Monte-Carlo Policy-Tree Decision Making

Acshi Haggenmiller[1] Edwin Olson[1,2]

Abstract— **In this paper, we propose Monte-Carlo Policy-Tree Decision Making (MCPTDM), an uncertainty-aware framework for high-variance planning problems with multiple dynamic agents. Planning when surrounded by multiple uncertain dynamic agents is hard because we cannot be certain of either the initial states or the future actions of those agents, leading to an exponential explosion in possible futures. Many important real-world problems, such as autonomous driving, fit this model.**

To address these difficulties, we combine Multi-policy Decision Making (MPDM) and Monte Carlo tree search (MCTS) and perform policy tree search with marginal action cost (MAC) estimation and repeated belief particles. We first design a synthetic experiment to evaluate these novel improvements in isolation. Then we evaluate the complete framework in a self-driving car simulation experiment and compare it against MPDM and Efficient Uncertainty-aware Decision Making (EUDM) methods. We release our complete source code for replicating our experiments and results.

I. Introduction

Planning with uncertainty is difficult because uncertainty compounds and marginalizing over each source of uncertainty is exponential in the number of possibilities. First, the space of possible action sequences increases exponentially with the length of the planning horizon and the number of dynamic agents. Second, uncertainty about the future world necessarily increases the further into the future we plan. The first difficulty poses computational challenges, while the second means that continuous re-planning is necessary to take advantage of new information.

Planning under uncertainty is often modelled as a Partially Observable Markov Decision Process (POMDP) [1][2], with a discrete set of states, actions, and observations, and with probabilistic state transition, observation, and reward functions. Exactly solving a real-world POMDP is intractable because of the exponential nature of the probabilistic belief space. Tools that approximately solve the exact POMDP are still only tractable for small discrete problems. More realistically, this computational cost can be made tractable through use of heuristics [3], sampling approaches [4][5], or domain-specific modeling simplifications [6]. Even so, the number of future scenarios to consider is still an exponential function of the number of possible actions (branching factor) and the length of the horizon (search depth). A brute-force tree search over all possible plans will only be possible when the action space is both discrete and small, the horizon is short, and the time discretization is coarse.

The Multi-Policy Decision-Making [7] (MPDM) framework is helpful for these kinds of planning problems because computation time is linear in both the number of policies and the length of the planning horizon. Instead of planning in action space and directly considering each possible control input, MPDM plans in policy space and only considers selecting from high-level closed-loop policies that encode domain-specific behaviors. MPDM handles uncertainty in other dynamic agents by sampling their states and assuming that they are also following policies, again limiting the computational complexity. By having policies that encode the breadth of reasonable behaviors for both the ego (the agent that we are planning for) and other agents, MPDM takes advantage of prior domain knowledge to avoid searching extremely unlikely and unrealistic portions of the complete search tree. Policies can also be used for both discrete and continuous action spaces. Besides both the ego agent and other agents being restricted to following a policy, MPDM is also limited in that it does not consider the possibility of switching policies within the planning horizon. This makes certain larger-scale behaviors, such as an autonomous vehicle passing another vehicle and then returning to its original lane, much more awkward to handle.

Efficient Uncertainty-aware Decision-Making (EUDM) [8] extends MPDM to help get around this limitation by using a tree search to allow up to one policy change at some future point in the planning horizon and also by using heuristics to identify situations with the obstacle agents that may lead to dangerous situations. This helps EUDM more effectively marginalize over uncertainty in the initial states and plans of the other agents. Even with policies, however, the number of possible initial belief states is still exponential in the number of obstacle vehicles to plan around.

We make further improvements to MPDM to get around the limitation of a single policy change and necessity of using critical-situation heuristics by combining insights from both MPDM and Monte Carlo Tree Search (MCTS) along with additional novel modifications that take advantage of the unique cost-structure and focus on safety in autonomous driving and other similarly structured tasks.

The principle contributions of this paper include:

1) Monte-Carlo Policy-Tree Decision Making (MCPTDM), which allows an agent to efficiently compute policies in partially-observable and stochastic problems having either discrete or continuous action and state spaces. It does this by composing its policy from a sequence of simpler policies.
2) Marginal action cost (*MAC*) estimation, which improves upper confidence bound (UCB)-based tree exploration and final action selection.
3) "Particle repetition", which improves fairness in cost

[1]Robotics Institute, University of Michigan
[2]Computer Science and Engineering Department, University of Michigan
{acshikh,ebolson}@umich.edu

estimation by reducing the effect of unlucky or unusual initial conditions.

4) Validation of our MAC estimation and particle repetition improvements on an abstract task.
5) Evaluation and comparison with EUDM [8] on an autonomous self-driving task.
6) Openly-released source code capable of reproducing all the figures and evaluation results from this paper for full replication of our results and further extensions.

II. Related Work

In this section, we categorize previous work into several families.

Many of these works use the Partially Observable Markov Decision Process (POMDP) [1][2] to frame their methodology. A POMDP consists of finite sets of states, actions, and observations, as well as stochastic transition, reward, and observation functions that may depend on both the current unknown state and chosen action. Littman's POMDP tutorial [9] is a friendly introduction to the topic.

We first show what these POMDP components might represent in a self-driving vehicle scenario. While the poses of the ego-vehicle and other vehicles can be directly observed, the intentions of other vehicles cannot, so our total system state is non-observable. Since the total state cannot be observed, a *belief* that can represent probabilities for each possible state must be maintained instead, and forms the de facto state used in planning. As all the vehicles execute their actions, there may be noise attributable to the environment (road surface, wind, etc.) or the vehicles themselves (limited precision engine control and steering), giving us a *stochastic transition function*. The progress our vehicle makes towards some goal location and keeping a safe distance from other vehicles to avoid crashing make up the *reward function*. Each vehicle has control over their continuous acceleration, braking, and steering control inputs, which form the *action space*. The ego vehicle's sensors continually make *observations* of the other vehicles, but can only infer the intentions of those vehicles from their position over time, making this problem *partially observable*.

Along with many other real-world problems, planning for self-driving vehicles can be modelled as a POMDP. Even when a POMDP model is not used explicitly, it can be a helpful framework for comparing and discussing methods.

A. Directly solving the POMDP

Early work, like the seminal paper from Åström [1], focused on solving for the optimal belief-space regions and the actions they map to, as well as extending from these finite horizon problems to infinite horizon problems, which sometimes required finding approximate or ϵ-optimal solutions [10]. Even with small problems, these approaches were not very efficient, with one algorithm solving a machine replacement problem with two states and two actions in 110 seconds [11].

A more recent work on directly solving a POMDP, "Grasping POMDPs" [6] from Hsiao et al., constructs reduced-size abstract state spaces for simple robot manipulation tasks by taking advantage of compliant motions (such as sliding along a surface). For a two-dimensional grasping task, the authors formed a 408-state POMDP and solved it in less than 10 minutes with a general solver.

Where possible, direct solutions to a POMDP are theoretically robust, but few practical problems are simple enough to be solved like this.

B. Simplification through macro-actions

A macro-action is a closed-loop policy with a termination condition that can be selected by an agent as an option in addition to its primitive actions, and were initially introduced as a tool to speed up reinforcement learning [12].

Theocharous and Kaelbling [13] employ macro-actions to approximately solve a POMDP. They discretize the belief space into a sparse dynamic multi-resolution grid and use hand-crafted macro-actions, like go-to-end-of-corridor, to avoid producing and evaluating non-productive intermediate states. Macro-actions have also been generated online, such as in PUMA (Planning under Uncertainty with Macro-Actions) [14], which generates macro-actions based on subgoals of reward and information gain and uses anytime refinement to progressively increase the resolution of their macro-actions.

Macro-actions share many similarities with the policies used in MPDM in that both are larger closed-loop abstractions over primitive actions, reducing the effective size of the search space in planning algorithms. The primary difference is that macro-actions have termination conditions and must still be composed, sometimes even with primitive actions. In multi-policy, policies do not necessarily have a termination condition nor need to be composed together during planning. In this sense, these policies are less flexible than macro-actions, but allow for faster planning in exchange for this lower flexibility.

C. Analytical uncertainty propagation with Gaussians

In some cases, a POMDP can be linearized in such a way that uncertainty can be represented and propagated as simple Gaussian distributions. In these cases, planning decisions can be made directly with respect to the analytical likelihood of acceptable outcomes.

For example, in linear-quadratic Gaussian motion planning (LQG-MP) [15], the authors linearize and jointly model motion planning, control, and state estimation with Gaussian uncertainty. This enables the planner to precompute the robot's covariance and potential deviation from its nominal path and choose the path most likely to succeed from some set of candidates. Similarly, FIRM (feedback controller-based information-state road map) [16] precomputes a probabilistic roadmap "FIRM graph", where each directed edge is a linear quadratic Gaussian (LQG) feedback controller designed to drive the belief into the neighborhood of the next node. So too the rapidly-exploring random belief tree (RRBT) [17], which iteratively builds and refines a tree of possible trajectories with local LQG feedback control to

propagate state covariance, maintain chance-constraints, and avoid collisions.

By taking advantage of the composability of Gaussians with LQG control, these methods significantly factor out uncertainty. While this is a big advantage, modeling all uncertainty as Gaussian is not suitable for highly non-linear problems like autonomous driving.

D. Sampling-based approaches

In contrast to the computational gains possible when propagating Gaussians, sampling-based approaches recognize that arbitrary probability distributions can be approximated with enough random samples. The true value of some uncertain process can be determined by using sampling to marginalize over each source of uncertainty involved.

An early reinforcement learning approach extended the use of Monte Carlo sampling from MDPs to POMDPs [4], alternating between estimating Q-values for the current policy and then improving the policy accordingly.

Because even representing the whole state space to store probabilities or Q-values can be intractable in larger problems, most approaches only represent parts of the space that they have explicitly explored. The Infinite POMDP (iPOMDP) algorithm [18] assumes that the target POMDP has an unbounded discrete state space and then extends an infinite hidden Markov model (iHMM) [19] to also handle actions and rewards, estimates belief with an approach based on beam-sampling [20], and uses a stochastic forward search for action selection. Partially Observable Monte Carlo Planning (POMCP) [5] approximately solves a POMDP given only a black-box simulator with Monte Carlo sampling, an unweighted particle filter to approximately represent the belief state, and MCTS to explore the action space.

For multi-agent scenarios, factored-value POMCP [21] extends POMCP to multi-agent problems by keeping separate trees for each factor (which may contain 1 or more agents) and applying the variable elimination algorithm [22] with their "mixture of experts optimization" to choose actions. Alternatively, decentralized Monte Carlo tree search (Dec-MCTS) [23] performs multi-robot planning by having each robot alternate between stages of growing/deepening a Monte Carlo search tree and updating the probability distributions of other robots.

Sampling is an effective way to explore high-dimensional spaces and to marginalize over uncertainty. Effective planning often only needs a representative sample of situations. In MCPTDM, we also use sampling to marginalize over uncertainty in our belief.

E. With neural-network learning

To solve a POMDP, a neural network may be trained with inputs corresponding to the ego agent's state and beliefs and with a loss function based on the cost or reward. By training a neural network to solve a POMDP, the network can hopefully encode expert rules and heuristics without the algorithm itself needing to specify this domain knowledge. This allows the algorithm to maintain more generality by only needing domain-specific knowledge for high-level objectives through a reward or cost function.

Deep Mind's Alpha Zero [24] plays Go at an expert-level by combining MCTS with deep learning networks for evaluating board positions and selecting moves. As a fully-observable game with a discrete action space, Go is not a perfect analog for robotics applications, but its large search space and difficulty evaluating moves and states make it very interesting. A0C [25] extends Alpha Zero to work in continuous action spaces by using progressive widening, by sampling and selecting promising new with the policy network, and by training the policy network to produce a valid probability distribution from the relative MCTS results by using the loss to enforce that it integrates to 1.

When actually applied to partially observable scenarios, most methods use deep learning for only part of the method. Ding et al. [26] use a recurrent neural network to examine multi-agent interactions and predict behaviors before they can be directly observed. Paxton et al. [27] use reinforcement learning to learn both low-level control policies as well as high-level goal-directed "option" policies to search through with MCTS; they use extracted feature inputs and a single hidden layer. Mukadam et al. [28] use deep Q-learning to find a high-level tactical lane-changing strategy that determines when to elect lane changing or acceleration actions to be performed by a low-level controller.

Because many neural network approaches focus on learning policies, they could be integrated into a multi-policy framework. In this paper, however, we use simpler policies.

F. Multi-policy approaches

In Multi-Policy Decision-Making (MPDM) [7], the ego agent elects one policy from a small set of high-level closed-loop policies and also models the other dynamic agents with these same policies. Monte Carlo sampling is used to marginalize over uncertainty in the state and future policies of the obstacle agents. In each Monte Carlo sample, all the agents are forward simulated together to capture closed-loop interactions [29] and the best ego agent policy is elected from the cumulative results. Further work in the MPDM framework focuses on discovering risky configurations through stochastic gradient descent of a heuristic cost function [30] and back-propagation techniques [31].

Other approaches incorporate multi-policy ideas into a larger framework. Zhou et al. [32] use a POMDP solver to determine optimal acceleration/deceleration actions for each obstacle agent policy and a particle filter to estimate the probability of each policy-agent pair and then combine this with model predictive control for the ego agent. In Efficient Uncertainty-aware Decision Making (EUDM) [8], MPDM is extended with a limited tree search and critical-scenario heuristics, as mentioned in the introduction.

Our proposed method is similar to EUDM because we also extend MPDM by modifying the policy search process and permitting policy changes in the planning horizon. However, we go farther in searching over arbitrary sequences of policies, rather than allowing only a single change in

policies. To make the larger search space tractable requires reformulating the search into a MCTS-type tree search, but where instead of evaluating actions at each decision point, we evaluate closed-loop policies.

III. Synthetic abstract scenario

We first examine Monte-Carlo Policy-Tree Decision Making (MCPTDM) with a synthetic scenario and experiments that model an abstract form of the self-driving scenario we will examine later in this paper. After describing this scenario in detail, we examine the effects of changing the expected-cost rule used by UCB for balancing the exploration-exploitation tradeoff and for selecting the final best action. We also examine the effects of various improvements to UCB. Finally, we explore the idea of *fairness with particle repetition*, helping to mitigate the effects of "unlucky" initial conditions, where poor outcomes are more attributable to the initial conditions than the specific plan being evaluated. In a self-driving situation, for example, an "unlucky" particle might include nearby vehicles having intentions that box the ego vehicle in while another vehicle performs a dangerous lane-change; boxed in like this, it doesn't matter which policy the ego vehicle chooses, even though this random coordination is very unlikely.

A. Problem statement

We consider an abstract version of an autonomous driving task with five policy (or action) choices, a time horizon split into four segments, and costs related to avoiding crashes and close calls and making forward progress.

While costs associated with making forward progress are likely to be relatively smooth, costs around safety and potentially crashing are more discontinuous. To model a more complex cost distribution, we use a mixture of two Gaussians. Gaussian mixtures have prior use in compactly approximating real-world events [33][34].

In addition, in a self-driving scenario, a lot of the uncertainty is in the initial belief about the behavior and intentions of other vehicles. Specific initial conditions that might be dangerous for one ego-agent policy are likely to be dangerous for the other policies as well. To model this "risky situation" correlation, we store the initial conditions in what we call *belief particles*. If the same belief particles are propagated through different paths in the tree, we should see correlated responses.

This synthetic scenario is effectively a Markov Decision Process (MDP) with a fully observed/known problem model and with stochastic rewards. The planning problem is to effectively explore the scenario as a black-box model in simulation before electing a final action choice.

In order to determine the empirical performance of our method, we perform a synthetic experiment. We construct a tree (of depth 4 and branching factor 5) and assign each node a random cost probability distribution that is a mixture of two Gaussian distributions with random mean μ_i, standard deviation σ_i, and mixture weight w_i, where the mixture weights sum to one. Because our model includes only positive costs, we saturate these Gaussian costs to be in the range $[0, 2\mu_i]$ so that the mean is not changed by the saturation.

When running a trial, we first sample a "belief particle" from the "initial conditions" so that all costs using this belief particle will be correlated. We do this by sampling z-scores from the standard Gaussian distribution (a z-score is a normalized Gaussian sample, where $z = (x-\mu)/\sigma$). We also sample a weight threshold t from 0 to 1 that we will use to select between the Gaussian mixture components. With these z-scores and threshold z_1, z_2, t from the belief particle known, the cost c for all node distributions would be both correlated and deterministic:

$$c = \begin{cases} \text{constrain}(\mu_1 + z_1\sigma_1, 0, 2\mu_1) & t \leq w_1 \\ \text{constrain}(\mu_2 + z_2\sigma_2, 0, 2\mu_2) & t > w_1 \end{cases} \tag{1}$$

$$\text{constrain}(x, l, h) = \begin{cases} l & x < l \\ h & x > h \\ x & \text{otherwise} \end{cases} \tag{2}$$

For each trial, after sampling a situation, we use UCB (or one of its variants) with the selected expected-cost rule to run that situation through the tree. After running all trials in our computational budget, we choose the lowest expected-cost top-level action as our result. We calculate the true expected costs of both the best path through the tree and also the best path starting from our chosen action. Our *regret* for this choice is then the chosen-action best-path expected cost minus the overall best-path expected cost.

When sampling node distributions, Gaussian means and standard deviations are all sampled uniformly and independently from 0 to 100.

The true marginal expected cost of any node i is then:

$$\mathbb{E}(\bar{c}_i) = w_{i,1}\mu_{i,1} + (1 - w_{i,1})\mu_{i,2} \tag{3}$$

B. Expected-cost estimation with marginal action costs (MAC)

In this section, we motivate and describe a focus on *marginal action costs* (MACs), which allow us to make a more informed exploration of the search tree as compared to just using terminal costs. To put this idea in context, a common application of MCTS is for board games such as Go [24] which involve many turns, a large branching factor, determinism, and a reward/cost assigned only when a terminating condition is reached (e.g. win/loss). By comparison, in a non-deterministic self-driving car scenario, while there may be a long-term goal (e.g. a destination to reach), the most important goal of the planner is to maintain safety at all times. To this end, MACs allow the search to distinguish, for example, between a collision at depth 1 and depth 4, and due to the high cost of collisions, the entire sub-tree below a collision can be effectively pruned away. This is only possible when the collision can be attributed to a specific node.

The expected cost of MCTS nodes may be used at two different points in MCTS. In the first case, they are inputs

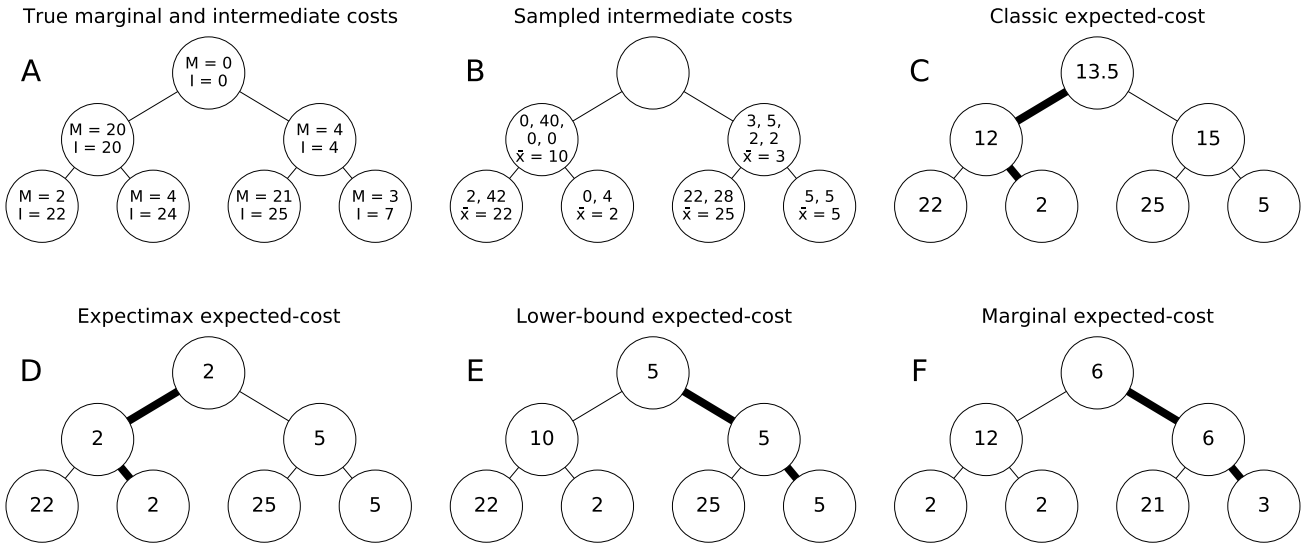

Fig. 1: Illustration and comparison of four different rules for estimating the expected-cost of an MCTS-style node. Each rule's selected best path is in bold. For this example, only the "lower-bound" and "marginal action cost" (MAC) rules choose the optimal path. **A**: Each action choice and node has some cost distribution. We label the mean of each node's distribution as the "true marginal cost" M of that node. We also label the "true intermediate cost" I of that node, which is the sum of marginal costs leading to that node. The optimal choice in this example would be to choose actions right-right, resulting in a total expected cost of only 7. **B**: We run 2 trials for each leaf node, recording the intermediate costs of each trial after each action. In a more classical MCTS formulation, costs (or rewards) would only be known at the leaf nodes. We also label the mean intermediate cost at each node. For the two middle nodes, the first two costs are associated with the left child and the second two costs are associated with the right child. **C**: Under the "classic" expected-cost rule, we use the mean value of every trial under a node, ignoring the intermediate costs of non-leaf nodes. **D**: An optimistic alternative "expectimax" is to set the expected-cost of a node to be equal to the best expected-cost of its children. This makes many actions appear much better than before. **E**: The "expectimax" rule is overly optimistic when a trial gets lucky and avoids a large intermediate cost in a parent node. We can use the mean parent intermediate cost as a "lower bound" on the expected-cost. **F**: Alternatively, we can also attempt to directly estimate the original "marginal action cost" of reaching each node from the intermediate costs. Then we set the estimated-cost to the marginal cost plus the best estimated-cost of a child.

to the UCB selection algorithm for guiding the exploration-exploitation tradeoff in search. In the second case, after the computational budget for MCTS trials is met, they may be used for *final action selection*, to select the final top-level action to execute. Besides using the expected-cost, a common alternative is to choose the most-visited action as the final choice [35]. A slight improvement may be found by continuing to search until both these measures agree [36] and so we use that combined "max-robust child" variation in this paper, putting a limit of 20% on the number of additional trials we might run. This explicit limit is only necessary because some of the parameter sweeps we perform include degenerate cases that do not converge.

1) Traditional MCTS expected cost estimation: The expected cost of an MCTS node is normally the mean of each trial that has passed through it [37]:

$$\bar{c}_i = \frac{1}{N_i} \sum_{k=1}^{N_i} c_{i,k} \tag{4}$$

where $\bar{c}_i$ is the expected cost of node i, N_i is the total number of trials that have passed through node i, and $c_{i,k}$ is the kth final trial cost that passed through node i. We call this rule "classic". (See Fig. 1 C.)

Instead of just taking the mean of all terminal costs from all child nodes, we can optimistically take the best (lowest) child cost at each controlled choice, only averaging over the multiple stochastic costs at leaf nodes. The expected costs then bubble up from the bottom of the tree:

$$\bar{c}_i = \begin{cases} \min_{j \in \text{Children}(i)} \bar{c}_j & i \text{ is a branch} \\ \frac{1}{N_i} \sum_{k=1}^{N_i} c_{i,k} & i \text{ is a leaf} \end{cases} \tag{5}$$

This rule is called "expectimax" [38] (although we are minimizing costs here instead of maximizing rewards) and was originally devised in the context of game tree search where the opponent plays stochastically. It is considered a generalization of minimax search, where the opponent is assumed to play optimally [37]. (See Fig. 1 D.)

2) Taking advantage of known intermediate costs: We observe that the above rule can be over-optimistic in some cases. For example, imagine a node corresponding to a risky action that has a 50% chance of observing a high cost, and that has two child nodes, neither of which impose any costs. If we run a trial for each of these children and get a high cost for one and zero for the other, we would optimistically choose the best option and assign an expected cost of zero to the parent node, even though the true expected cost is high. However, when we have intermediate costs for each node, we know that the high cost is attributable to the parent node and not to either of the children. We should not be deceived into believing that a low cost path exists for one of the children since the parent cost applies to both. We devise a new rule that takes the cost of the best child and applies

a lower bound of the mean partial/intermediate cost of the parent node:

$$\bar{c}_i = \max\left(\bar{p}_i, \begin{cases} \min_{j \in \text{Children}(i)} \bar{c}_j & i \text{ is a branch} \\ 0 & i \text{ is a leaf} \end{cases}\right) \quad (6)$$

where $\bar{p}_i$ is the mean partial/intermediate cost of node i. We call this rule "lower bound". (See Fig. 1 E.)

3) Marginal action costs (MAC): Taking this idea of using additional information from intermediate costs even further, we can also directly calculate and assign marginal costs to each node and use the sum of mean marginal costs along the best path through the tree as the total expected cost:

$$\bar{c}_i = \bar{m}_i + \begin{cases} \min_{j \in \text{Children}(i)} \bar{c}_j & i \text{ is a branch} \\ 0 & i \text{ is a leaf} \end{cases} \quad (7)$$

where $\bar{m}_i$ is the mean marginal cost of node i, equal to the mean intermediate cost of node i minus the mean intermediate cost of the parent, if any. We notice a similarity to Q-learning in that the total expected value combines an immediate value with the best child node/successor state expected value. We call this rule "marginal" or MAC. (See Fig. 1 F.)

C. Fairness and particle repetition

At the start of each Monte Carlo trial we sample a set of initial conditions. This *particle* then takes some path through the tree, influencing the final cost of the trial. Some of these particles may be either lucky or unlucky, making the actions and path they take look unfairly better or worse than they actually are. For example, if all the obstacle vehicles either simultaneously avoid or crowd the ego vehicle, such that no matter the ego vehicle's policy choice, there is either no possible crash or an inevitable crash. The idea is that with enough trials and particles, these effects will average out to be fair in the end, resulting in a good outcome. However, when the number of trials is computationally limited, it may help if we intentionally repeat particles along different paths through the tree to reduce any bias resulting from this good or bad luck.

We do this by recording all particles, the paths they take, and their terminal costs (because our initial conditions are compact, this does not add up to a significant amount of information). Then, after a top-level action has been elected for the next trial, we first check if there is a particle that has not gone down this path before. If so, we repeat this particle instead of sampling a new one. If there are multiple particles, we chose the one with the highest cost. We note that although it would be possible to perform particle repetition at any depth of the tree, we only found it worthwhile at the top level.

Once the computational budget is large enough, repeating particles for fairness will no longer be necessary. Instead, it may be more effective to only draw new particles to better marginalize uncertainty. While for smaller numbers of Monte Carlo trials it is helpful to repeat particles as much as possible, for large numbers it may be best to not repeat at all. We find that the best number of particles to repeat appears to be inversely proportional to the total number of trials in our budget. This means that for an any-time implementation, it would be best to have an estimated budget before starting. We use a repetition constant to set the maximum number of particle repetitions to perform as the repetition constant divided by the number of trials in our budget.

Algorithm 1: The MCPTDM algorithm, where the inputs include initial state s_0, uncertainty belief b, and policy set P. We use p for policy, s for belief sample/initial conditions particle, n' for a child node of n, m for a marginal action cost, m_n for the set of marginal action costs observed by node n, and $\bar{c}_n$ for the expected cost of node n. In practice, an implementation may want to limit the number of repeated particles. A system may also want to specify a preference, in addition to UCB, to determine expansion order when multiple children are completely unexplored; we prefer to first explore the child with the same policy as the parent.

```
function CHOOSEPOLICY(s_0, b, P)
    n ← CREATENODE(s_0)
    k ← 0
    for k < trial_budget ∨ continue for max-robust child
     do
        RECURSE(n, DRAWBELIEFSAMPLE(b), P)
        k ← k + 1
    end
    return arg min_p c̄_{n'} for child n' with policy p
function RECURSE(n, s, P)
    if depth of n ≥ max depth then
        return
    end
    if n not expanded then
        // Add child nodes at depth+1 for each policy in P
        EXPAND(n, P)
    end
    n' ← KL-UCB(n)
    if depth of n = 0 then
        // Save/retrieve particles before forward simulation
        if n' has unplayed particles then
            s ← WORSTUNPLAYEDPARTICLE(n')
        else
            SAVEPARTICLE(n, s)
        end
    end
    m ← FORWARDSIMULATE(n', s)
    m_{n'} ← m_{n'} ∪ {m} // Track marginal costs
    RECURSE(n', s, P)
    c̄_n ← MACEXPECTEDCOST(n, s)
```

D. Experiments

To evaluate and compare the different expected-cost rules, UCB variations, and particle repetition, we perform a variety of parameter sweeps. When not sweeping over Monte Carlo

trials, we instead marginalize the number of trials with ten powers of two from 8 to 4,096. We perform enough complete runs of the algorithm to show significant results in the figures, where error bars indicate plus or minus one standard deviation of the mean (standard error).

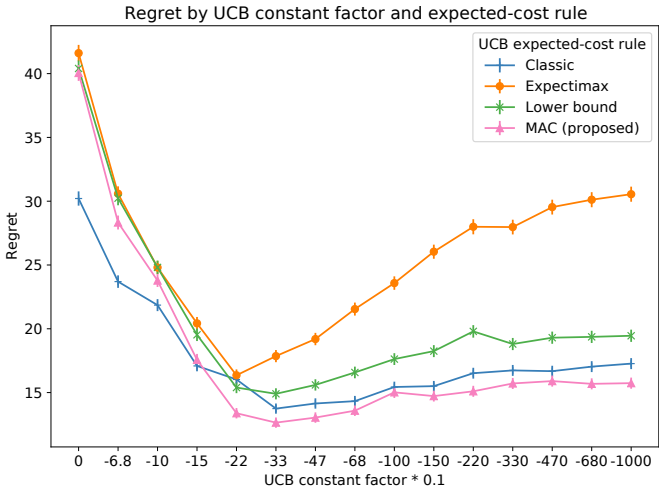

Fig. 2: Parameter sweep of UCB constant for each expected-cost rule, showing that the marginal action cost (MAC) and "classic" rules perform best.

1) On expected-cost rules: We start by sweeping the UCB constant (a free parameter) for each expected-cost rule. Fig. 2 shows that the classic and MAC rules consistently outperform the expectimax and lower-bound rules, and that these differences persist even for large UCB constants which may encourage almost pure exploration. We reason that these differences reflect the effect of the expected-cost rules not on exploration, but on final action selection.

To tease apart how the expected-cost rules influences UCB and final action selection, we repeat the experiment, this time always using MAC for final action selection, but still varying the expected-cost rule for UCB. In Fig. 3 we see this hypothesis confirmed, that the MAC rule is most important for final action selection and that most of the expected cost rules work just as well with UCB. We also include a uniform (pure exploration) rule for comparison.

We also compare expected-cost rules by computational budget (number of Monte Carlo trials) in Fig. 4. Each rule uses the best UCB constant for it as found in Fig. 3 and all use MAC for final action selection. We notice that MAC converges close to zero mean regret much faster than the other rules.

2) On UCB variations: Acknowledging that there are many enhancements and alternatives to UCB, we also wanted to see if one of these would be able to improve performance. For this purpose we also implemented UCB-V [39], UCB(δ) [40], and KL-UCB and variation KL-UCB+ [41]. We performed parameter sweeps for each UCB variation to choose the parameters that produced the lowest regret for each, and then compared them. In Fig. 5 we see that each improved variation performs fairly similarly and significantly

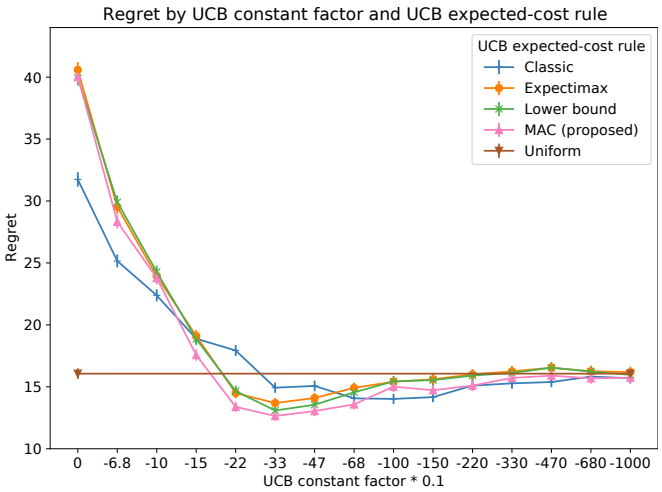

Fig. 3: Parameter sweep of UCB constant for each UCB expected-cost rule, while using MAC for final action selection. We see that as UCB values increase, each rule's performance approaches that of uniform/pure exploration.

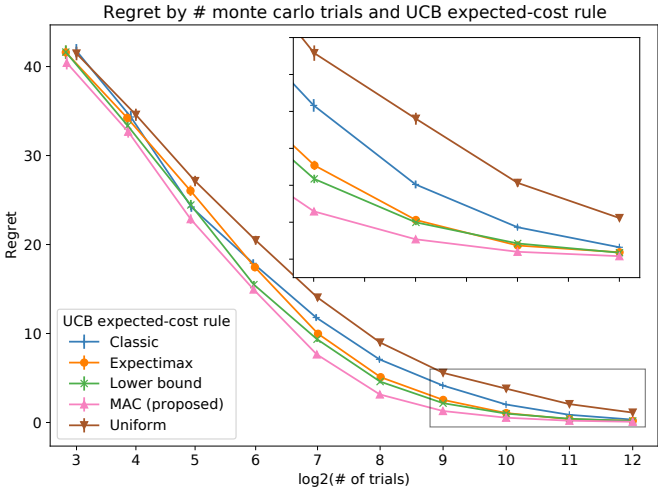

Fig. 4: Parameter sweep of Monte Carlo trials for each UCB expected-cost rule, while using MAC for final action selection. MAC achieves a low regret faster than the other rules. Thanks to also using the "max-robust child" rule to make a final decision at a good time, uniform exploration also does surprisingly well.

better than UCB, although the relative differences start to widen with the highest numbers of trials. We use KL-UCB rule for the remainder of this paper because of this advantage.

3) On particle repetition: In Fig. 6 we sweep the repetition constant and show the *relative regret*, normalizing by the regret of the w/o-repetition case. For most of the cases, we see that improvements saturate when particles are being repeated as much as possible. Intuitively, we would expect that particle repetition would become less important as the number of particles increases, since the effects of individual "unlucky" particles would be mitigated by the large number of particles. We see exactly this behavior for 1,024 Monte Carlo trials in our experiment. Because we are plotting relative regret, the smaller the absolute regret, the

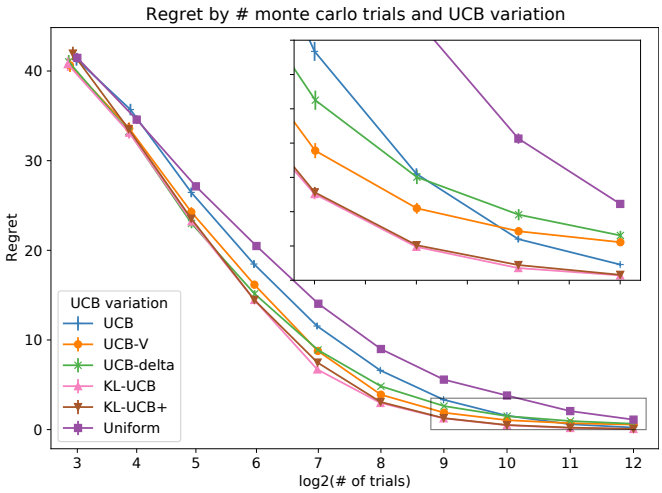

Fig. 5: Parameter sweep of Monte Carlo trials for each UCB variation, using MAC for expected-cost and final action selection. All the improved rules outperform UCB in most cases by about the same margin as UCB outperforms uniform exploration. KL-UCB consistently performs best, with KL-UCB+ performing very similarly.

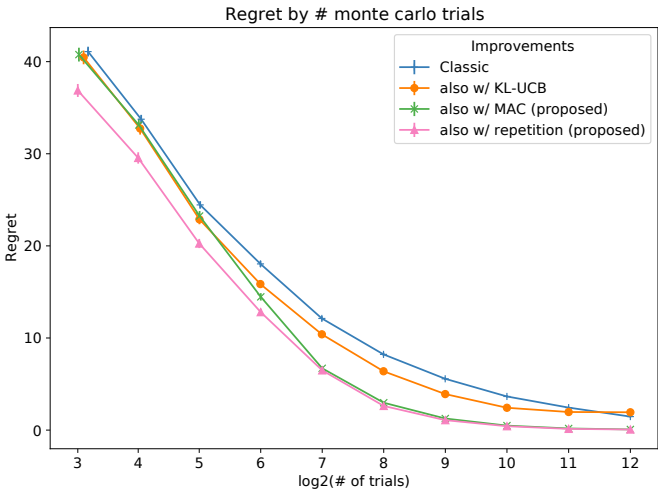

Fig. 7: Ablation study of our method, showing the advantage of starting from traditional MCTS using UCB and "max-robust child", then adding KL-UCB, marginal action costs (MAC), and finally also particle repetition. Our full enhanced method performs better than all the ablative cases.

larger the error bars. See the absolute regret trends in Fig. 7 where we use a repetition constant of 2^{16}.

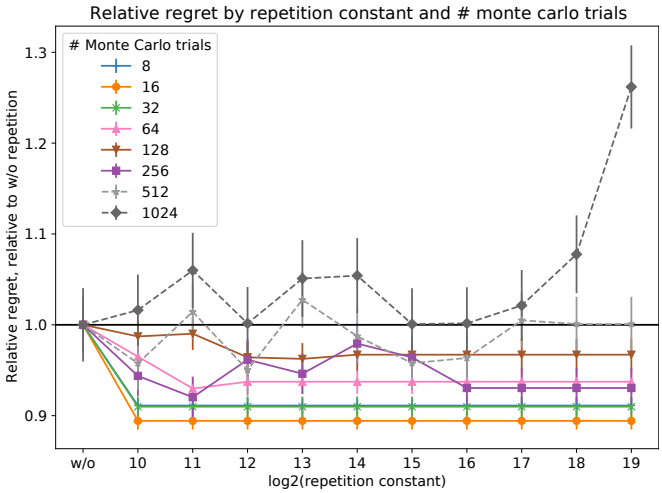

Fig. 6: Plot of relative regret (normalized by the no-repetition case), the particle-repetition constant, and the number of trials, showing that particle repetition is strictly beneficial at least up to 256 trials, with up to about a 10% reduction in regret. Note that the cases with 1,024+ trials all have very low absolute regret (see Fig. 7).

4) Cumulative improvement: Finally we perform an ablation study in Fig. 7 to compare a traditional MCTS search with UCB and "max-robust child" final action selection to our enhanced method that adds in KL-UCB, MAC expected-cost estimation, and finally particle repetition. Each addition significantly reduces regret, although at different points along the curve.

Overall, our total combined method with MAC and particle repetition does significantly better than traditional MCTS. Our complete method is shown in Algorithm 1.

IV. Autonomous Driving Scenario

We now evaluate MCPTDM on a more concrete problem suggestive of an autonomous driving scenario, very similar to that proposed by Zhang et al. for evaluating EUDM [8], but with only two-lanes going in a single direction (see Fig. 8 and 9). Where possible, we have chosen equivalent parameters and settings to the EUDM paper. Otherwise, we have chosen parameters that: provide interesting and varied behavior; give a fair comparison between each of our evaluation methods; and allow us to run simulations fast enough to generate sufficient data for our figures. Our goal is not to model real-world driving, a research problem in its own right, but to construct a benchmarking domain to elicit some of the same structural properties.

We use a bicycle model, the intelligent driver model [42], and pure pursuit lateral control [43] for all vehicles, along with five policies: left-lane-maintain, left-lane-accelerate, right-lane-maintain, right-lane-accelerate, or decelerate. Electing a policy for a different lane than the current one causes a vehicle to perform a lane-change maneuver. For integration of the overall outer simulation we use $dt = 0.01\,\mathrm{s}$ and for the inner forward simulations we use $dt = 0.2\,\mathrm{s}$.

Besides the ego vehicle, we simulate 13 "obstacle" vehicles, and obstacle vehicles are removed and respawned so that we can maintain 13 vehicles within a certain distance of the ego vehicle. This number of vehicles ensures that there may be complex interactions between multiple other vehicles both in front of and behind the ego vehicle, but also keeps the environment from being too congested. Each obstacle vehicle is parameterized by a random (within some range) preferred velocity (15-35 MPH), acceleration (1 - 2 $\mathrm{m\,s^{-2}}$), and follow-time ($0.8\,\mathrm{s}$ - $2.0\,\mathrm{s}$), to provide some variation and uncertainty in their behaviors. We chose these ranges to be compatible with our choices of dt for integration and

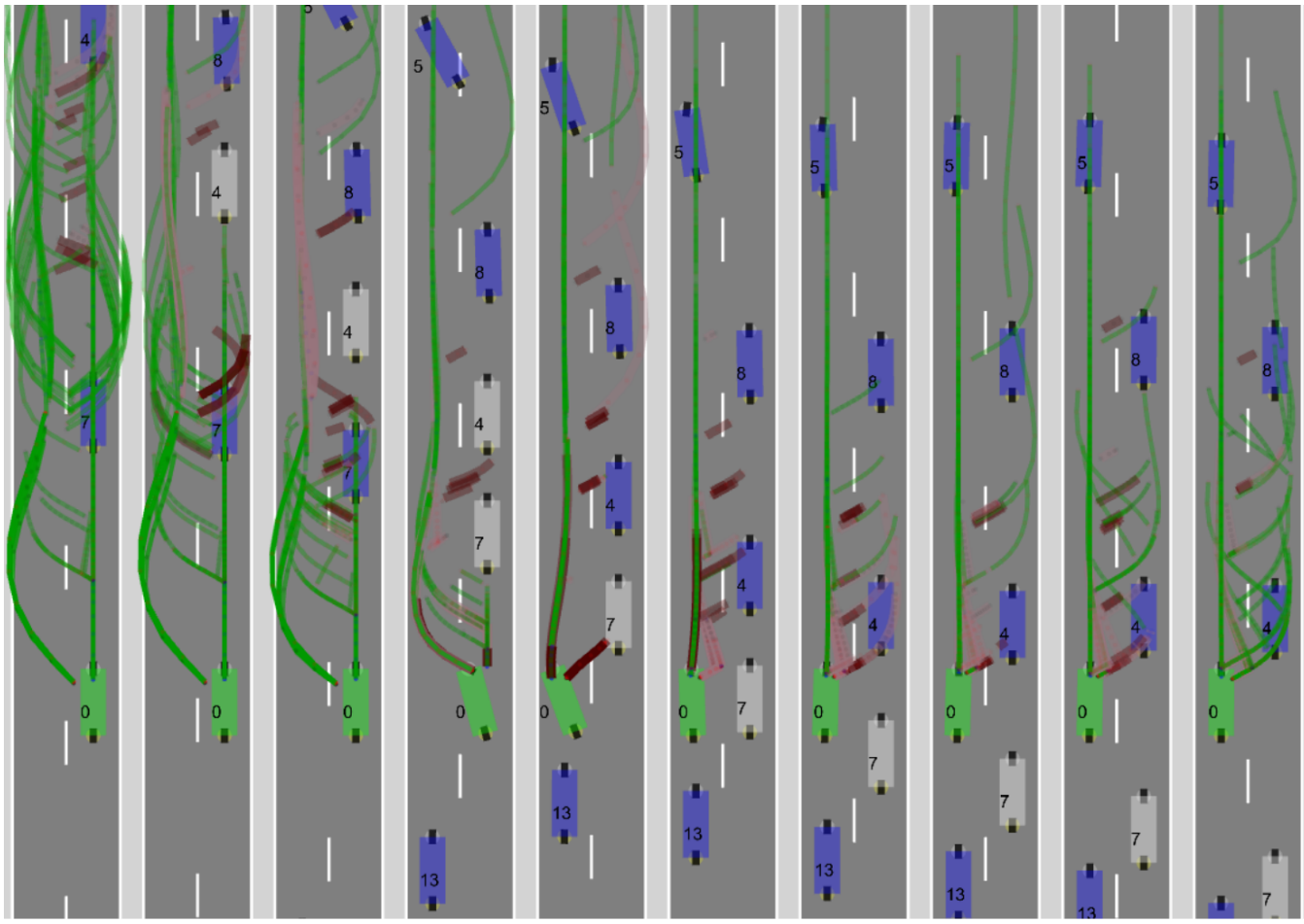

Fig. 8: MCPTDM passing a vehicle and keeping distance from others in our simulated road environment. Vehicle 4 comes to a stop ahead of vehicle 7, causing it to stop in ahead of the ego vehicle (number 0). The ego vehicle moves into the left lane, passes vehicle 7, and then keeps a slight distance behind vehicle 4. We see how MCPTDM both performs tactical passing to make forward progress and also prefers to keep distance from vehicle 4, just in case other vehicles behave erratically. The ego vehicle cuts relatively close to vehicle 13 coming from behind because its model of vehicle 13 is confident about its behavior and because the cost function does not penalize this situation.

The ego vehicle is colored green, and obstacle vehicles are either blue while moving or gray while stationary. Monte Carlo trials are shown by their forward-simulated traces, which are dark red for traces leading to a crash, pink for traces that are somewhat unsafe, and green for safe traces. Frames are left-to-right in one-second increments. A video version of this sequence is provided at `https://youtu.be/HYYTT3EYY1Q`.

to prevent spontaneous crashing between obstacle vehicles. The true values for these random parameters are considered unknown and the ego vehicle assumes nominal values for each (a preferred velocity of either the current velocity or 15 MPH, whichever is greater; acceleration of $2\,\mathrm{m\,s^{-2}}$; and a follow-time of $1.2\,\mathrm{s}$). Every 0.2 seconds, each obstacle vehicle has a small chance of randomly choosing a new policy (5% probability each second). Because obstacle vehicle changes occur randomly, the policies used by the obstacle vehicles will first check that the next lane is clear at least a half-vehicle's length ahead and behind before making a lane-change maneuver. The policies used by the ego vehicle do not make this check so they can be more flexible. As obstacle vehicles are assumed to follow a single closed-loop policy, these random policy changes are only modelled by the ego vehicle's uncertain belief over obstacle vehicle policies, described below.

The ego car tries to safely and smoothly maintain a target velocity by minimizing a cost function that incorporates velocity, safety, and control inputs:

$$C_{\mathrm{vel}} = |v - v_{\mathrm{target}}| \tag{8}$$

$$C_{\mathrm{acc}} = W_{\mathrm{acc}}\dot{v}^2 \tag{9}$$

$$C_{\mathrm{steer}} = W_{\mathrm{steer}}\dot{\theta}^2 \tag{10}$$

$$C_{\mathrm{safety}} = W_{\mathrm{safety}}(1 + e^{-k_{\mathrm{safety}}(d_{\mathrm{min}} - d_{\mathrm{safety}})})^{-1} \tag{11}$$

$$C = \int (C_{\mathrm{vel}} + C_{\mathrm{acc}} + C_{\mathrm{steer}} + C_{\mathrm{safety}})\alpha^t \,\mathrm{d}t \tag{12}$$

where v and θ are the ego vehicle's forward velocity and angle; $v_{\mathrm{target}} = 11.2\,\mathrm{m/s}$ is the ego vehicle's target velocity (25 MPH); $W_{acc} = 0.1$, $W_{steer} = 20$, and $W_{safety} = 600$

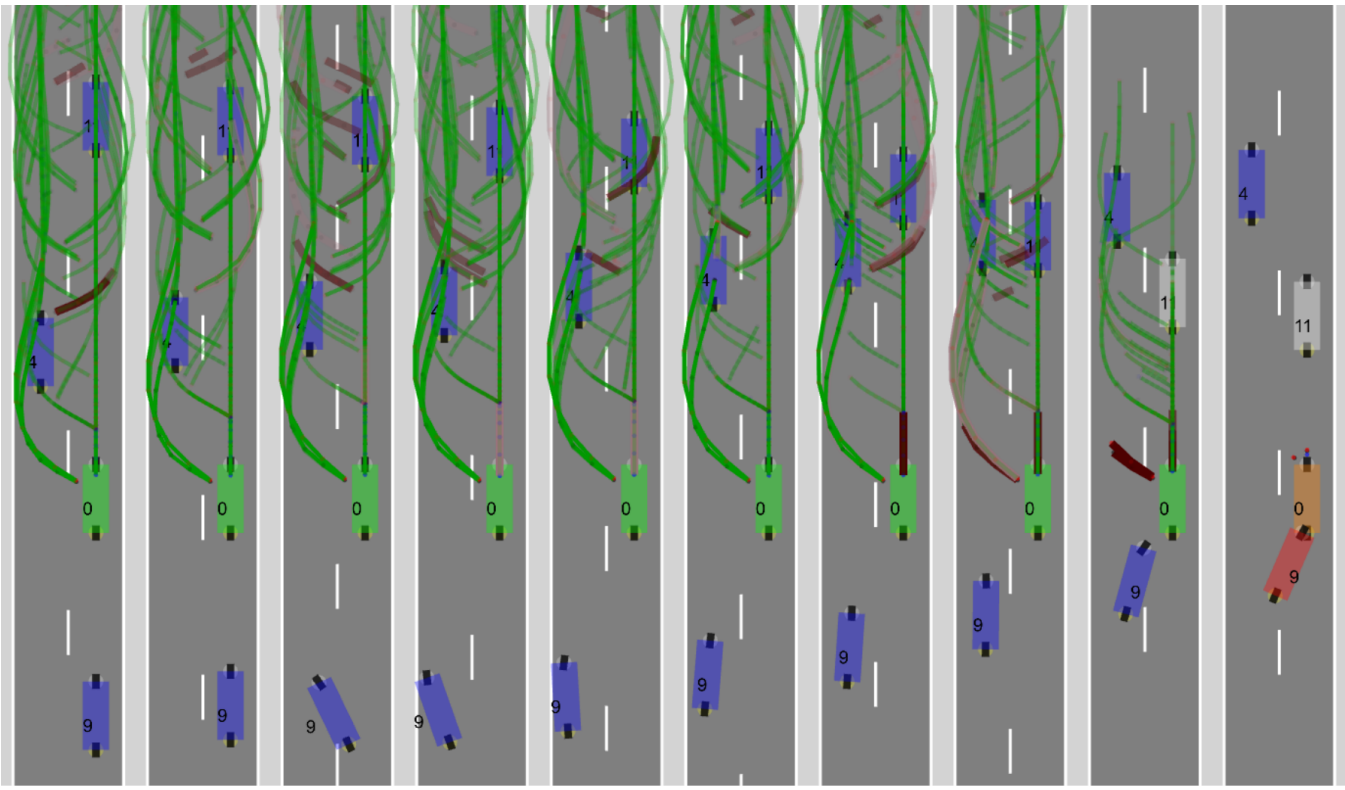

Fig. 9: MCPTDM experiencing a crash in our simulated road environment (compare to Fig. 8). Although the situation we depict here is not realistic, it illustrates the difficult and fairly random situations we are attempting to plan for. As vehicle 11 comes to a stop in front of the ego vehicle, vehicle 9 from behind starts to make an unsafe lane-change into the right lane which the ego vehicle is unable to avoid. It is possible that the ego vehicle could avoid this crash if it were using a replanning rate of faster than 4 Hz. From the forward-simulated traces in the second-to-last frame, it appears that only scenarios with vehicle 11 accelerating first manage to avoid this crash, since the ego vehicle's intelligent driver model requires it to maintain a certain following distance. Frames are left-to-right in half-second increments. A video version of this sequence is provided at `https://youtu.be/6vK-RxXwBGw`.

are the cost weights; $d_{\min}$ is the minimum distance between the ego vehicle and any other vehicle; $k_{\text{safety}} = -5$ and $d_{\text{safety}} = 1$ define the shape of a logistic sigmoid used for safety penalties; and $\alpha = 0.8$ is a discount factor. Minimum distance $d_{\min}$ is calculated from the closest points between the rotated vehicle rectangles.

We note that this cost function does not penalize situations that may be considered risky from the perspective of non-ego vehicles.

A. Belief estimation

Each obstacle vehicle may or may not be intending to perform a lane-change maneuver. From the perspective of the ego agent, this is hidden state and must be estimated in order to perform a forward rollout.

For simplicity, we implement a stateless heuristic for belief estimation based on thresholds for the direction a vehicle is pointing, its position in its lane, and its velocity relative to the vehicle ahead of it. While this belief estimation leaves room for improvement, it should not affect the fairness of our method comparisons.

B. Methods

1) Multi-policy decision making (MPDM): Our classic MPDM comparison takes samples (according to the computational budget) from the belief state and closed-loop forward simulates them through each of our five policy choices for 8 seconds. Finally, the policy with the lowest mean cost is elected.

2) Efficient uncertainty-aware decision making (EUDM): EUDM is an extension to MPDM that includes both a specific tree search through the policies as well as a heuristic for selecting belief samples that represent the most important/risky cases.

The tree search used by EUDM, the "domain-specific closed-loop policy tree", allows for only one policy change in the planning horizon, and this change must happen below the root node of the tree. Just as in the original paper, we use a tree depth of 4 with each layer taking 2 seconds so that we have a total horizon of 8 seconds. As policy changes must happen after the root node, EUDM has a built-in hysteresis and will only actually change policies if it still wants to 2 seconds after first making that decision. The original authors use the current EUDM-selected policy as an input to a separate "spatio-temporal semantic corridor" trajectory generation module [44] which produces the actual behavior for the ego vehicle. This extra module allows their ego vehicle to react to changing circumstances in a risk-aware fashion even with the 2 seconds of policy hysteresis.

We modified EUDM to consider switching policies at any time, including immediately. This deviates from the original EUDM method, but we found it improves its performance in our comparison.

The heuristic used by EUDM, "conditional focused branching" (CFB), selects nearby obstacle vehicles, filters to just the vehicles whose policies we are uncertain about, then performs open-loop forward simulations of each belief policy, and finally makes a set of the most likely belief samples formed by the Cartesian product of vehicles and policies for each obstacle vehicle deemed risky by the open-loop simulations. All non-risky vehicles are assigned their most-likely policy.

In our implementation of EUDM, we perform open-loop forward simulations by giving only the ego and obstacle vehicle under examination dynamic policies, and simulate all the other vehicles with just a constant velocity. We use the same horizon of 8 seconds. We order obstacle vehicles according to their "risk", the difference between the minimum and maximum costs from each of the policy choices, and then we choose the 4 most risky vehicles. We form the Cartesian product of these risky obstacle vehicles and their policies and then finally select the most probable scenarios according to our belief, and weight them according to their probabilities. We take as many scenarios as allowed by our computational budget.

3) Monte Carlo Policy Tree Decision Making (MCPTDM): Our final method for application to the automated driving scenario uses the improvements from the synthetic experiments described above: marginal action cost (MAC) expected-cost estimation and particle repetition. We use KL-UCB and "max-robust child" selection just as in the earlier experiments. In addition, when expanding a node in the search tree, we first explore the child with the same policy as the parent, since most of the time the ego vehicle will be maintaining its current policy.

C. Experiments

We perform 16,384 runs for each method in order to get significant results. To complete all these runs in a reasonable time frame, we limit each run to 30 simulated seconds and replan at 4 Hz. Runs are performed on a 2.5 GHz Intel Xeon E5-2640 with a single thread for each run so that multiple runs can be performed in parallel.

To make fair comparisons, we plot the final cost observed in each run (with no discount) against the 95% computational latency. That is, 95% of the replanning periods of a 30 second run will have a latency less than this. The closer a method is to the bottom left corner (closer to zero cost and zero time), the better.

Over a total of 589,824 30-second long runs to make all of these figures, there were a total of 2,573 crashes (0.44%). Because crashes are both relatively rare and also lead to much higher final costs, the error bars are relatively large in some of the figures.

First we perform an ablation to see the separate contributions of MACs and particle repetition. We performed a parameter sweep to determine reasonable constants for KL-UCB (for the full method) and for particle repetition, and found that it did best when repeating as much as possible. We find (see Fig. 10) that both MACs and particle repetition result in significant improvements.

Finally we perform a comparison of our MCPTDM method to our baseline methods of MPDM and EUDM in Fig. 11. MCPTDM achieves significantly lower cost for similar computational time. Our cost function is composed of a safety cost (for avoiding crashes and being too close), an efficiency cost (for being close to a target velocity), and steering and acceleration costs (for minimizing control inputs), where the safety and efficiency are the most significant. We compare just the safety and efficiency costs in figures 12 and 13, and note that the majority of the lower cost improvements of MCPTDM against MPDM and EUDM come from keeping efficiency without sacrificing safety. Intuitively it makes sense that in most cases increased safety (lower safety cost) comes with worse efficiency (lower average velocity and a higher efficiency cost).

We notice that the EUDM's CFB heuristic is only an improvement in our self-driving scenario at the point of highest computational cost, even though it shares significant similarities with the scenarios used in the EUDM paper. This may be a consequence of omitting the separate trajectory optimizer from their paper, of implementing CFB in a slightly different way, or of another factor. In addition, we are surprised that vanilla MPDM is so competitive with EUDM because MPDM is not able to forward simulate policy changes like EUDM or MCPTDM.

We also note that our absolute (in figures 10 through 13) and qualitative results (in figures 8 and 9) are highly dependent on the specific closed-loop policy, cost function, and problem domain/vehicle modeling choices. We make a fair comparison to MPDM and EUDM by using identical components for each of these choices. As MCPTDM does not provide tools for selecting or designing these components, they must be carefully provided by some other method for a practical application. MCPTDM is most appropriate when a set of closed-loop policies can be effectively designed for the problem domain.

V. Conclusion

In this paper we have presented Monte-Carlo Policy-Tree Decision Making (MCPTDM), an MCTS-based search framework for problems where the marginal costs of each action or policy election are both available and important, such as autonomous vehicle planning. We have presented several novel ideas including the use of marginal action costs and fairness-based particle repetition. We have validated these improvements on both a simplified abstract task and also an autonomous vehicle planning task. We have demonstrated that MCPTDM has better performance than both MPDM and EUDM. The code used to generate all the results in this paper can be found at either `https://osf.io/pguhz/` or `https://github.com/acshi/MCPTDM` to aid in replicating and extending this work.

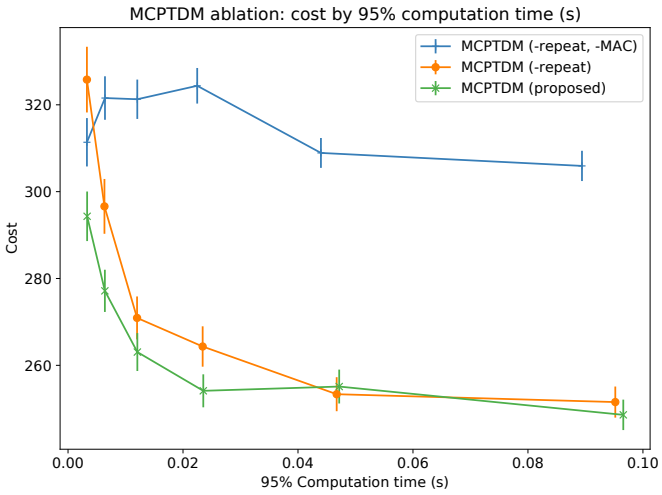

Fig. 10: We perform an ablation of MCPTDM by evaluating it without particle repetition and then also with "classic" expected-cost estimation instead of marginal action costs. We see that both improvements are significant.

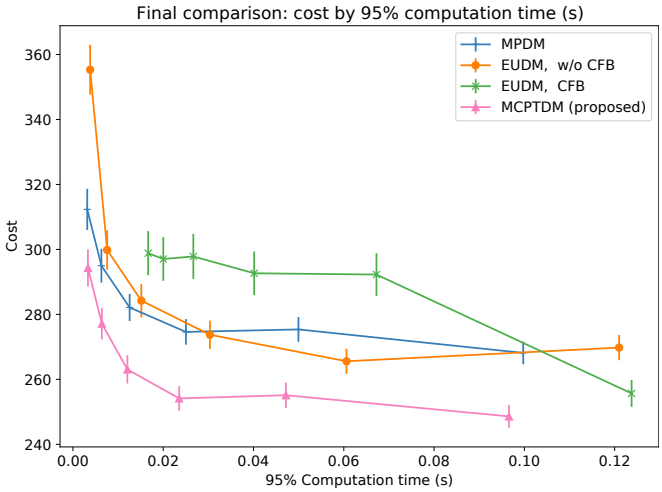

Fig. 11: Final comparison of MCPTDM with EUDM (both with and without the CFB heuristic) and MPDM. MCPTDM achieves either significantly lower final cost or significantly lower computational time than either EUDM or MPDM.

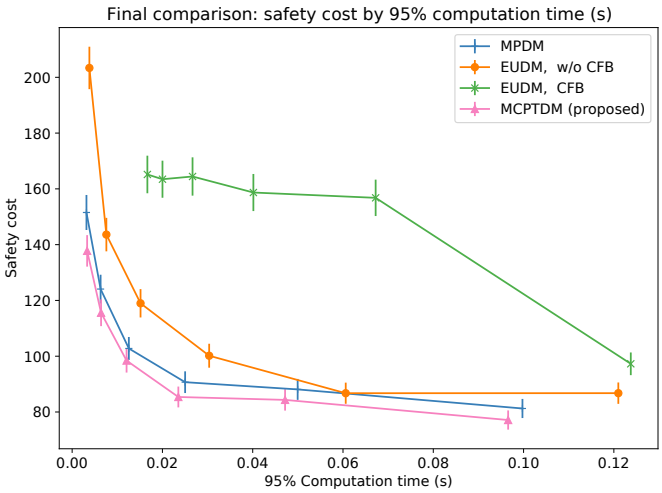

Fig. 12: Comparison of just the final safety cost (lower is better) between each method. At all computation times, MPDM is only slightly less safe that MCPTDM. For larger computation times, EUDM is also very similar. Compare with Fig. 13 and the plot of just the efficiency cost.

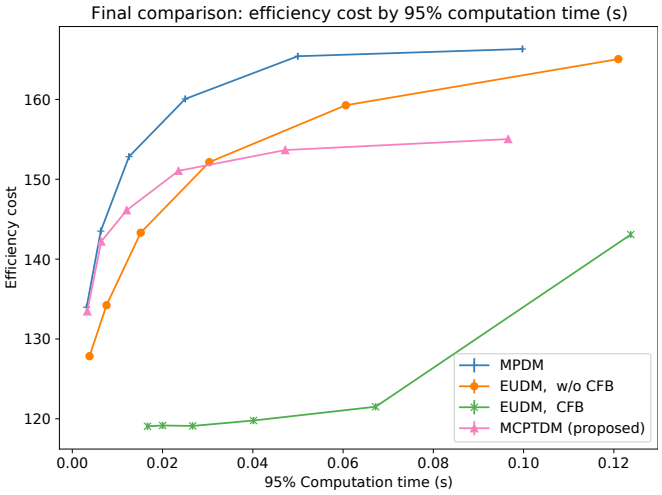

Fig. 13: Comparison of just the final efficiency cost (lower is better) between each method. MCPTDM is quick to worsen efficiency (for better safety) and also keeps the efficiency cost relatively low as the computational budget increases. Compare with Fig. 12 and the plot of just the safety cost.

A. Acknowledgements

This work was supported by grants from the NSF (1830615).

Disclosure: Edwin Olson has a financial interest in a company that may have rights to foreground or background technology described in this paper.

References

[1] K. J. Astrom, "Optimal control of Markov decision processes with incomplete state estimation," *J. Math. Anal. Applic.*, vol. 10, pp. 174–205, 1965.

[2] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable Markov processes over a finite horizon," *Operations research*, vol. 21, no. 5, pp. 1071–1088, 1973.

[3] G. A. Hollinger, "Long-horizon robotic search and classification using sampling-based motion planning." in *Robotics: Science and Systems*, vol. 3, 2015.

[4] T. Jaakkola, S. P. Singh, and M. I. Jordan, "Reinforcement learning algorithm for partially observable Markov decision problems," *Advances in neural information processing systems*, pp. 345–352, 1995.

[5] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Advances in neural information processing systems*, 2010, pp. 2164–2172.

[6] K. Hsiao, L. P. Kaelbling, and T. Lozano-Perez, "Grasping POMDPs," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 4685–4692.

[7] A. G. Cunningham, E. Galceran, R. M. Eustice, and E. Olson, "MPDM: Multipolicy decision-making in dynamic, uncertain environments for autonomous driving," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, June 2015.

[8] L. Zhang, W. Ding, J. Chen, and S. Shen, "Efficient uncertainty-aware decision-making for automated driving using guided branching," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 3291–3297.

[9] M. L. Littman, "A tutorial on partially observable Markov decision processes," *Journal of Mathematical Psychology*, vol. 53, no. 3, pp. 119–125, 2009.

[10] G. E. Monahan, "State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms," *Management science*, vol. 28, no. 1, pp. 1–16, 1982.

[11] J. Satia and R. Lave, "Markovian decision processes with probabilistic observation of states," *Management Science*, vol. 20, no. 1, pp. 1–13, 1973.

[12] A. McGovern, R. S. Sutton, and A. H. Fagg, "Roles of macro-actions in accelerating reinforcement learning," in *Grace Hopper celebration of women in computing*, vol. 1317, 1997, p. 15.

[13] G. Theocharous and L. P. Kaelbling, "Approximate planning in POMDPs with macro-actions," in *Advances in Neural Information Processing Systems*, 2004, pp. 775–782.

[14] R. He, E. Brunskill, and N. Roy, "PUMA: Planning under uncertainty with macro-actions." in *AAAI*, 2010, p. 7.

[15] J. Van Den Berg, P. Abbeel, and K. Goldberg, "LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 895–913, 2011.

[16] A.-A. Agha-Mohammadi, S. Chakravorty, and N. M. Amato, "FIRM: Feedback controller-based information-state roadmap–a framework for motion planning under uncertainty," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 4284–4291.

[17] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 723–730.

[18] F. Doshi-Velez, "The infinite partially observable Markov decision process," *Advances in neural information processing systems*, vol. 22, pp. 477–485, 2009.

[19] M. J. Beal, Z. Ghahramani, and C. E. Rasmussen, "The infinite hidden Markov model," *Advances in neural information processing systems*, vol. 1, pp. 577–584, 2002.

[20] J. Van Gael, Y. Saatci, Y. W. Teh, and Z. Ghahramani, "Beam sampling for the infinite hidden Markov model," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1088–1095.

[21] C. Amato and F. Oliehoek, "Scalable planning and learning for multiagent POMDPs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[22] C. Guestrin, D. Koller, and R. Parr, "Multiagent planning with factored MDPs," in *NIPS*, vol. 1, 2001, pp. 1523–1530.

[23] G. Best, O. M. Cliff, T. Patten, R. R. Mettu, and R. Fitch, "Decentralised Monte Carlo tree search for active perception," in *WAFR*, 2016.

[24] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[25] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "A0c: Alpha zero in continuous action space," *arXiv preprint arXiv:1805.09613*, 2018.

[26] W. Ding, J. Chen, and S. Shen, "Predicting vehicle behaviors over an extended horizon using behavior interaction network," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8634–8640.

[27] C. Paxton, V. Raman, G. D. Hager, and M. Kobilarov, "Combining neural networks and tree search for task and motion planning in challenging environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6059–6066.

[28] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, "Tactical decision making for lane changing with deep reinforcement learning," in *NIPS 2017 Workshop on Machine Learning for Intelligent Transportation Systems*, 12 2017.

[29] E. Galceran, A. G. Cunningham, R. M. Eustice, and E. Olson, "Multi-policy decision-making for autonomous driving via changepoint-based behavior prediction," in *Proceedings of Robotics: Science and Systems (RSS)*, Rome, Italy, July 2015.

[30] D. Mehta, G. Ferrer, and E. Olson, "Fast discovery of influential outcomes for risk-aware MPDM," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2017.

[31] ——, "Backprop-MPDM: Faster risk-aware policy evaluation through efficient gradient optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[32] B. Zhou, W. Schwarting, D. Rus, and J. Alonso-Mora, "Joint multi-policy behavior estimation and receding-horizon trajectory planning for automated urban driving," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2388–2394.

[33] R. Morton and E. Olson, "Robust sensor characterization via max-mixture models: GPS sensors," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, November 2013.

[34] R. W. Wolcott and R. M. Eustice, "Robust LIDAR localization using multiresolution Gaussian mixture maps for autonomous driving," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 292–319, 2017.

[35] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *International conference on computers and games*. Springer, 2006, pp. 72–83.

[36] G. Chaslot, M. Winands, H. Herik, J. Uiterwijk, and B. Bouzy, "Progressive strategies for Monte-Carlo tree search," *New Mathematics and Natural Computation*, vol. 04, pp. 343–357, 11 2008.

[37] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[38] D. Michie and R. A. Chambers, "BOXES: An experiment in adaptive control," *Machine intelligence*, vol. 2, no. 2, pp. 137–152, 1968.

[39] J.-Y. Audibert, R. Munos, and C. Szepesvári, "Exploration–exploitation tradeoff using variance estimates in multi-armed bandits," *Theoretical Computer Science*, vol. 410, no. 19, pp. 1876–1902, 2009.

[40] Y. Abbasi-Yadkori, D. Pál, and C. Szepesvári, "Improved algorithms for linear stochastic bandits," *Advances in neural information processing systems*, vol. 24, pp. 2312–2320, 2011.

[41] A. Garivier and O. Cappé, "The KL-UCB algorithm for bounded stochastic bandits and beyond," in *Proceedings of the 24th annual conference on learning theory*. JMLR Workshop and Conference Proceedings, 2011, pp. 359–376.

[42] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[43] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992.

[44] W. Ding, L. Zhang, J. Chen, and S. Shen, "Safe trajectory generation for complex urban environments using spatio-temporal semantic corridor," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2997–3004, 2019.