented a novel technique to gear an existing suboptimal policy towards the optimum by learning a correction term. The correction term is represented by a neural network and is learned using deep Q-learning. This method inspired from multi-fidelity optimization can significantly improve a policy computed with the decomposition method. We verified this statement empirically on an autonomous driving scenario involving an occluded crosswalk. Adding the corrective factor lead to a much more efficient policy than using the decomposition only. It also outperformed a deep Q-network policy trained on the full scale problem. We also demonstrated that learning the

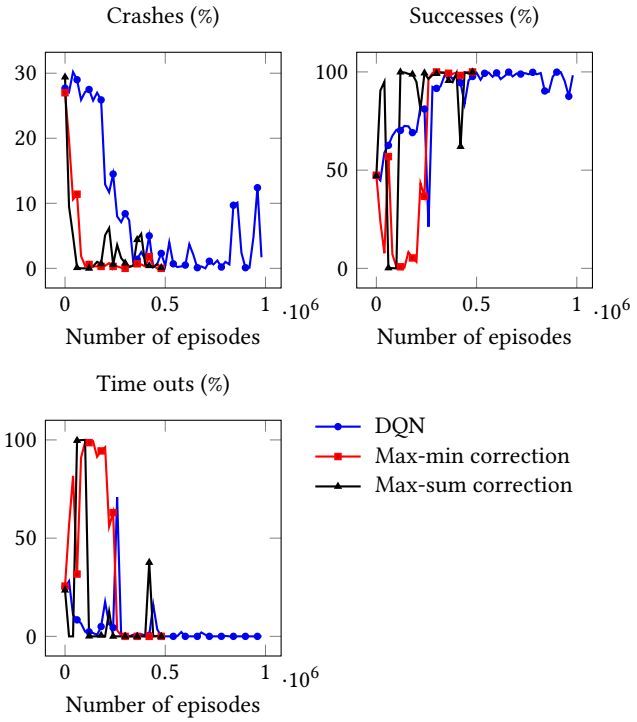


Figure 5: Evaluation of the policy performance throughout the training. The correction function is being trained on twice as less samples than the regular DQN policy but is still converging and outperforming the other for both choices of the decomposition method (max-sum or max-min).

corrective factor can be done with fewer training samples than directly learning the value function of the multi-entity setting.

In the future, more sophisticated multi-fidelity optimization techniques could be used to represent the correction term. Rather than an additive correction, we could try a multiplicative term [6]. Another possibility involves learning the function used for utility fusion itself. A straightforward extension would be to learn a linear weighted combination of the utilities from the single entity problem. Finally, we would like to explore the generality of the correction method. We wish to extend the use of decomposition methods to correcting policies coming from potentially different solving techniques such as an offline POMDP solver. Further experiments for different applications than the one presented in this paper could also highlight the benefit of learning a correction to an existing policy.

REFERENCES

[1] Haoyu Bai, David Hsu, and Wee Sun Lee. 2014. Integrated perception and planning in the continuous space: A POMDP approach. *International Journal of Robotics Research* 33, 9, 1288–1302.

[2] Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, and Daniela Rus. 2012. Intention-Aware Motion Planning. In *Algorithmic Foundations of Robotics X*. 475–491.

[3] Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. 2014. Probabilistic decision-making under uncertainty for autonomous driving using continuous POMDPs. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. 392–399.

[4] James P Chryssanthacopoulos and Mykel J Kochenderfer. 2012. Decomposition methods for optimized collision avoidance with multiple threats. *AIAA Journal of Guidance, Control, and Dynamics* 35, 2, 398–405.

[5] Mark Cutler, Thomas J Walsh, and Jonathan P How. 2015. Real-world reinforcement learning via multifidelity simulators. *IEEE Transactions on Robotics* 31, 3, 655–671.

[6] Michael Eldred and Daniel Dunlavy. 2006. Formulations for Surrogate-Based Optimization with Data Fit, Multifidelity, and Reduced-Order Models. *AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*.

[7] Martin Gottwald, Dominik Meyer, Hao Shen, and Klaus Diepold. 2017. Learning to walk with prior knowledge. In *IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. 1369–1374.

[8] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *IEEE International Conference on Robotics and Automation (ICRA)*. 3389–3396.

[9] Mykel J Kochenderfer. 2015. *Decision Making Under Uncertainty: Theory and Application*. MIT Press.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540, 529–533.

[11] Hao Yi Ong and Mykel J. Kochenderfer. 2015. Short-term conflict resolution for unmanned aircraft traffic management. In *Digital Avionics Systems Conference (DASC)*. 5A4–1–5A4–13.

[12] Dev Rajnarayan, Alex Haas, and Ilan Kroo. 2008. A Multifidelity Gradient-Free Optimization Method and Application to Aerodynamic Design. *AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*.

[13] Julio K Rosenblatt. 2000. Optimal selection of uncertain actions by maximizing expected utility. *Autonomous Robots* 9, 1, 17–25.

[14] Stuart J. Russell and Andrew Zimdars. 2003. Q-Decomposition for Reinforcement Learning Agents. In *International Conference on Machine Learning (ICML)*. 656–663.

[15] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized Experience Replay. In *International Conference on Learning Representations (ICLR)*.

[16] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized experience replay. *International Conference on Learning Representations (ICLR)*.

[17] William D. Smart and Leslie Pack Kaelbling. 2002. Effective reinforcement learning for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*. 3404–3410 vol.4.

[18] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning - an Introduction*. MIT Press.

[19] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning.. In *AAAI Conference on Artificial Intelligence (AAAI)*. 2094–2100.

[20] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. 2016. Dueling Network Architectures for Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML)*. 1995–2003.