CMAX++ : Leveraging Experience in Planning and Execution using Inaccurate Models

Anirudh Vemula[1], **J. Andrew Bagnell**[2], **Maxim Likhachev**[1]
[1] Robotics Institute, Carnegie Mellon University
[2] Aurora Innovation
vemula@cmu.edu, dbagnell@ri.cmu.edu, maxim@cs.cmu.edu

Abstract

Given access to accurate dynamical models, modern planning approaches are effective in computing feasible and optimal plans for repetitive robotic tasks. However, it is difficult to model the true dynamics of the real world before execution, especially for tasks requiring interactions with objects whose parameters are unknown. A recent planning approach, CMAX, tackles this problem by adapting the planner online during execution to bias the resulting plans away from inaccurately modeled regions. CMAX, while being provably guaranteed to reach the goal, requires strong assumptions on the accuracy of the model used for planning and fails to improve the quality of the solution over repetitions of the same task. In this paper we propose CMAX++, an approach that leverages real-world experience to improve the quality of resulting plans over successive repetitions of a robotic task. CMAX++ achieves this by integrating model-free learning using acquired experience with model-based planning using the potentially inaccurate model. We provide provable guarantees on the completeness and asymptotic convergence of CMAX++ to the optimal path cost as the number of repetitions increases. CMAX++ is also shown to outperform baselines in simulated robotic tasks including 3D mobile robot navigation where the track friction is incorrectly modeled, and a 7D pick-and-place task where the mass of the object is unknown leading to discrepancy between true and modeled dynamics.

Introduction

We often require robots to perform tasks that are highly repetitive, such as picking and placing objects in assembly tasks and navigating between locations in a warehouse. For such tasks, robotic planning algorithms have been highly effective in cases where system dynamics is easily specified by an efficient forward model (Berenson, Abbeel, and Goldberg 2012). However, for tasks involving interactions with objects, dynamics are very difficult to model without complete knowledge of the parameters of the objects such as mass and friction (Ji and Xiao 2001). Using inaccurate models for planning can result in plans that are ineffective and fail to complete the task (McConachie et al. 2020). In addition for such repetitive tasks, we expect the robot's task performance to improve, leading to efficient plans in later repetitions. Thus, we need a planning approach that can use potentially inaccurate models while leveraging experience from past executions to complete the task in each repetition, and improve performance across repetitions.

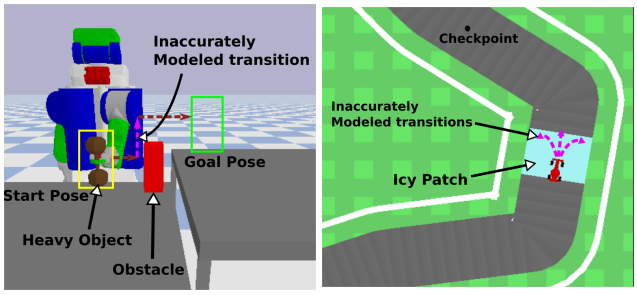

Figure 1: (left) PR2 lifting a heavy dumbbell, that is modeled as light, to a goal location that is higher than the start location resulting in dynamics that are inaccurately modeled (right) Mobile robot navigating around a track with icy patches with unknown friction parameters leading to the robot skidding. In both cases, any path to the goal needs to contain a transition (pink) whose dynamics are not modeled accurately.

A recent planning approach, CMAX, introduced in (Vemula et al. 2020) adapts its planning strategy online to account for any inaccuracies in the forward model without requiring any updates to the dynamics of the model. CMAX achieves this online by inflating the cost of any transition that is found to be incorrectly modeled and replanning, thus biasing the resulting plans away from regions where the model is inaccurate. It does so while maintaining guarantees on completing the task, without any resets, in a finite number of executions. However, CMAX requires that there always exists a path from the current state of the robot to the goal containing only transitions that have not yet been found to be incorrectly modeled. This is a strong assumption on the accuracy of the model and can often be violated, especially in the context of repetitive tasks.

For example, consider the task shown in Figure 1(left) where a robotic arm needs to repeatedly pick a heavy object, that is incorrectly modeled as light, and place it on top of a taller table while avoiding an obstacle. As the object

is heavy, transitions that involve lifting the object will have discrepancy between true and modeled dynamics. However, any path from the start pose to the goal pose requires lifting the object and thus, the resulting plan needs to contain a transition that is incorrectly modeled. This violates the aforementioned assumption of CMAX and it ends up inflating the cost of any transition that lifts the object, resulting in plans that avoid lifting the object in future repetitions. Thus, the quality of CMAX solution deteriorates across repetitions and, in some cases, it even fails to complete the task. Figure 1(right) presents another example task where a mobile robot is navigating around a track with icy patches that have unknown friction parameters. Once the robot enters a patch, any action executed results in the robot skidding, thus violating the assumption of CMAX because any path to the goal from current state will have inaccurately modeled transitions. CMAX ends up inflating the cost of all actions executed inside the icy patch, leading to the robot being unable to find a path in future laps and failing to complete the task. Thus, in both examples, we need a planning approach that allows solutions to contain incorrectly modeled transitions while ensuring that the robot reaches the goal.

In this paper we present CMAX++, an approach for interleaving planning and execution that uses inaccurate models and leverages experience from past executions to provably complete the task in each repetition without any resets. Furthermore, it improves the quality of solution across repetitions. In contrast to CMAX, CMAX++ requires weaker conditions to ensure task completeness, and is provably guaranteed to converge to a plan with optimal cost as the number of repetitions increases. The key idea behind CMAX++ is to combine the conservative behavior of CMAX that tries to avoid incorrectly modeled regions with model-free Q-learning that tries to estimate and follow the optimal cost-to-goal value function with no regard for any discrepancies between modeled and true dynamics. This enables CMAX++ to compute plans that utilize inaccurately modeled transitions, unlike CMAX. Based on this idea, we present an algorithm for small state spaces, where we can do exact planning, and a practical algorithm for large state spaces using function approximation techniques. We also propose an adaptive version of CMAX++ that intelligently switches between CMAX and CMAX++ to combine the advantages of both approaches, and exhibits goal-driven behavior in earlier repetitions and optimality in later repetitions. The proposed algorithms are tested on simulated robotic tasks: 3D mobile robot navigation where the track friction is incorrectly modeled (Figure 1 right) and a 7D pick-and-place task where the mass of the object is unknown (Figure 1 left).

Related Work

A typical approach to planning in tasks with unknown parameters is to use acquired experience from executions to update the dynamics of the model and replan (Sutton 1991). This works well in practice for tasks where the forward model is flexible and can be updated efficiently. However for real world tasks, the models used for planning cannot be updated efficiently online (Todorov, Erez, and Tassa 2012) and are often precomputed offline using expensive procedures (Hauser et al. 2006). Approaches, such as (Ramos, Possas, and Fox 2019), that estimate parameters of the dynamical model online can fail in cases where the uncertainty cannot be accounted by any of the parameters. Another line of works (Saveriano et al. 2017; Abbeel, Quigley, and Ng 2006) seek to learn a residual dynamical model to account for the inaccuracies in the initial model. However, it can take a prohibitively large number of executions to learn the true dynamics, especially in domains like deformable manipulation (Essahbi, Bouzgarrou, and Gogu 2012). This precludes these approaches from demonstrating a goal-driven behavior as we show in our experimental analysis.

Recent works such as CMAX (Vemula et al. 2020) and (McConachie et al. 2020) pursue an alternative approach which does not require updating the dynamics of the model or learning a residual component. These approaches exhibit goal-driven behavior by focusing on completing the task and not on modeling the true dynamics accurately. While CMAX achieves this by inflating the cost of any transition whose dynamics are inaccurately modeled, (McConachie et al. 2020) present an approach that learns a binary classifier offline that is used online to predict whether a transition is accurately modeled or not. Although these methods work well in practice for goal-oriented tasks, they do not leverage experience acquired online to improve the quality of solution when used for repetitive tasks.

Our work is closely related to approaches that integrate model-based planning with model-free learning. (Lee et al. 2020) use model-based planning in regions where the dynamics are accurately modeled and switch to a model-free policy in regions with high uncertainty. However, they mostly focus on perception uncertainty and require a coarse estimate of the uncertain region prior to execution, which is often not available for tasks with other modalities of uncertainty like unknown inertial parameters. A very recent work by (Lagrassa, Lee, and Kroemer 2020) uses a model-based planner until a model inaccuracy is detected and switches to a model-free policy to complete the task. Similar to our approach, they deal with general modeling errors but rely on expert demonstrations to learn the model-free policy. In contrast, our approach does not require any expert demonstrations and only uses the experience acquired online to obtain model-free value estimates that are used within planning.

Finally, our approach is also related to the field of real-time heuristic search which tackles the problem of efficient planning in large state spaces with bounded planning time. In this work, we introduce a novel planner that is inspired by LRTA* (Korf 1990) which limits the number of expansions in the search procedure and interleaves execution with planning. Crucially, our planner also interleaves planning and execution but unlike these approaches, employs model-free value estimates obtained from past experience within the search.

Problem Setup

Following the notation of (Vemula et al. 2020), we consider the deterministic shortest path problem that can be represented using the tuple $M = (\mathbb{S}, \mathbb{A}, \mathbb{G}, f, c)$ where $\mathbb{S}$ is the state space, $\mathbb{A}$ is the action space, $\mathbb{G} \subseteq \mathbb{S}$ is the non-empty

set of goals, $f : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$ is a deterministic dynamics function, and $c : \mathbb{S} \times \mathbb{A} \to \mathbb{R}^+ \cup \{0\}$ is the cost function. Crucially, our approach assumes that the action space $\mathbb{A}$ is discrete, and any goal state $g \in \mathbb{G}$ is a cost-free termination state. The objective of the shortest path problem is to find the least-cost path from a given start state $s_1 \in \mathbb{S}$ to any goal state $g \in \mathbb{G}$ in M. As is typical in shortest path problems, we assume that there exists at least one path from each state $s \in \mathbb{S}$ to one of the goal states, and that the cost of any transition from a non-goal state is positive (Bertsekas 2005). We will use $V(s)$ to denote the state value function (a running estimate of cost-to-goal from state s,) and $Q(s, a)$ to denote the state-action value function (a running estimate of the sum of transition cost and cost-to-goal from successor state,) for any state s and action a. Similarly, we will use the notation $V^*(s)$ and $Q^*(s, a)$ to denote the corresponding optimal value functions. A value estimate is called admissible if it underestimates the optimal value function at all states and actions, and is called consistent if it satisfies the triangle inequality, i.e. $V(s) \leq c(s, a) + V(f(s, a))$ and $Q(s, a) \leq c(s, a) + V(f(s, a))$, and $V(g) = 0$ for all $g \in \mathbb{G}$.

In this work, we focus on repetitive robotic tasks where the true deterministic dynamics f are unknown but we have access to an approximate model described using $\hat{M} = (\mathbb{S}, \mathbb{A}, \mathbb{G}, \hat{f}, c)$ where $\hat{f}$ approximates the true dynamics. In each repetition of the task, the robot acts in the environment M to acquire experience over a single trajectory and reach the goal, without access to any resets. This rules out any episodic approach. Since the true dynamics are unknown and can only be discovered through executions, we consider the online real-time planning setting where the robot has to interleave planning and execution. In our motivating navigation example (Figure 1 right,) the approximate model $\hat{M}$ represents a track with no icy patches whereas the environment M contains icy patches. Thus, there is a discrepancy between the modeled dynamics $\hat{f}$ and true dynamics f. Following (Vemula et al. 2020), we will refer to state-action pairs that have inaccurately modeled dynamics as "incorrect" transitions, and use the notation $\mathcal{X} \subseteq \mathbb{S} \times \mathbb{A}$ to denote the set of discovered incorrect transitions. The objective in our work is for the robot to reach a goal in each repetition, despite using an inaccurate model for planning while improving performance, measured using the cost of executions, across repetitions.

Approach

In this section, we will describe the proposed approach CMAX++. First, we will present a novel planner used in CMAX++ that can exploit incorrect transitions using their model-free Q-value estimates. Second, we present CMAX++ and its adaptive version for small state spaces, and establish their guarantees. Finally, we describe a practical instantiation of CMAX++ for large state spaces leveraging function approximation techniques.

Hybrid Limited-Expansion Search Planner

During online execution, we want the robot to acquire experience and leverage it to compute better plans. This requires a hybrid planner that is able to incorporate value estimates obtained using past experience in addition to model-based planning, and quickly compute the next action to execute. To achieve this, we propose a real-time heuristic search-based planner that performs a bounded number of expansions and is able to utilize Q-value estimates for incorrect transitions.

Algorithm 1 Hybrid Limited-Expansion Search

1: **procedure** SEARCH($s, \hat{M}, V, Q, \mathcal{X}, K$)
2: Initialize $g(s) = 0$, min-priority open list O, and closed list C
3: Add s to open list O with priority $p(s) = g(s) + V(s)$
4: **for** $i = 1, 2, \cdots, K$ **do**
5: Pop s_i from O
6: **if** s_i is a dummy state or $s_i \in \mathbb{G}$ **then**
7: Set $s_{\text{best}} \leftarrow s_i$ and go to Line 22
8: **for** $a \in \mathbb{A}$ **do** ▷ *Expanding state* s_i
9: **if** $(s_i, a) \in \mathcal{X}$ **then** ▷ *Incorrect transition*
10: Add a dummy state s' to O with priority $p(s') = g(s_i) + Q(s_i, a)$
11: **continue**
12: Get successor $s' = \hat{f}(s_i, a)$
13: If $s' \in C$, **continue**
14: **if** $s' \in O$ and $g(s') > g(s_i) + c(s_i, a)$ **then**
15: Set $g(s') = g(s_i) + c(s_i, a)$ and recompute $p(s')$
16: Reorder open list O
17: **else if** $s' \notin O$ **then**
18: Set $g(s') = g(s_i) + c(s_i, a)$
19: Add s' to O with priority $p(s') = g(s') + V(s')$
20: Add s_i to closed list C
21: Pop s_{best} from open list O
22: **for** $s' \in C$ **do**
23: Update $V(s') \leftarrow p(s_{\text{best}}) - g(s')$
24: Backtrack from s_{best} to s, and set a_{best} as the first action on path from s to s_{best} in the search tree
return a_{best}

The planner is presented in Algorithm 1. Given the current state s, the planner constructs a lookahead search tree using at most K state expansions. For each expanded state s_i, if any outgoing transition has been flagged as incorrect based on experience, i.e. $(s_i, a) \in \mathcal{X}$, then the planner creates a dummy state with priority computed using the model-free Q-value estimate of that transition (Line 10). Note that we create a dummy state because the model $\hat{M}$ does not know the true successor of an incorrect transition. For the transitions that are correct, we obtain successor states using the approximate model $\hat{M}$. This ensures that we rely on the inaccurate model only for transitions that are not known to be incorrect. At any stage, if a dummy state is expanded then we need to terminate the search as the model $\hat{M}$ does not know any of its successors, in which case we set the best state s_{best} as the dummy state (Line 7). Otherwise, we choose s_{best} as the best state (lowest priority) among the leaves of the search tree after K expansions (Line 21). Finally, the best action to execute at the current state s is computed as the first action along the path from s to s_{best}

Algorithm 2 CMAX++ (regular text) and A-CMAX++ (regular and *italics* text) in small state spaces

Require: Model $\hat{M}$, start state s, initial value estimates V, Q, number of expansions K, $t \leftarrow 1$, incorrect set $\mathcal{X} \leftarrow \{\}$, Number of repetitions N, *Sequence* $\{\alpha_i \geq 1\}_{i=1}^{N}$, *initial penalized value estimates* $\tilde{V} = V$, *penalized model* $\tilde{M} \leftarrow \hat{M}$

1: **for** each repetition $i = 1, \cdots, N$ **do**
2: $t \leftarrow 1$, $s_1 \leftarrow s$
3: **while** $s_t \notin \mathbb{G}$ **do**
4: Compute $a_t = \texttt{SEARCH}(s_t, \hat{M}, V, Q, \mathcal{X}, K)$
5: *Compute* $\tilde{a}_t = \texttt{SEARCH}(s_t, \tilde{M}, \tilde{V}, Q, \{\}, K)$
6: *If* $\tilde{V}(s_t) \leq \alpha_i V(s_t)$, *assign* $a_t = \tilde{a}_t$
7: Execute a_t in environment to get $s_{t+1} = f(s_t, a_t)$
8: **if** $s_{t+1} \neq \hat{f}(s_t, a_t)$ **then**
9: Add (s_t, a_t) to the set: $\mathcal{X} \leftarrow \mathcal{X} \cup \{(s_t, a_t)\}$
10: Update: $Q(s_t, a_t) = c(s_t, a_t) + V(s_{t+1})$
11: *Update penalized model* $\tilde{M} \leftarrow \tilde{M}_{\mathcal{X}}$
12: $t \leftarrow t + 1$

in the search tree (Line 24). The planner also updates state value estimates V of all expanded states using the priority of the best state $p(s_{\text{best}})$ to make the estimates more accurate (Lines 22 and 23) similar to RTAA* (Koenig and Likhachev 2006).

The ability of our planner to exploit incorrect transitions using their model-free Q-value estimates, obtained from past experience, distinguishes it from real-time search-based planners such as LRTA* (Korf 1990) which cannot utilize model-free value estimates during planning. This enables CMAX++ to result in plans that utilize incorrect transitions if they enable the robot to get to the goal with lower cost.

CMAX++ in Small State Spaces

CMAX++ in small state spaces is simple and easy-to-implement as it is feasible to maintain value estimates in a table for all states and actions and to explicitly maintain a running set of incorrect transitions with fast lookup without resorting to function approximation techniques.

The algorithm is presented in Algorithm 2 (only the regular text.) CMAX++ maintains a running estimate of the set of incorrect transitions $\mathcal{X}$, and updates the set whenever it encounters an incorrect state-action pair during execution. Crucially, unlike CMAX, it maintains a Q-value estimate for the incorrect transition that is used during planning in Algorithm 1, thereby enabling the planner to compute paths that contain incorrect transitions. It is also important to note that, like CMAX, CMAX++ never updates the dynamics of the model. However, instead of using the penalized model for planning as CMAX does, CMAX++ uses the initial model $\hat{M}$, and utilizes both model-based planning and model-free Q-value estimates to replan a path from the current state to a goal.

The downside of CMAX++ is that estimating Q-values from online executions can be inefficient as it might take many executions before we obtain an accurate Q-value estimate for an incorrect transition. This has been extensively studied in the past and is a major disadvantage of model-free methods (Sun et al. 2019). As a result of this inefficiency, CMAX++ lacks the goal-driven behavior of CMAX in early repetitions of the task, despite achieving optimal behavior in later repetitions. In the next section, we present an adaptive version of CMAX++ (A-CMAX++) that combines the goal-driven behavior of CMAX with the optimality of CMAX++.

Adaptive Version of CMAX++

Background on CMAX Before we describe A-CMAX++, we will start by summarizing CMAX. For more details, refer to (Vemula et al. 2020). At each time step t during execution, CMAX maintains a running estimate of the incorrect set $\mathcal{X}$, and constructs a penalized model specified by the tuple $\tilde{M}_{\mathcal{X}} = (\mathbb{S}, \mathbb{A}, \mathbb{G}, \hat{f}, \tilde{c}_{\mathcal{X}})$ where the cost function $\tilde{c}_{\mathcal{X}}(s, a) = |\mathbb{S}|$ if $(s, a) \in \mathcal{X}$, else $\tilde{c}_{\mathcal{X}}(s, a) = c(s, a)$. In other words, the cost of any transition found to be incorrect is set high (or inflated) while the cost of other transitions are the same as in $\hat{M}$. CMAX uses the penalized model $\tilde{M}_{\mathcal{X}}$ to plan a path from the current state s_t to a goal state. Subsequently, CMAX executes the first action a_t along the path and observes if the true dynamics and model dynamics differ on the executed action. If so, the state-action pair (s_t, a_t) is appended to the incorrect set $\mathcal{X}$ and the penalized model $\tilde{M}_{\mathcal{X}}$ is updated. CMAX continues to do this at every timestep until the robot reaches a goal state.

Observe that the inflation of cost for any incorrect state-action pair biases the planner to "explore" all other state-action pairs that are not yet known to be incorrect before it plans a path using an incorrect transition. This induces a goal-driven behavior in the computed plan that enables CMAX to quickly find an alternative path and not waste executions learning the true dynamics

A-CMAX++ A-CMAX++ is presented in Algorithm 2 (regular and *italics* text.) A-CMAX++ maintains a running estimate of incorrect set $\mathcal{X}$ and constructs the penalized model $\tilde{M}$ at each time step t, similar to CMAX. For any state at time step t, we first compute the best action a_t based on the approximate model $\hat{M}$ and the model-free Q-value estimates (Line 4.) In addition, we also compute the best action $\tilde{a}_t$ using the penalized model $\tilde{M}$, similar to CMAX, that inflates the cost of any incorrect transition (Line 5.) The crucial step in A-CMAX++ is Line 6 where we compare the penalized value $\tilde{V}(s_t)$ (obtained using penalized model $\tilde{M}$) and the non-penalized value $V(s_t)$ (obtained using approximate model $\hat{M}$ and Q-value estimates.) Given a sequence $\{\alpha_i \geq 1\}$ for repetitions $i = 1, \cdots, N$ of the task, if $\tilde{V}(s_t) \leq \alpha_i V(s_t)$, then we execute action $\tilde{a}_t$, else we execute a_t. This implies that if the cost incurred by following CMAX actions in the future is within α_i times the cost incurred by following CMAX++ actions, then we prefer to execute CMAX.

If the sequence $\{\alpha_i\}$ is chosen to be non-increasing such that $\alpha_1 \geq \alpha_2 \cdots \geq \alpha_N \geq 1$, then we can observe that A-CMAX++ has the desired anytime-like behavior. It remains goal-driven in early repetitions, by choosing CMAX

actions, and converges to optimal behavior in later repetitions, by choosing CMAX++ actions. Further, the executions needed to obtain accurate Q-value estimates is distributed across repetitions ensuring that A-CMAX++ does not have poor performance in any single repetition. Thus, A-CMAX++ combines the advantages of both CMAX and CMAX++.

Theoretical Guarantees

We will start with formally stating the assumption needed by CMAX to ensure completeness:

Assumption 0.1 ((Vemula et al. 2020))**.** *Given a penalized model $\tilde{M}_{\mathcal{X}_t}$ and the current state s_t at any time step t, there always exists at least one path from s_t to a goal that does not contain any state-action pairs (s, a) that are known to be incorrect, i.e. $(s, a) \in \mathcal{X}_t$.*

Observe that the above assumption needs to be valid at every time step t before the robot reaches a goal and thus, can be hard to satisfy. Before we state the theoretical guarantees for CMAX++, we need the following assumption on the approximate model $\hat{M}$ that is used for planning:

Assumption 0.2. *The optimal value function $\hat{V}^*$ using the dynamics of approximate model $\hat{M}$ underestimates the optimal value function V^* using the true dynamics of M at all states, i.e. $\hat{V}^*(s) \leq V^*(s)$ for all $s \in \mathbb{S}$.*

In other words, if there exists a path from any state s to a goal state in the environment M, then there exists a path with the same or lower cost from s to a goal in the approximate model $\hat{M}$. In our motivating example of pick-and-place (Figure 1 left,) this assumption is satisfied if the object is modeled as light in $\hat{M}$, as the object being heavy in reality can only increase the cost. This assumption was also considered in previous works such as (Jiang 2018) and is known as the *Optimistic Model Assumption*. More discussion on this assumption is given in Appendix C. We can now state the following guarantees:

Theorem 0.1 (Completeness)**.** *Assume the initial value estimates V, Q are admissible and consistent. Then we have,*

1. *If Assumption 0.2 holds then using either* CMAX++ *or* A-CMAX++*, the robot is guaranteed to reach a goal state in at most $|\mathbb{S}|^3$ time steps in each repetition.*
2. *If Assumption 0.1 holds then (a) using* A-CMAX++ *with a large enough α_i in any repetition i (typically true for early repetitions - more details in Appendix B) the robot is guaranteed to reach a goal state in at most $|\mathbb{S}|^2$ time steps, and (b) using* CMAX++*, it is guaranteed to reach a goal state in at most $|\mathbb{S}|^3$ time steps in each repetition*

Proof Sketch. The first part of theorem follows from the analysis of Q-learning for systems with deterministic dynamics (Koenig and Simmons 1993). In the worst case, if the model is incorrect everywhere and if Assumption 0.2 (or Assumption 0.1) holds then, Algorithm 2 reduces to Q-learning, and hence we can borrow its worst case bounds. The second part of the theorem concerning A-CMAX++ follows from the completeness proof of CMAX. □

Algorithm 3 CMAX++ in large state spaces

Require: Model $\hat{M}$, start state s, value function approximators V_θ, Q_ζ, number of expansions K, $t \leftarrow 1$, Discrepancy threshold ξ, Radius of hypersphere δ, Set of hyperspheres $\mathcal{X}^\xi \leftarrow \{\}$, Number of repetitions N, Batch size B, State buffer $\mathcal{D}_S$, Transition buffer $\mathcal{D}_{SA}$, Learning rate η, Number of updates U

for each repetition $i = 1, \cdots, N$ **do**
  $t \leftarrow 1$, $s_1 \leftarrow s$
  while $s_t \notin \mathbb{G}$ **do**
    Compute $a_t = \texttt{SEARCH}(s_t, \hat{M}, V_\theta, Q_\zeta, \mathcal{X}^\xi, K)$
    Execute a_t in environment to get $s_{t+1} = f(s_t, a_t)$
    if $d(s_{t+1}, \hat{f}(s_t, a_t)) > \xi$ **then**
      Add hypersphere: $\mathcal{X}^\xi \leftarrow \mathcal{X}^\xi \cup \{\texttt{sphere}(s_t, a_t, \delta)\}$
    Add s_t to $\mathcal{D}_S$, and (s_t, a_t, s_{t+1}) to $\mathcal{D}_{SA}$
    for $u = 1, \cdots, U$ **do** ▷ *Approximator updates*
      $\texttt{Q\_UPDATE}(Q_\zeta, V_\theta, \mathcal{D}_{SA})$
      $\texttt{V\_UPDATE}(V_\theta, Q_\zeta, \mathcal{D}_S, \mathcal{X}^\xi)$
    $t \leftarrow t + 1$
procedure $\texttt{Q\_UPDATE}(Q_\zeta, V_\theta, \mathcal{D}_{SA})$
  Sample B transitions from $\mathcal{D}_{SA}$ with replacement
  Construct training set $\mathbb{X}_Q = \{((s_i, a_i), Q(s_i, a_i))\}$ for each sampled transition (s_i, a_i, s_i') and compute $Q(s_i, a_i) = c(s_i, a_i) + V_\theta(s_i')$
  Update: $\zeta \leftarrow \zeta - \eta \nabla_\zeta \mathcal{L}_Q(Q_\zeta, \mathbb{X}_Q)$
procedure $\texttt{V\_UPDATE}(V_\theta, Q_\zeta, \mathcal{D}_S, \mathcal{X}^\xi)$
  Sample B states from $\mathcal{D}_S$ with replacement
  Call $\texttt{SEARCH}(s_i, \hat{M}, V_\theta, Q_\zeta, \mathcal{X}^\xi, K)$ for each sampled s_i to get all states on closed list s_i' and their corresponding value updates $V(s_i')$ to construct training set $\mathbb{X}_V = \{(s_i', V(s_i')\}$
  Update: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_V(V_\theta, \mathbb{X}_V)$

Theorem 0.2 (Asymptotic Convergence)**.** *Assume Assumption 0.2 holds, and that the initial value estimates V, Q are admissible and consistent. For sufficiently large number of repetitions N, there exists an integer $j \leq N$ such that the robot follows a path with the optimal cost to the goal using* CMAX++ *in Algorithm 2 in repetitions $i \geq j$.*

Proof Sketch. The guarantee follows from the asymptotic convergence of Q-learning (Koenig and Simmons 1993). □

It is important to note that the conditions required for Theorem 0.1 are weaker than the conditions required for completeness of CMAX. Firstly, if either Assumption 0.1 or Assumption 0.2 holds then CMAX++ can be shown to be complete, but CMAX is guaranteed to be complete only under Assumption 0.1. Furthermore, Assumption 0.2 only needs to hold for the approximate model $\hat{M}$ we start with, whereas Assumption 0.1 needs to be satisfied for every penalized model $\tilde{M}$ constructed at any time step t during execution.

Large State Spaces

In this section, we present a practical instantiation of CMAX++ for large state spaces where it is infeasible to maintain tabular value estimates and the incorrect set $\mathcal{X}$

explicitly. Thus, we leverage function approximation techniques to maintain these estimates. Assume that there exists a metric d under which $\mathbb{S}$ is bounded. We relax the definition of incorrect set using this metric to define $\mathcal{X}^{\xi}$ as the set of all (s, a) pairs such that $d(f(s,a), \hat{f}(s,a)) > \xi$ where $\xi \geq 0$. Typically, we chose ξ to allow for small modeling discrepancies that can be compensated by a low-level path following controller.

CMAX++ in large state spaces is presented in Algorithm 3. The algorithm closely follows CMAX for large state spaces presented in (Vemula et al. 2020). The incorrect set $\mathcal{X}^{\xi}$ is maintained using sets of hyperspheres with each set corresponding to a discrete action. Whenever the agent executes an incorrect state-action (s, a), CMAX++ adds a hypersphere centered at s with radius δ, as measured using metric d, to the incorrect set corresponding to action a. In future planning, any state-action pair (s', a') is declared incorrect if s' lies inside any of the hyperspheres in the incorrect set corresponding to action a'. After each execution, CMAX++ proceeds to update the value function approximators (Line 9) by sampling previously executed transitions and visited states from buffers and performing gradient descent steps (Procedures 13 and 17) using mean squared loss functions given by $\mathcal{L}_Q(Q_\zeta, \mathbb{X}_Q) = \frac{1}{2|\mathbb{X}_Q|} \sum_{(s_i,a_i)\in\mathbb{X}_Q} (Q(s_i,a_i) - Q_\zeta(s_i,a_i))^2$ and $\mathcal{L}_V(V_\theta, \mathbb{X}_V) = \frac{1}{2|\mathbb{X}_V|} \sum_{s_i\in\mathbb{X}_V} (V(s_i) - V_\theta(s_i))^2$.

By using hyperspheres, CMAX++ "covers" the set of incorrect transitions, and enables fast lookup using KD-Trees in the state space. Like Algorithm 2, we never update the approximate model $\hat{M}$ used for planning. However, unlike Algorithm 2, we update the value estimates for sampled previous transitions and states (Lines 14 and 18). This ensures that the global function approximations used to maintain value estimates V_θ, Q_ζ have good generalization beyond the current state and action. Algorithm 3 can also be extended in a similar fashion as Algorithm 2 to include A-CMAX++ by maintaining a penalized value function approximation and updating it using gradient descent.

Experiments

We test the efficiency of CMAX++ and A-CMAX++ on simulated robotic tasks emphasizing their performance in each repetition of the task, and improvement across repetitions. In each task, we start the next repetition only if the robot reached a goal in previous repetition.

3D Mobile Robot Navigation with Icy Patches

In this experiment, the task is for a mobile robot with Reed-Shepp dynamics (Reeds and Shepp 1990) to navigate around a track M with icy patches (Figure 1 right.) This can be represented as a planning problem in 3D discrete state space $\mathbb{S}$ with any state represented using the tuple (x, y, θ) where (x, y) is the 2D position of the robot and θ describes its heading. The XY-space is discretized into 100×100 grid and the θ dimension is discretized into 16 cells. We construct a lattice graph (Pivtoraiko, Knepper, and Kelly 2009) using 66 motion primitives that are pre-computed offline respecting the differential constraints on the motion of the robot. The

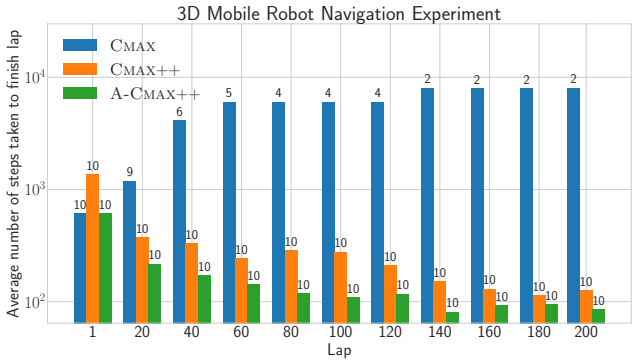

Figure 2: Number of steps taken to finish a lap averaged across 10 instances each with 5 icy patches placed randomly around the track. The number above each bar reports the number of instances in which the robot was successful in finishing the respective lap within 10000 time steps.

model $\hat{M}$ used for planning contains the same track as M but without any icy patches, thus the robot discovers transitions affected by icy patches only through executions.

Since the state space is small, we use Algorithm 2 for CMAX++ and A-CMAX++. For A-CMAX++, we use a non-increasing sequence with $\alpha_i = 1 + \beta_i$ where $\beta_1 = 100$ and β_i is decreased by 2.5 after every 5 repetitions (See Appendix A for more details on choosing the sequence.) We compare both algorithms with CMAX. For all the approaches, we perform $K = 100$ expansions. Since the motion primitives are computed offline using an expensive procedure, it is not feasible to update the dynamics of model $\hat{M}$ online and hence, we do not compare with any model learning baselines. We also conducted several experiments with model-free Q-learning, and found that it performed poorly requiring a very large number of executions and finishing only 10 laps in the best case. Hence, we do not include it in our results shown in Figure 2.

CMAX performs well in the early laps computing paths with lower costs compared to CMAX++. However, after a few laps the robot using CMAX gets stuck within an icy patch and does not make any more progress. Observe that when the robot is inside the icy patch, Assumption 0.1 is violated and CMAX ends up inflating all transitions that take the robot out of the patch leading to the robot finishing 200 laps in 2 out of 10 instances. CMAX++, on the other hand, is suboptimal in the initial laps, but converges to paths with lower costs in later laps. More importantly, the robot using CMAX++ manages to finish 200 laps in all 10 instances. A-CMAX++ also successfully finishes 200 laps in all 10 instances. However, it outperforms both CMAX and CMAX++ in all laps by intelligently switching between them achieving goal-driven behavior in early laps and optimal behavior in later laps. Thus, A-CMAX++ combines the advantages of CMAX and CMAX++.

7D Pick-and-Place with a Heavy Object

The task in this experiment is to pick and place a heavy object from a shorter table, using a 7 degree-of-freedom (DOF)

Repetition→	1		5		10		15		20	
	Steps	*Success*	*Steps*	*Success*	*Steps*	*Success*	*Steps*	*Success*	*Steps*	*Success*
CMAX	$\mathbf{17.8 \pm 3.4}$	100%	13.6 ± 0.5	60%	18 ± 0	20%	15 ± 0	20%	15 ± 0	20%
CMAX++	$\mathbf{17 \pm 4.9}$	100%	14.2 ± 3.3	100%	$\mathbf{10.6 \pm 0.3}$	100%	$\mathbf{11 \pm 0}$	100%	$\mathbf{10.8 \pm 0.1}$	100%
A-CMAX++	$\mathbf{17.8 \pm 3.4}$	100%	$\mathbf{11.6 \pm 0.7}$	100%	17 ± 6	100%	$\mathbf{10.4 \pm 0.3}$	100%	$\mathbf{10.6 \pm 0.4}$	100%
Model KNN	40.6 ± 7.3	100%	12.8 ± 1.3	100%	29.6 ± 16.1	100%	15.8 ± 2.9	100%	12.4 ± 1.4	100%
Model NN	56 ± 16.2	100%	208.2 ± 92.1	80%	124.5 ± 81.6	40%	28 ± 7.7	40%	37.5 ± 20.1	40%
Q-learning	172.4 ± 75	100%	23.2 ± 10.3	80%	26.5 ± 6.7	80%	18 ± 2.8	80%	10.2 ± 0.6	80%

Table 1: Number of steps taken to reach the goal in 7D pick-and-place experiment for 5 instances, each with random start and obstacle locations. We report mean and standard error *only among* successful instances in which the robot reached the goal within 500 timesteps. The success subcolumn indicates percentage of successful instances.

robotic arm (Figure 1 left) to a goal pose on a taller table, while avoiding an obstacle. As the object is heavy, the arm cannot generate the required force in certain configurations and can only lift the object to small heights. The problem is represented as planning in 7D discrete statespace where the first 6 dimensions describe the 6 DOF pose of the arm end-effector, and the last dimension corresponds to the redundant DOF in the arm. The action space $\mathbb{A}$ is a discrete set of 14 actions corresponding to moving in each dimension by a fixed offset in the positive or negative direction. The model $\hat{M}$ used for planning models the object as light, and hence does not capture the dynamics of the arm correctly when it tries to lift the heavy object. The state space is discretized into 10 cells in each dimension resulting in a total of 10^7 states. Thus, we need to use Algorithm 3 for CMAX++ and A-CMAX++. The goal is to pick and place the object for 20 repetitions where at the start of each repetition the object is in the start pose and needs to reach the goal pose by the end of repetition.

We compare with CMAX for large state spaces, model-free Q-learning (van Hasselt, Guez, and Silver 2016), and residual model learning baselines (Saveriano et al. 2017). We chose two kinds of function approximators for the learned residual dynamics: global function approximators such as Neural Networks (NN) and local memory-based function approximators such as K-Nearest Neighbors regression (KNN.) Q-learning baseline uses Q-values that are cleverly initialized using the model $\hat{M}$ making it a strong model-free baseline. We use the same neural network function approximators for maintaining value estimates for all approaches and perform $K = 5$ expansions. We chose the metric d as the manhattan metric and use $\xi = 0$ for this experiment. We use a radius of $\delta = 3$ for the hyperspheres introduced in the 7D discrete state space, and to ensure fair comparison use the same radius for KNN regression. These values are chosen to reflect the discrepancies observed when the arm tries to lift the object. All approaches use the same initial value estimates obtained through planning in $\hat{M}$. A-CMAX++ uses a non-increasing sequence $\alpha_i = 1+\beta_i$ where $\beta_1 = 4$ and $\beta_{i+1} = 0.5\beta_i$.

The results are presented in Table 1. Model-free Q-learning takes a large number of executions in the initial repetitions to estimate accurate Q-value estimates but in later repetitions computes paths with lower costs managing to finish all repetitions in 4 out of 5 instances. Among the residual model learning baselines, the KNN approximator is successful in all instances but takes a large number of executions to learn the true dynamics, while the NN approximator finishes all repetitions in only 2 instances. CMAX performs well in the initial repetitions but quickly gets stuck due to inflated costs and manages to complete the task for 20 repetitions in only 1 instance. CMAX++ is successful in finishing the task in all instances and repetitions, while improving performance across repetitions. Finally as expected, A-CMAX++ also finishes all repetitions, sometimes even having better performance than CMAX and CMAX++.

Discussion and Future Work

A major advantage of CMAX++ is that, unlike previous approaches that deal with inaccurate models, it can exploit inaccurately modeled transitions without wasting online executions to learn the true dynamics. It estimates the Q-value of incorrect transitions leveraging past experience and enables the planner to compute solutions containing such transitions. Thus, CMAX++ is especially useful in robotic domains with repetitive tasks where the true dynamics are intractable to model, such as deformable manipulation, or vary over time due to reasons such as wear and tear. Furthermore, the optimistic model assumption is easier to satisfy, when compared to assumptions used by previous approaches like CMAX, and performance of CMAX++ degrades gracefully with the accuracy of the model reducing to Q-learning in the case where the model is inaccurate everywhere. Limitations of CMAX++ and A-CMAX++ include hyperparameters such as the radius δ and the sequence $\{\alpha_i\}$, which might need to be tuned for the task. However, from our sensitivity experiments (see Appendix A) we observe that A-CMAX++ performance is robust to the choice of sequence $\{\alpha_i\}$ as long as it is non-increasing. Note that Assumption 0.2 can be restrictive for tasks where designing an initial optimistic model requires extensive domain knowledge. However, it is infeasible to relax this assumption further without resorting to global undirected exploration techniques (Thrun 1992), which are highly sample inefficient, to ensure completeness.

An interesting future direction is to interleave model identification with CMAX++ to combine the best of approaches that learn the true dynamics and CMAX++. For instance, given a set of plausible forward models we seek to quickly identify the best model while ensuring efficient performance in each repetition.

Acknowledgements

This work was supported by ONR grant N00014-18-1-2775 and ARL grant W911NF-18-2-0218. AV would like to thank Jacky Liang, Fahad Islam, Ankit Bhatia, Allie Del Giorno, Dhruv Saxena and Pragna Mannam for their help in reviewing the draft. AV is supported by the CMU presidential fellowship endowed by TCS. Finally, AV would like to thank Caelan Garrett for developing and maintaining the wonderful `ss-pybullet` library.

References

Abbeel, P.; Quigley, M.; and Ng, A. Y. 2006. Using inaccurate models in reinforcement learning. In Cohen, W. W.; and Moore, A. W., eds., *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*, 1–8. ACM. doi:10.1145/1143844.1143845. URL https://doi.org/10.1145/1143844.1143845.

Berenson, D.; Abbeel, P.; and Goldberg, K. 2012. A robot path planning framework that learns from experience. In *IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA*, 3671–3678. IEEE. doi:10.1109/ICRA.2012.6224742. URL https://doi.org/10.1109/ICRA.2012.6224742.

Bertsekas, D. P. 2005. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific. ISBN 1886529264. URL https://www.worldcat.org/oclc/314894080.

Essahbi, N.; Bouzgarrou, B. C.; and Gogu, G. 2012. Soft Material Modeling for Robotic Manipulation. In *Mechanisms, Mechanical Transmissions and Robotics*, volume 162 of *Applied Mechanics and Materials*, 184–193. Trans Tech Publications Ltd. doi:10.4028/www.scientific.net/AMM.162.184.

Hauser, K. K.; Bretl, T.; Harada, K.; and Latombe, J. 2006. Using Motion Primitives in Probabilistic Sample-Based Planning for Humanoid Robots. In Akella, S.; Amato, N. M.; Huang, W. H.; and Mishra, B., eds., *Algorithmic Foundation of Robotics VII, Selected Contributions of the Seventh International Workshop on the Algorithmic Foundations of Robotics, WAFR 2006, July 16-18, 2006, New York, NY, USA*, volume 47 of *Springer Tracts in Advanced Robotics*, 507–522. Springer. doi:10.1007/978-3-540-68405-3_32. URL https://doi.org/10.1007/978-3-540-68405-3_32.

Ji, X.; and Xiao, J. 2001. Planning Motions Compliant to Complex Contact States. *IJ Robotics Res.* 20(6): 446–465. doi:10.1177/02783640122067480. URL https://doi.org/10.1177/02783640122067480.

Jiang, N. 2018. PAC Reinforcement Learning With an Imperfect Model. In McIlraith, S. A.; and Weinberger, K. Q., eds., *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 3334–3341. AAAI Press. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16052.

Koenig, S.; and Likhachev, M. 2006. Real-time adaptive A*. In Nakashima, H.; Wellman, M. P.; Weiss, G.; and Stone, P., eds., *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, 281–288. ACM. doi:10.1145/1160633.1160682. URL https://doi.org/10.1145/1160633.1160682.

Koenig, S.; and Simmons, R. G. 1993. Complexity Analysis of Real-Time Reinforcement Learning. In Fikes, R.; and Lehnert, W. G., eds., *Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, USA, July 11-15, 1993*, 99–107. AAAI Press / The MIT Press. URL http://www.aaai.org/Library/AAAI/1993/aaai93-016.php.

Korf, R. E. 1990. Real-Time Heuristic Search. *Artif. Intell.* 42(2-3): 189–211. doi:10.1016/0004-3702(90)90054-4. URL https://doi.org/10.1016/0004-3702(90)90054-4.

Lagrassa, A.; Lee, S.; and Kroemer, O. 2020. Learning skills to patch plans based on inaccurate models. In *2020 IEEE International Conference on Intelligent Robots and Systems (IROS)*.

Lee, M. A.; Florensa, C.; Tremblay, J.; Ratliff, N. D.; Garg, A.; Ramos, F.; and Fox, D. 2020. Guided Uncertainty-Aware Policy Optimization: Combining Learning and Model-Based Strategies for Sample-Efficient Policy Learning. *CoRR* abs/2005.10872. URL https://arxiv.org/abs/2005.10872.

McConachie, D.; Power, T.; Mitrano, P.; and Berenson, D. 2020. Learning When to Trust a Dynamics Model for Planning in Reduced State Spaces. *IEEE Robotics Autom. Lett.* 5(2): 3540–3547. doi:10.1109/LRA.2020.2972858. URL https://doi.org/10.1109/LRA.2020.2972858.

Pivtoraiko, M.; Knepper, R. A.; and Kelly, A. 2009. Differentially constrained mobile robot motion planning in state lattices. *J. Field Robotics* 26(3): 308–333. doi:10.1002/rob.20285. URL https://doi.org/10.1002/rob.20285.

Ramos, F.; Possas, R.; and Fox, D. 2019. BayesSim: Adaptive Domain Randomization Via Probabilistic Inference for Robotics Simulators. In Bicchi, A.; Kress-Gazit, H.; and Hutchinson, S., eds., *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*. doi:10.15607/RSS.2019.XV.029. URL https://doi.org/10.15607/RSS.2019.XV.029.

Reeds, J. A.; and Shepp, L. A. 1990. Optimal paths for a car that goes both forwards and backwards. *Pacific J. Math.* 145(2): 367–393. URL https://projecteuclid.org:443/euclid.pjm/1102645450.

Saveriano, M.; Yin, Y.; Falco, P.; and Lee, D. 2017. Data-efficient control policy search using residual dynamics learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, 4709–4715. IEEE. doi:10.1109/IROS.2017.8206343. URL https://doi.org/10.1109/IROS.2017.8206343.

Sun, W.; Jiang, N.; Krishnamurthy, A.; Agarwal, A.; and Langford, J. 2019. Model-based RL in Contextual Decision Processes: PAC bounds and Exponential Improvements over Model-free Approaches. In Beygelzimer, A.; and Hsu, D., eds., *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*, volume 99 of *Proceedings of Machine Learning Research*, 2898–2933. PMLR. URL http://proceedings.mlr.press/v99/sun19a.html.

Sutton, R. S. 1991. Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *SIGART Bull.* 2(4): 160–163. doi:10.1145/122344.122377. URL https://doi.org/10.1145/122344.122377.

Thrun, S. 1992. Efficient Exploration In Reinforcement Learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, Pittsburgh, PA.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, 5026–5033. IEEE. doi:10.1109/IROS.2012.6386109. URL https://doi.org/10.1109/IROS.2012.6386109.

van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep Reinforcement Learning with Double Q-Learning. In Schuurmans, D.; and Wellman, M. P., eds., *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, 2094–2100. AAAI Press. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389.

Vemula, A.; Oza, Y.; Bagnell, J.; and Likhachev, M. 2020. Planning and Execution using Inaccurate Models with Provable Guarantees. In *Proceedings of Robotics: Science and Systems*. Corvalis, Oregon, USA. doi:10.15607/RSS.2020.XVI.001.