Decentralized Cooperative Planning for Automated Vehicles with Hierarchical Monte Carlo Tree Search

Karl Kurzer*[1], Chenyang Zhou*[1] and J. Marius Zöllner[1,2]

***Abstract*— Today's automated vehicles lack the ability to cooperate implicitly with others. This work presents a Monte Carlo Tree Search (MCTS) based approach for decentralized cooperative planning using macro-actions for automated vehicles in heterogeneous environments. Based on cooperative modeling of other agents and Decoupled-UCT (a variant of MCTS), the algorithm evaluates the state-action-values of each agent in a cooperative and decentralized manner, explicitly modeling the interdependence of actions between traffic participants. Macro-actions allow for temporal extension over multiple time steps and increase the effective search depth requiring fewer iterations to plan over longer horizons. Without predefined policies for macro-actions, the algorithm simultaneously learns policies over and within macro-actions. The proposed method is evaluated under several conflict scenarios, showing that the algorithm can achieve effective cooperative planning with learned macro-actions in heterogeneous environments.**

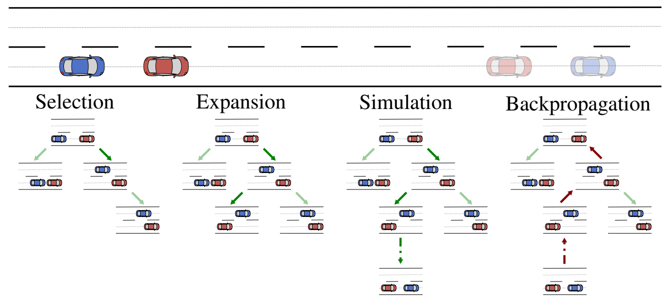


Fig. 1: Phases of Monte Carlo Tree Search for an overtaking maneuver; the selection phase descends the tree by selecting promising children until a node is encountered that has not yet been fully expanded. Upon expansion random actions are simulated until the planning horizon is reached. The result is backpropagated through all nodes along the chosen path. Eventually the algorithm converges to an optimal action sequence.

I. Introduction

While the quality of automated driving is progressing at a staggering pace, today's automated vehicles are lacking a key ingredient that heavily separates them from their human counterparts — implicit cooperation. In contrast to the traditional egoistic maneuver planning methods for automated vehicles, human drivers take other drivers' subtle actions into consideration enabling them to make cooperative decisions even without explicit communication.

Thus, in recent years a variety of cooperative planning approaches for vehicles have been proposed that take the interdependence of ones own action and the actions of others into account. Methods for solving this challenging task are frequently based on research in the area of game theory [1], [2] and multi-agent systems, where cooperative automated driving can be viewed as a cooperative game with multiple players, imperfect information and simultaneous moves. Besides, when ignoring any execution uncertainty, transitions from the current to the next state are fully dependent on the joint actions of all players, the problem can thus be treated as a multi-agent Markov Decision Process (MDP). Algorithms designed for single-agent systems often suffer from the *curse of dimensionality* when applying them to multi-agent systems, which means that the number of possible outcomes increases exponentially as the number of agents grows. This is even more severe when planning for longer time horizons.

Monte Carlo Tree Search (MCTS), a reinforcement learning method [3], has shown promising results on multiple occasions facing problems of this kind. The most popular example is the software AlphaGo, reaching super-human performance in the game of Go [4], [5]. MCTS repeatedly samples a model to improve value estimates of actions at a given state using backpropagation. These updates guide the selection and expansion phases towards more promising areas of the search space. An example for the domain of automated driving is given in Fig. 1. A thorough overview of MCTS and its extensions is presented in [6]. Since it was shown that the performance of MCTS is dominated by its effective search depth [7], and multi-agent problems have an inherently large branching factor, temporal abstraction is used in this work by extending actions over several time steps, hereafter macro-actions (MAs). MAs address the curse of dimensionality by generally reducing the problem complexity, leading to quicker convergence [8].

DeCoH-MCTS generates **de**centralized **co**operative **h**ierachical plans and applies it to the domain of automated driving. First, we model the problem of decentralized simultaneous decision making as a matrix game and solve it with Decoupled-UCT (a variant of MCTS), removing dependencies on the decisions of others. Additionally, to achieve longer planning horizons we integrate temporally extended macro-actions (MAs) in Decoupled-UCT. These MAs are designed in a flexible way that requires only initial and terminal conditions to be defined, allowing the algorithm to simultaneously learn which MA to choose and how to execute it. Last we evaluate the capabilities of DeCoH-MCTS in simulation, showing that it can achieve effective cooperative planning with learned macro-actions in heterogeneous environments. In addition the comparison with flat MCTS indicates that our algorithm can generate feasible plans in complex traffic scenarios with fewer

[1]FZI Research Center for Information Technology, Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany {czhou, zoellner}@fzi.de
[2]Karlsruhe Institute of Technology, Kaiserstr. 12, 76131 Karlsruhe, Germany {kurzer}@kit.edu — ***These authors contributed equally.**

Preprint submitted to 2018 IEEE Intelligent Vehicles Symposium (IV).
Received February 14, 2018.

iterations but higher quality.

The remainder of this paper is structured as follows: Section II gives a brief overview of research on cooperative automated driving, as well as reinforcement learning in multi-agent systems. The problem is formally defined and its terminology is introduced in section III. Adaptations to flat MCTS are presented in section IV and the resulting DeCoH-MCTS algorithm is described in detail in Section V. Lastly, DeCoH-MCTS is evaluated in a variety of scenarios.

II. Related Work

A. Cooperative Driving

Instead of assuming traffic participants follow merely their own agenda, cooperative planning considers others' anticipations and reactions to the ego vehicle's behavior and chooses actions that are cooperative, increasing the total utility [9], [10]. The first successful demonstration of cooperative automated vehicles emerged from the California PATH program [11] in the 1990s, where the notion of string stability was introduced to maintain the stability of a group of automated vehicles. Later, other projects focusing on the potential of cooperative perception and motion planning as well as the required software and hardware structures were conducted [12].

A definition of cooperative driving behavior, its necessary preconditions and an algorithm to generate cooperative plans is presented in [10], [13]. Using a utility focused approach cooperative driving behavior is achieved by increasing the overall utility consisting of every agent's own utility. Assuming that utilities of all agents can be perfectly estimated, the presented algorithm finds the combination of actions with maximum utility through an exhaustive search. Actions of agents are represented by quintic polynomials optimized for safety, energy, time, and comfort.

The potential of MCTS for cooperative driving is first presented in [14]. Based on Information-Set MCTS presented in [15], [16] they ensure decoupled decision making and conduct decentralized planning. Similar to [10], [13] they define a set of high-level actions resembled by motion primitives with action duration of one second. The algorithm is demonstrated in three different merge scenarios, with up to three vehicles directly interacting with the ego vehicle, while the others are merely guided by an Intelligent Driver Model [14]. Additionally, the number of lane changes within the planning horizon is restricted to one.

B. Hierarchical Reinforcement Learning

To address the problem of combinatorial explosion when planning for longer horizons, temporally extendable actions have long been studied in the domain of reinforcement learning. Sutton et al. [17] provided a comprehensive framework incorporating temporal abstraction into reinforcement learning. While they pointed out that there will be a loss of optimality due to the fixed internal structure of the options, they presented intra-option learning methods [17], [18] to achieve more flexible options.

Another framework for hierarchical reinforcement learning was presented in [19], where the entire task is decomposed hierarchically and then solved by dealing with multiple smaller tasks. Policies of all elements in the hierarchy can be learned simultaneously. The state-action-value function can be recursively decomposed into combinations of the state-action-value of primitive actions. This algorithm is proved to converge to the recursive optimality.

The combination of MCTS and macro-actions can be done in two ways. Either pre-defined/offline learned MAs represent the action space of the MCTS or online MA-learning is conducted within the search process. A trivial definition of MAs is the repetition of actions, which can deliver good results, but cannot be generalized [20], [21]. [22] creates more complex MAs including domain knowledge and additional algorithms for guided exploration within the macro-actions. MAs can also be learned offline by a DQN [23]. The resulting MAs are more flexible to an extent than the traditional pre-defined MAs, but are still limited. Additional Monte Carlo based planning methods using MAs are presented in [24], [25], which adopt the MAX-Q framework and propose a hierarchical MCTS algorithm, where each MA is learned by a nested MCTS in the larger search tree.

III. Problem Statement

We formulate the problem of cooperative planning with MAs as a decentralized Semi Markov Decision Process (Dec-SMDP). At each time step, all agents choose an action simultaneously without knowledge of future actions of others, receive an immediate reward and transfer the system to the consecutive state. The reward and the state transition is dependent on all agents' actions.

Formally, a Dec-SMDP is described by a tuple $\langle \Upsilon, \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, where

- Υ is the finite set of *agents* indexed by $i \in 1, 2, \ldots n$.
- $\mathcal{S}^i$ is the finite *state space* of an agent, $\mathcal{S} = \times \mathcal{S}^i$ represents the joint state space of Υ.
- $\mathcal{A}^i$ is the finite *action space* of an agent, $\mathcal{A} = \times \mathcal{A}^i$ represents the joint action space of Υ.
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the *transition function* $P(s'|s, \boldsymbol{a})$ which specifies the probability of the transition from state s to state s' under the joint action $\boldsymbol{a}$ defined by each agent's choice.
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function with $r(s, s', \boldsymbol{a})$ representing the reward after the joint action $\boldsymbol{a}$ is executed.
- $\gamma \in [0, 1]$ is a *discount factor* which controls the influence of future rewards on the current state.

We use superscript ^{i} to denote that a parameter is related to agent i. The solution to the Dec-SMDP is the joint policy $\Pi = \langle \pi^1, \ldots, \pi^n \rangle$, where π^i denotes the individual policy for a single agent, i.e., a mapping from the state to the probabilities of each available action, $\pi^i : \mathcal{S}^i \times \mathcal{A}^i \rightarrow [0, 1]$.

In an MDP, each agent tries to maximize the expected cumulative reward starting from its current state:

$G = \sum \gamma^t r(s, s', a)$ where t is the time and G is the return, representing the cumulated discounted reward. $V(s)$ is called the state-value function, given by $V^\pi(s) = E[G|s,\pi]$. Similarly, the state-action-value function $Q(s,a)$ is defined as $Q^\pi(s,a) = E[G|s,a]$, representing the expected return when choosing action a in state s.

The optimal policy starting at state s is defined as $\pi^* = \arg\max_\pi V^\pi(s)$. The state-value function is optimal under the optimal policy: $max\ V = V^{\pi^*}$, the same is true for the state-action-value function: $max\ Q = Q^{\pi^*}$. The optimal policy can be found by maximizing over $Q^*(s,a)$:

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a\in\mathcal{A}} Q^*(s,a) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Once Q^* has been determined, the optimal policies can easily be derived. Thus the goal is transformed to learning the optimal state-action-value function $Q^*(s,a)$ for each state-action combination (macro/primitive).

Compared to the single level of policies in an MDP, there exists a hierarchy of policies in an SMDP where each MA ω has its own policy π_ω for actions a. Additionally, there exists a policy π_μ for MAs, which decides how to choose the next MA ω' when the previous one terminates. The Bellman equations of the SMDP can be written as:

$$Q^\pi(s,\omega) = r^\omega + \sum_{s',\tau} \gamma^\tau p(s',\tau|s,\omega) \sum_{\omega'} \mu(\omega'|s') Q^\mu(s',\omega') \tag{2}$$

The former part of $Q^\mu(s,\omega)$ is the cumulated reward for this MA ω during its execution for τ steps:

$$r^\omega = \sum_{k=1}^{\tau} \gamma^{k-1} r_{t+k} \tag{3}$$

The latter part is the completion term C, which can be further decomposed until primitive actions are encountered [19]. Thus the state-action-value of choosing MA ω can be viewed as the cumulative discounted reward by following the policy of the chosen MA π_ω and then the policy π_μ which chooses this MA until π_μ ends.

IV. Approach

A. Hierarchical Action Graph

This section presents our design of macro-actions with a hierarchical graph for the cooperative driving domain, based on the *Option* [17] and *MAXQ* [19] frameworks.

While MAs reduce the complexity and thus the search space, the following are key challenges that must be addressed when implementing MAs.

1) Asynchronous decision making: In a multi-agent system with variable duration of MAs, MAs end asynchronously. Different strategies are presented in [26].

- t_{all} keeps some agents idle to wait for others finishing their MAs
- t_{any} simply interrupts all MAs when the first agent finishes its MA
- $t_{continue}$ allows asynchronous selection of MAs, which means that each agent independently decides its next macro-action once it terminates its current MA.

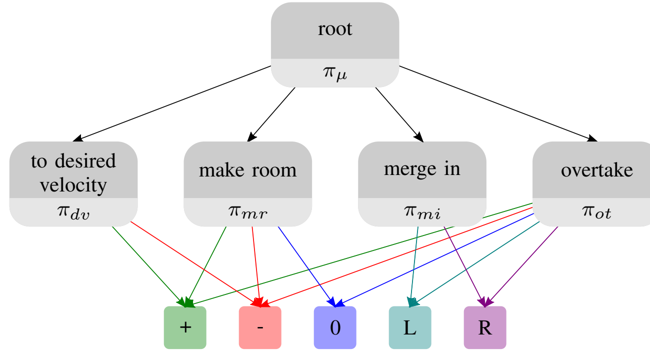


Fig. 2: The hierarchical action graph originates from an abstract root macro-action μ with a policy π_μ that selects other abstract macro-actions ω. These MAs have different action sets that are defined by primitive actions, that the agent can execute.

Clearly, the first two schemes t_{all} and t_{any} force a synchronization of the decision epochs and can only be realized in a centralized way, while $t_{continue}$ allows decentralized asynchronous decision making.

2) Flexible design of MAs: Naive pre-defined MAs are even more harmful than only planning with motion primitives [17]. To mitigate the risk, the policy inside a MA should be learned online and be flexible according to the current situation.

3) Cooperation Level: Learning of MAs can be distracted by lower level actions of other agents in a multi-agent system [27]. Consequently, approaches, such as localized macro-actions do not consider cooperation at the level of primitive actions is not [28], [29], [30]. However, this is unsuitable for cooperative automated driving, where consideration of primitive actions of all agents is required, e.g., for collision checking.

The *Option* framework [17] generalizes the primitive actions into temporally extended MAs with three components $\langle I, \pi, \beta\rangle$, where the MA is referred to as the option. I is the *initiation set* which specifies if this MA is available at the current state. $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$ is the above mentioned *policy* for this MA. $\beta : \mathcal{S} \to [0,1]$ is the *termination probability* that the MA terminates at the current state. Note that for primitive motion these three components are defined as $\pi(s,a) = 1$, $\beta(s) = 1$ and $I = S$.

The *MAXQ* framework [19] formulates the whole scenario as a root task and decomposes the root task into sub-tasks. To solve the root task, sub-tasks are sequentially chosen according to the root policy π_{root} and the sub-tasks are solved according to their own policies π_{sub}. The sub-tasks can be further decomposed into sub-sub-tasks until a primitive task (action) is encountered, the policies are decomposed accordingly.

We adopt the hierarchical action graph from the *MAXQ* framework. Considering the listed conflict scenarios that require cooperative driving in [31], we propose four MAs: *overtake*, *merge in*, *make room*, *to desired velocity*. Each MA has four components $\langle I, \pi, \mathcal{A}_\omega, \beta\rangle$, where $\mathcal{A}_\omega$ is the set of available actions at the immediate lower level. The primitive actions are defined as *acceleration*, *deceleration*, *do-nothing*,

TABLE I: Initial and Terminal Conditions for Macro-Actions

Macro-Action	Initial Condition	Terminal Condition
overtake	behind slower vehicle and left lane exists	ahead of slower vehicle
merge in	not in desired lane	in desired lane
make room	always possible	always possible
to desired velocity	not at desired velocity	at desired velocity

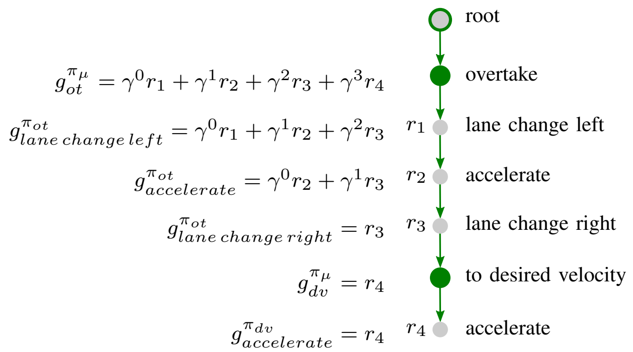


Fig. 3: Hierarchically bounded return for the example of a single agent; the returns of this iteration are defined on the left side. Note that the return for choosing MA *overtake* under the root policy π_μ includes all rewards from r_1 to r_4, since π_μ terminates at $t = 4$, while the return for choosing *lane change left* under policy π_{ot} only includes the reward from r_1 to r_3 because the MA *overtake* terminates at $t = 3$. This means that the return of an action a is bounded within its parent MA ω.

lane change left and *lane change right*. Each MA has a possible subset of these primitive actions and is referred as the parent action ω of the primitive actions, as depicted in Fig. 2. The solution to the general driving task is generalized as the *root* MA μ that entails all lower MAs. The initiation set I (or initial condition) and termination probability β (or termination condition) of all macro-actions are defined in Table I.

As defined by Eq. 2, the value of choosing action a at state s according to policy π is the cumulative discounted reward starting from the current state until π ends. An example for one iteration in the single agent domain with *hierarchically bounded return* is depicted in Fig. 3.

B. Decision Making without Communication

Since agents are not communicating, decentralized planning needs to be conducted, where each agent can only influence its own action, rather than the joint action of all agents. The agent's state-action-value estimation $Q(s, a)$ cannot distinguish among all joint actions containing this agent's action. Thus the agent needs to marginalize the state-action-value $Q(s, a)$ for all $a \in \mathcal{A}^i$. Problems of this kind were explored for simultaneous games [32] and the distributed reinforcement learning area [33]. The former uses Decoupled-UCT with matrix game settings which calculates the weighted average of all $Q(s, a)$ with the weight of each action being the visit count $N(s, a)$. The latter projects the larger Q-table with $Q(s, a)$-values into a smaller Q-table with marginalization, which is essentially the same as the former.

C. Cooperative Reward Function

As opposed to classical multi-agent systems with an explicit common goal and an immediate reward for the joint action, the common goal for cooperative driving is rather implicitly stated — solving a scenario with conflicting interests maximizing the overall reward, given each vehicle's safety, efficiency and comfort preferences. Similar to [10], [14] DeCoH-MCTS assumes identical reward functions for all agents. For each agent i a cooperative reward r^i_{coop} is calculated, which is the sum of its own reward r_i according to Eq. 4 as well as the rewards of all other agents based on Eq. 5.

$$\begin{aligned} r^i &= r^i_\phi + r^i_{action} \\ &= r^i_\phi + r^i_{safety} + r^i_{efficiency} + r^i_{comfort} \end{aligned} \tag{4}$$

$$r^i_{coop} = r^i + \lambda \sum_{j=0, j\neq i}^{n} r^{(j)} \tag{5}$$

r^i_ϕ is the shaping term described in the next section. $\lambda^i \in [0, 1]$ is a cooperation factor that determines the agent's willingness to cooperate with other agents (from $\lambda^i = 0$ *egoistic*, to $\lambda^i = 1$ *fully cooperative*). With the goal to generate cooperative maneuver decisions λ^i should be larger than 0.

D. Reward Shaping

Potential based reward shaping is used to accelerate the convergence of the learning process, while being optimality invariant [34]. Our work describes a desire that the agent strives to fulfill as a certain velocity and lane index at a given time. If the taken action brings the agent closer to its desired state, it will be rewarded positively.

A potential function $\phi(s)$ is defined to determine the potential of each state. The closer the current state to the desired state is, the higher the potential will be. In the MDP, the potential based reward for a transition from state s to s' by action a is written as:

$$r_\phi(s, s', a) = \gamma\phi(s') - \phi(s) \tag{6}$$

Thus, the ego reward function for each agent can be written as:

$$r^i_\phi = r^i_{action} + \gamma\phi(s') - \phi(s) \tag{7}$$

The potential shaping term can be generalized in the SMDP with an additional parameter τ denoting the duration of the MA, defined as:

$$r_\phi(s, s_{t+\tau}, \omega) = \gamma^\tau\phi(s_{t+\tau}) - \phi(s_t) \tag{8}$$

It can be proved that Eq. 8 is equivalent to the discounted sum of the shaped terms for each primitive action within its MA ω:

$$\gamma^\tau\phi(s_{t+\tau}) - \phi(s_t) = \sum_{k=1}^{\tau} \gamma^{k-1} r_\phi(s, s_{t+\tau}, \omega)^{a_{t+k}} \tag{9}$$

Preprint submitted to 2018 IEEE Intelligent Vehicles Symposium (IV).
Received February 14, 2018.

Algorithm 1 DeCoH-MCTS

```
function PLANNING(Υ, 𝒜, s)
    a ← ∅
    while driving do
        new root node n_μ ← n⟨a, Υ, 𝒜, s⟩
        a ← DeCoH-MCTS(n_μ)
        s ← ExecuteAction(a)
    end while
    return
end function

function DECOH-MCTS(n_μ)
    while computational budget not reached do
        ⟨n_leaf, ℛ⟩ ←TreePolicy(n_μ)
        ℛ ← SimulationPolicy(n_leaf, ℛ)
        BackpropagationPolicy(n_leaf, ℛ)
    end while
    return a ←FinalSelection(n_μ)
end function

function TREEPOLICY(n)
    repeat
        for i = 1 to |Υ| do
            a^i ← UCTAction(n, i)
            a ← [a, a^i]
        end for
        n ←SelectNode(n, a)
        ℛ ←CollectReward(n, a)
    until n == ∅
    ⟨n_leaf, ℛ⟩ ← ExpandNode(n, a)
    return n_leaf, ℛ
end function

function SIMULATIONPOLICY(n)
    for i = 1 to |Υ| do
        action a^i ← RandomSelection(n, i)
        a ← [a, a^i]
    end for
    ℛ ←CollectReward(n, a)
    return ℛ
end function

function BACKPROPAGATIONPOLICY(n, ℛ)
    while n ≠ n_μ do
        N(n) ← N(n) + 1
        for i = 1 to |Υ| do
            N(a^i) ← N(a^i) + 1
            G^i ← Σ^{a^i_p} γ^t r^i
            Q(a^i) ← Q(a^i) + (G^i − Q(a^i)) / N(a^i)
        end for
        n ← n_parent
    end while
end function
```

V. DECOH-MCTS WITH HIERARCHICALLY BOUNDED RETURN

We call our algorithm DeCoH-MCTS, its most important functions are outlined in Algorithm 1. It preserves the classical four steps of MCTS: selection, expansion, simulation and backpropagation. The function TREEPOLICY contains the selection and expansion steps. Like traditional MCTS the algorithm builds a search tree of possible future states, starting from the root node μ representing the initial state.

1) Tree Policy: UCT with a single agent, expands nodes until all available actions have been tried and then continues to grow the tree deeper. In the decentralized multi-agent system, the agent cannot distinguish between the joint actions. As a result, the tree can grow deeper once each agent has explored all of its available actions once.

We found that the node will only expand m child nodes regardless of the number of iterations if $\lambda^i = 1$.

At the first m iterations, each agent has tried all available actions and formed m joint actions. In each iteration, each agent's r^i_{coop} is the same as others' because $\lambda^i = 1$. Using the decentralization method, the state-action-value of each available action will be equal to the corresponding r^i_{coop}. At the $(m+1)$*-th* iteration, each agent chooses an action using the UCT value based on the previous calculated Q-estimates and visit counts, i.e., m for its parent and one for its children. As a result, the selected joint action must be the one with the largest r^i_{coop} among the explored m joint actions.

To address this problem, ϵ-Greedy is introduced and each agent selects an action with stochastic UCT as follows:

$$\pi^{\epsilon}(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|} & \text{if } a = arg\,max_{a \in \mathcal{A}} UCT(a) \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases} \tag{10}$$

As illustrated before, only primitive actions receive immediate rewards and can trigger system transitions. A joint action $\boldsymbol{a}$ is hence required to contain only primitive actions (not MAs) to be executed and transfer the system to a consecutive state. This implies, that all agents select according to their hierarchical policies until a primitive action is chosen. Our approach further adopts the $t_{continue}$ termination mechanism, dealing with MAs of variable duration, i.e., the decision making for the next MA is independent of the others' current MAs and thus asynchronous for all agents.

2) Simulation Policy: As no prior knowledge is used the simulation policy simply chooses MAs and their respective primitive actions at random.

3) Backpropagation Policy: Basic MCTS usually uses the simulation outcome without any intermediate rewards for actions in the backpropagation step [6]. By contrast, the return, i.e., cumulative discounted reward is used in our approach. As mentioned before, the return for the current action is bounded within its parent action. Both [24], [25] use the recursive form of MCTS based on the POMCP [35] to realizes the hierarchically bounded return, which is only applicable in the single-agent system and the multi-agent system with t_{any} or t_{all} termination rule.

TABLE II: Configuration for the Overtake Scenario

ID	color	x_0	v_0	l_0	v_{des}	l_{des}
0	blue	$5\,\mathrm{m}$	$15\,\mathrm{m/s}$	0	$25\,\mathrm{m/s}$	0
1	red	$25\,\mathrm{m}$	$15\,\mathrm{m/s}$	0	$20\,\mathrm{m/s}$	0
2	green	$45\,\mathrm{m}$	$15\,\mathrm{m/s}$	0	$15\,\mathrm{m/s}$	0

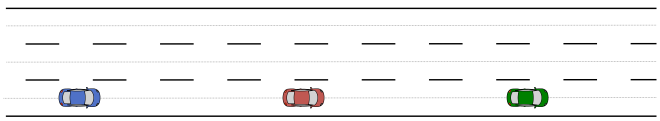


Fig. 4: Scenario: Overtake

To combine the hierarchically bounded return with $t_{continue}$ in a multi-agent system, the rewards along each iteration are stored in $\mathcal{R}$ together with the corresponding hierarchical information about the action. We then conduct a hierarchical boundary check based on this reward sequence, determining which MA the reward is associated with.

4) Final Selection and Execution: When the termination conditions are met, the agent has learned a policy hierarchy and chooses an action according to the max reward or max visits principle. It should be mentioned that the selected primitive action belongs to a certain macro-action. When starting a new search, this information can be incorporated in the new tree or discarded, which is called *hierarchical* control mode and *polling* control mode respectively. [17] showed that *polling*, which starts a new planning cycle without any memory about the previous step yields better results because polling allows the premature termination of macro-actions at each step and is thus more flexible.

VI. Evaluation

The evaluation is conducted using a simulation. We use three different scenarios to test if our algorithm can meet the following goals:

- Learning of MAs
- Converging quicker than flat MCTS
- Finding robust solutions when encountering non-cooperative drivers

Each scenario is defined by initial variable values indexed with $_0$, and desired values indexed by $_{des}$ denoting the agents desire. Where x denotes the position, v the velocity and l the lane index respectively. A video of the algorithm in execution can be found online .[1]

A. Learning of Macro-Actions

The overtake scenario is considered to test the algorithm's ability to simultaneously learn which MA to choose and how to execute it. The scenario is depicted in Fig. 4. Table II defines the settings for the scenario.

All three vehicles are controlled by their own DeCoH-MCTS with $\lambda^i = 1$.

The step length is set to $2\,\mathrm{s}$, a total of 2,000 iterations are executed with a maximum planing horizon of 20 steps. The resulting plan found at step 0 with the given scenario configuration is depicted in Table III

[1] http://url.fzi.de/DeCoH-MCTS-IV

TABLE III: Plan Result at Step 0

Agent	Planned Action Sequences
0	Overtake `L L + + 0 0 R R`
1	Overtake `L + + 0 R 0 0 0`
2	Make Room `+ 0 - 0 0 - + 0`

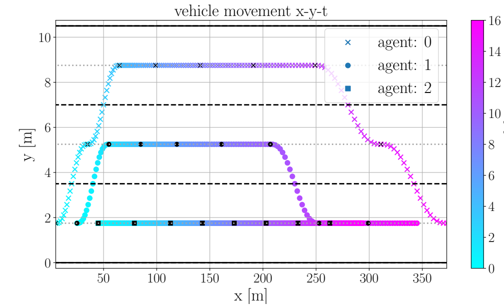


Fig. 5: Trajectories for each agent of the overtake scenario; The color represents the time according to the color bar on the right side. It can be seen that agent 1 changes to the left after driving in front of agent 2, while agent 0 stays in lane 2 until it gets in front of both two vehicles and then makes two lane changes to its desired lane 0.

It can be seen that agent 0 learns the MA *overtake* differently from agent 1, where agent 0 makes two lane changes to the left, accelerates to get in front of the other two vehicles and finally changes twice to its desired lane. Agent 2 shows cooperative behavior by making room for others.

In the polling control mode, each agent executes the planned action, i.e., L, L, + respectively, and then starts a new plan without memorizing the previous result. Fig. 5 shows the 2D trajectories for each agent.

B. Convergence

A test of the convergence speeds between DeCoH-MCTS and flat MCTS is conducted using the double merge scenario, visualized by Fig. 6.

There are two vehicles on a three-lane road with another four parked vehicles blocking the rightmost and leftmost lane. Both vehicle 0 (blue) and vehicle 1 (red) start with a speed of $25\,\mathrm{m/s}$ and want to keep their current lane and velocity.

We equip the macro-action overtake in one test with domain knowledge and implement it similar to the ϵ-greedy policy during the simulation, i.e., selecting an action given the domain knowledge with probability of $1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|}$, otherwise a random simulation is executed.

The algorithm runs in polling control mode with a step length of 2 seconds and terminates after 10 steps or reaching

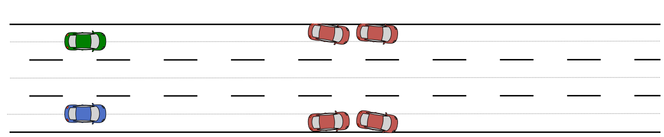
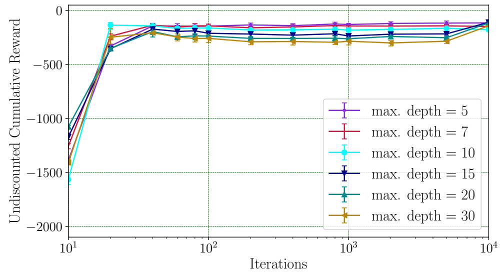
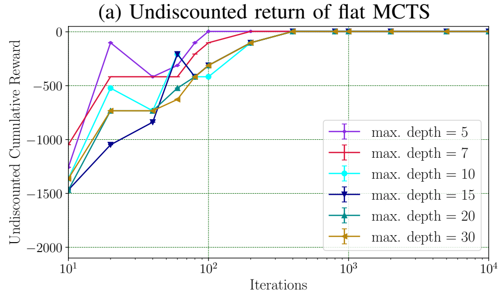


Fig. 6: Scenario: Double Merge

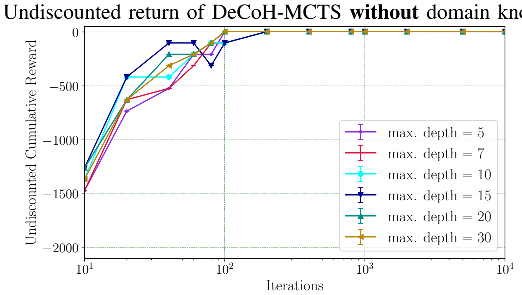
Preprint submitted to 2018 IEEE Intelligent Vehicles Symposium (IV).
Received February 14, 2018.



(a) Undiscounted return of flat MCTS



(b) Undiscounted return of DeCoH-MCTS **without** domain knowledge



(c) Undiscounted return of DeCoH-MCTS **with** domain knowledge

Fig. 7: Performance comparison of different MCTS versions in the double merge scenario

a terminal condition, e.g., a collision. The undiscounted cumulated rewards $r^i_{coop}$of vehicle 0 w.r.t. different numbers of iterations and maximal tree depths are calculated and compared, see Fig. 7. Each data point is the mean value based on 30 runs.

It can be seen that flat MCTS performs better in terms of undiscounted return for a low number of iterations, but DeCoH-MCTS clearly converges to a higher optimum as the number of iterations increases (>100). Considering that the number of iterations lies usually around 2000, the DeCoH-MCTS performs better than flat MCTS, as an increase in the maximal search depth leads to poorer performance for the classical MCTS as opposed to DeCoH-MCTS. The reason is that classical MCTS does not have any domain specific knowledge and conducts the simulation and expansion phase totally random. Larger maximum search depth means that the random expansions and simulations are more likely to end in collisions. As a result, almost all actions are perceived negatively. It becomes difficult for the agent to choose actions that fulfill its desire while not leading to a colliding state, so that the agent eventually chooses to decelerate to a standstill, being the safest option.

In addition, the comparison between DeCoH-MCTS with and without domain knowledge shows that the integration of domain knowledge can accelerate the learning speed while it does not affect the optimality of the solution.

TABLE IV: Configuration for the Bottleneck Scenario

ID	color	x_0	v_0	l_0	v_{des}	l_{des}
0	blue	5 m	10 m/s	0	15 m/s	0
1	red	195 m	5-17m/s	0	5-17m/s	1
2	green	100 m	0 m/s	0	0 m/s	0

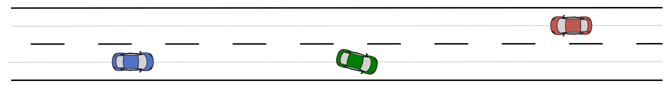
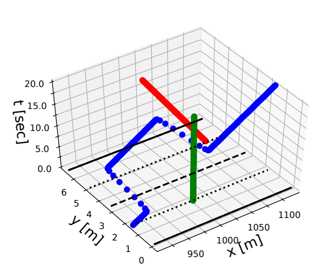


Fig. 8: Scenario: Bottleneck, where the green vehicle blocks one lane

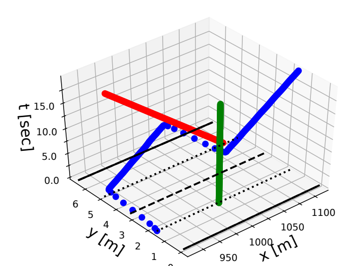
C. Robustness when encountering non-cooperative drivers

The bottle neck scenario is used to demonstrate the robustness in situations where other agents do not behave as the algorithm assumes (see IV-C). As Fig. 8 and Table IV show, vehicle 0 (blue) approaches from the left and is controlled by DeCoH-MCTS, vehicle 1 (red) drives from the right at different constant velocities in the range of $v \in [5\,\mathrm{m/s}, 17\,\mathrm{m/s}]$. Vehicle 1 keeps its velocity, and does not react at all to vehicle 0. Vehicle 2 (green) blocks the lane of vehicle 0.

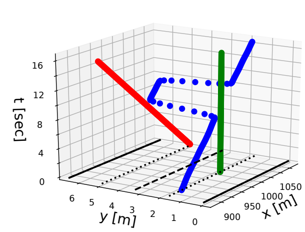
Fig. 9 depicts the trajectories of the three vehicles for different velocities for vehicle 1. When vehicle 1 drives at lower speeds, vehicle 0 chooses to drive faster to pass the bottleneck first. When vehicle 1 drives at higher speeds, vehicle 0 changes its plan according to the current situation and lets the oncoming vehicle pass before passing the obstacle. This shows that our algorithm is able to generate robust solutions even in heterogeneous environments. While the algorithm models others decisions as presented by IV-C, it explorers many possible and suboptimal future scenarios. Thus when another vehicle does not behave as assumed



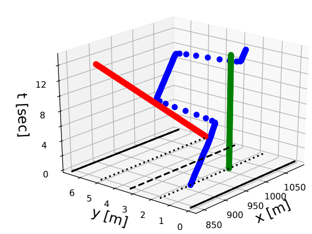
(a) $v_{red} = 5\,\mathrm{m/s}$



(b) $v_{red} = 9\,\mathrm{m/s}$



(c) $v_{red} = 13\,\mathrm{m/s}$



(d) $v_{red} = 17\,\mathrm{m/s}$

Fig. 9: Different behavior for varying levels of cooperation

DeCoH-MCTS will adjust its plan more aggressively the sooner the interaction will occur.

VII. Conclusions

In this paper, we propose a decentralized planning method of MAs based on MCTS to generate cooperative maneuvers with longer time horizons. We integrate MAs in Decoupled-UCT inspired by hierarchical reinforcement learning. By only specifying the initial and terminal conditions of MAs, the execution of MAs and the choice over MAs are learned simultaneously. The tests under several conflict scenarios show that our algorithm is able to handle a variety of conflict scenarios and shows potential over traditional MCTS.

Future work will focus on state abstraction, which will allow to share knowledge of macro-actions between different depths in the tree as well as enable the recycling of the search tree, requiring even fewer iterations as possible future states have already been evaluated in previous plans. Another aspect will be the integration of a learned prior distribution over actions as well as macro actions.

VIII. Acknowledgements

We wish to thank the German Research Foundation (DFG) for funding the project Cooperatively Interacting Automobiles (CoInCar) within which the research leading to this contribution was conducted. The information as well as views presented in this publication are solely the ones expressed by the authors.

References

[1] M. Bahram, A. Lawitzky, *et al.*, "A Game-Theoretic Approach to Replanning-Aware Interactive Scene Prediction and Planning," *IEEE Transactions on Vehicular Technology*, 2016.

[2] R. Elvik, "A review of game-theoretic models of road user behaviour," *Accident Analysis and Prevention*, 2014.

[3] T. Vodopivec and B. Ster, "On Monte Carlo Tree Search and Reinforcement Learning Spyridon Samothrakis," *Journal of Artificial Intelligence Research*, 2017.

[4] D. Silver, A. Huang, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, 2016.

[5] D. Silver, J. Schrittwieser, *et al.*, "Mastering the game of Go without human knowledge," *Nature*, 2017.

[6] C. B. Browne, E. Powley, *et al.*, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, 2012.

[7] M. Kearns, Y. Mansour, and A. Y. Ng, "A sparse sampling algorithm for near-optimal planning in large Markov decision processes," in *IJCAI International Joint Conference on Artificial Intelligence*, 1999.

[8] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, 2003.

[9] R. Axelrod and W. D. Hamilton, "The Evolution of Cooperation," *Science*, 1981.

[10] M. Düring and P. Pascheka, "Cooperative decentralized decision making for conflict resolution among autonomous agents," *IEEE International Symposium on Innovations in Intelligent Systems and Applications*, 2014.

[11] D. Swaroop and J. K. Hedrick, "String stability of interconnected systems," *IEEE Transactions on Automatic Control*, 1996.

[12] C. Stiller, G. Farber, and S. Kammel, "Cooperative cognitive automobiles," *IEEE, Intelligent Vehicles Symposium*, 2007.

[13] P. Pascheka and M. During, "Advanced cooperative decentralized decision making using a cooperative reward system," in *IEEE International Symposium on Innovations in Intelligent Systems and Applications*, 2015.

[14] D. Lenz, T. Kessler, *et al.*, "Tactical cooperative planning for autonomous highway driving using Monte-Carlo Tree Search," in *IEEE Intelligent Vehicles Symposium*, 2016.

[15] Cowling, Peter I., E. J. Powley, and D. Whitehouse, "Information set monte carlo tree search," *IEEE Transactions on Computational Intelligence and AI in Games*, 2012.

[16] D. Soemers, "Tactical Planning Using MCTS in the Game of StarCraft," Ph.D. dissertation, Maastricht University, 2014.

[17] R. S. Sutton, D. Precup, *et al.*, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, 1999.

[18] R. S. Sutton, D. Precup, and S. P. Singh, "Intra-option learning about temporally abstract actions," in *ICML*, 1998.

[19] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of Artificial Intelligence Research*, 2000.

[20] E. J. Powley, D. Whitehouse, and P. I. Cowling, "Monte Carlo Tree Search with macro-actions and heuristic route planning for the Physical Travelling Salesman Problem," in *IEEE Conference on Computational Intelligence and Games*, 2012.

[21] D. Perez, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, and S. M. Lucas, "Solving the physical traveling salesman problem: Tree search and macro actions," *IEEE Transactions on Computational Intelligence and AI in Games*, 2014.

[22] M. de Waard, D. M. Roijers, and S. C. Bakkes, "Monte carlo tree search with options for general video game playing," in *IEEE Conference on Computational Intelligence and Games*, 2016.

[23] C. Paxton, V. Raman, *et al.*, "Combining Neural Networks and Tree Search for Task and Motion Planning in Challenging Environments," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[24] M. Toussaint, "Hierarchical Monte-Carlo Planning," in *AAAI Conference on Artificial Intelligence*, 2015.

[25] A. Bai, S. Srivastava, and S. Russell, "Markovian state and action abstractions for MDPs via hierarchical MCTS," in *IJCAI International Joint Conference on Artificial Intelligence*, 2016.

[26] S. Mahadevan, M. Ghavamzadeh, *et al.*, "Hierarchical Approaches to Concurrency, Multiagency, and Partial Observability," *Learning and Approximate Dynamic Programming: Scaling up to the Real World*, 2004.

[27] M. Ghavamzadeh, S. Mahadevan, *et al.*, "Hierarchical multi-agent reinforcement learning," *Auton Agent Multi-Agent Sys*, 2006.

[28] M. Liu, K. Sivakumar, *et al.*, "Learning for Multi-robot Cooperation in Partially Observable Stochastic Environments with Macro-actions," 2017.

[29] C. Amato, G. D. Konidaris, *et al.*, "Planning with Macro-Actions in Decentralized POMDPs," *International conference on Autonomous Agents and Multi-Agent Systems*, 2014.

[30] Omidshafiei, Shayegan and Agha-Mohammadi, Ali-Akbar and others, "Decentralized control of partially observable markov decision processes using belief space macro-actions," in *International Conference on Robotics and Automation*, 2015.

[31] S. Ulbrich, S. Grossjohann, *et al.*, "Structuring Cooperative Behavior Planning Implementations for Automated Driving," in *IEEE International Conference on Intelligent Transportation Systems*, 2015.

[32] M. J. Tak, M. Lanctot, and M. H. Winands, "Monte Carlo Tree Search variants for simultaneous move games," in *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2014.

[33] M. Lauer and M. Riedmiller, "An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems," in *International Conference on Machine Learning, ICML*, 2000.

[34] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *International Conference on Machine Learning, ICML*, 1999.

[35] D. Silver, J. Veness, *et al.*, "Monte-Carlo Planning in Large POMDPs," in *Advances in neural information processing systems, NIPS*, 2010.