CAMPs: Learning Context-Specific Abstractions for Efficient Planning in Factored MDPs

Rohan Chitnis[†*], **Tom Silver**[†*], **Beomjoon Kim**[§], **Leslie Pack Kaelbling**[†], **Tomás Lozano-Pérez**[†]
[†]MIT Computer Science and Artificial Intelligence Laboratory [§]KAIST Graduate School of AI
{ronuchit, tslvr, lpk, tlp}@mit.edu, beomjoon.kim@kaist.ac.kr

Abstract: Meta-planning, or learning to guide planning from experience, is a promising approach to improving the computational cost of planning. A general meta-planning strategy is to learn to impose constraints on the states considered and actions taken by the agent. We observe that (1) imposing a constraint can induce *context-specific independences* that render some aspects of the domain irrelevant, and (2) an agent can take advantage of this fact by imposing constraints *on its own behavior*. These observations lead us to propose the context-specific abstract Markov decision process (CAMP), an abstraction of a factored MDP that affords efficient planning. We then describe how to learn constraints to impose so the CAMP optimizes a trade-off between rewards and computational cost. Our experiments consider five planners across four domains, including robotic navigation among movable obstacles (NAMO), robotic task and motion planning for sequential manipulation, and classical planning. We find planning with learned CAMPs to consistently outperform baselines, including Stilman's NAMO-specific algorithm. Video: https://youtu.be/wTXt6djcAd4 Code: https://git.io/JTnf6

Keywords: learning for planning, abstractions, context-specific independence

1 Introduction

Online planning is a popular paradigm for sequential decision-making in robotics and beyond, but its practical application is limited by the computational burden of planning while performing a task. In *meta-planning*, the agent learns to guide planning efficiently and effectively based on its previous planning experience. *Learning to impose constraints* on the states considered and actions taken by an agent is a promising paradigm for meta-planning; it reduces the space of policies the agent must consider [1, 2, 3]. In contrast to (e.g., physical or kinematic) constraints beyond the agent's control, these constraints are imposed by the agent on itself for the sole purpose of efficient planning.

Beyond reducing the space of policies, imposing constraints can improve planning efficiency in another important way. In factored domains, where the states and actions decompose into variables, imposing constraints can induce *context-specific independences* (CSIs) [4] that render some variables irrelevant. For example, consider the two navigation among movable obstacles (NAMO) problems in Figure 1A–B. Imposing a constraint that forbids certain rooms induces CSIs between the robot's position and that of all obstacles in those forbidden rooms. Consequently, these obstacles can be ignored, as in Figure 1C–D. A planning problem with a context imposed[1] and the resulting irrelevant variables removed constitutes an *abstraction* of the original problem [5, 6]. Adopting the Markov decision process (MDP) formalism, we refer to this as a *context-specific abstract* MDP (CAMP).

Planning in a CAMP is often more efficient than planning in the original MDP, but may abstract away important details of the environment, leading to a suboptimal policy. Practically speaking, we are often interested in a trade-off: we would like our planners to produce highly rewarding behavior, but not be too computationally burdensome; we are willing to sacrifice optimality, and in the case of goal-based tasks, even soundness and completeness, to maximize this trade-off in expectation. In this work, we propose a learning-based approach to maximize this trade-off. Given a set of training

[*]Equal contribution.
[1]We henceforth use *context* as a synonym for *constraint*.

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

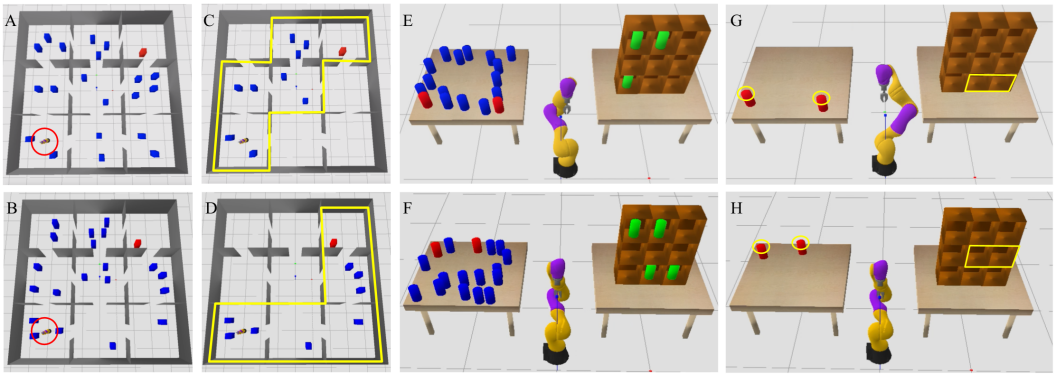


Figure 1: (A, B) In the NAMO domain, the robot (red circle) must reach the red object, which requires navigating there while moving obstacles out of the way. Two sample problems are shown. (C, D) If the robot constrains itself to stay within certain rooms (yellow), the obstacles in other rooms become irrelevant. (E, F) In the sequential manipulation domain, the robot must put the red objects into bins. (G, H) If the robot constrains itself to *top* grasps, the blue objects become irrelevant. Similarly, if the robot constrains target placements to certain bins (yellow), the green objects become irrelevant.

tasks with a shared transition model and factored states and actions, we first approximate the set of CSIs present in these tasks. We then train a *context selector*, which predicts a context that should be imposed for a given task. At test time, given a novel task, we use the learned context selector and CSIs to induce a CAMP, which we then use to plan. This overall pipeline is summarized in Figure 2.

Our approach rests on the premise that predicting contexts to impose is easier, and generalizes better, than learning a reactive policy. Intuitively, the burden on reactive policy learning is higher, as the policy must exactly carve out a specific, good path through transition space, whereas an imposed context must only carve out a region of transition space that includes at least one good path.

In experiments, we consider four domains, including robotic NAMO and sequential manipulation, that collectively exhibit discrete and continuous states and actions, relational states, sparse rewards, stochastic transitions, and long planning horizons. To evaluate the generality of CAMPs, we consider multiple planners, including Monte Carlo tree search [7], FastDownward [8], and a task and motion planner [9]. Our results suggest that planning with learned CAMPs strikes a strong balance between pure planning and pure policy learning [10]. In the NAMO domain, we also find that CAMPs with a generic task and motion planner outperform Stilman's NAMO-specific algorithm [11]. We conclude that CAMPs offer a promising path toward fast, effective planning in large and challenging domains.

2 Preliminaries

We introduce notation and background formalism (§2.1), and then present a problem definition (§2.2).

2.1 Context-Specific Independence in Factored Markov Decision Processes

A Markov decision process (MDP) is given by $(\mathcal{S}, \mathcal{A}, T, R, H)$, with: state space $\mathcal{S}$; action space $\mathcal{A}$; transition model $T(s_t, a_t, s_{t+1}) = P(S_{t+1} = s_{t+1} \mid S_t = s_t, A_t = a_t)$ where $s_t, s_{t+1} \in \mathcal{S}$, $a_t \in \mathcal{A}$, and S_t, A_t are random variables denoting the state and action taken at time t; reward function $R(s_t) = r_t \in \mathbb{R}$; and horizon H.[2] The solution to an MDP is a policy $\pi : \mathcal{S} \to \mathcal{A}$, a mapping from states to actions, that maximizes the expected sum of rewards with respect to the transitions.

We focus on *factored* MDPs [12], where each state variable S is factored into n variables $\{S^1, \ldots, S^n\}$, where S^i has domain $\mathcal{S}^i$. A state $s \in \mathcal{S}$ is then an assignment $s = [s^1, \ldots, s^n]$ with $s^i \in \mathcal{S}^i$; thus, $\mathcal{S} \subseteq \mathcal{S}^1 \times \ldots \times \mathcal{S}^n$. Actions are similarly factored into m variables $\{A^1, \ldots, A^m\}$ with domains $\mathcal{A}^i$ so that $a \in \mathcal{A}$ is an assignment $[a^1, \ldots, a^m]$ with $a^i \in \mathcal{A}^i$; thus, $\mathcal{A} \subseteq \mathcal{A}^1 \times \ldots \times \mathcal{A}^m$. The reward function for a factored MDP is defined in terms of a subset of state variables $S_{\text{rew}} \subseteq \{S^1, \ldots, S^n\}$,

[2] We say the reward is a function of only s_t for simplicity of notation; this is not critical to our method.

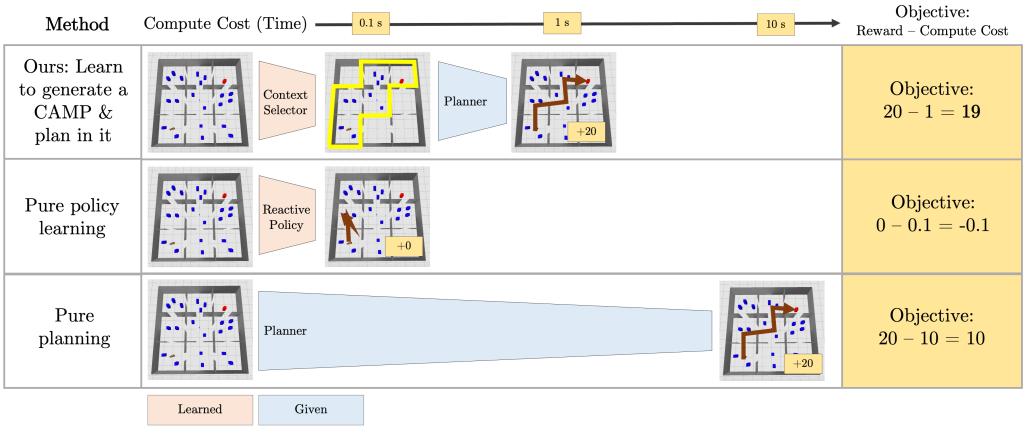


Figure 2: Three approaches to solving an MDP. Given a task, our approach (top row) applies its learned context selector to generate a CAMP, then plans in this CAMP to get a policy. Our approach often achieves higher reward than pure policy learning (middle row), and lower computational cost than pure planning (bottom row), leading to a good objective value (right; uses $\lambda = 1$ in Equation 1).

which we call the *reward variables*. Let $V = \{S^1, \ldots, S^n\} \cup \{A^1, \ldots, A^m\}$ denote all state and action variables together. Variable domains may be discrete or continuous for both states and actions.

Following [4], we define a *context* as a pair $(C, \mathcal{C})$, where $C \subseteq V$ is some subset of state and action variables, and $\mathcal{C}$ is a space of possible joint assignments. A state-action pair (s, a) is *in the context* $(C, \mathcal{C})$ when its joint assignment of variables in C is present in $\mathcal{C}$. Two variables $X, Y \subseteq V \setminus C$ are *contextually independent* under $(C, \mathcal{C})$ if $P(X \mid Y, C = c) = P(X \mid C = c)\ \forall\, c \in \mathcal{C}$, in which case we write $X \perp\!\!\!\perp Y \mid (C, \mathcal{C})$. This relation is called a *context-specific independence* (CSI). In this paper, we explore how CSIs can be automatically identified and exploited for planning in factored MDPs.

2.2 Problem Formulation

A *task* is a pair of initial state and reward function, denoted $\omega = (s_0, R)$. We are given a set of N training tasks, $W_{\text{train}} = \{\omega^{(i)}\}_{i=1}^{N}$, and a test task, ω_{test}, all of which are drawn from some unseen distribution $P(\mathcal{W})$. All tasks share the same factored state space $\mathcal{S}$, factored action space $\mathcal{A}$, transition model T, and horizon H; therefore, each task induces a factored MDP, denoted $\mathcal{M}_\omega$. Each task is parameterized by a feature vector, denoted $\theta_\omega = \phi(s_0, R)$, with featurizer ϕ. For instance, in our robotic NAMO domain, ϕ gives a top-down image of the initial scene (which also implicitly describes the goal). The agent interacts with T and R as black boxes; it does not know their analytical representations or causal structure. We also assume a black-box MDP solver, PLAN, which takes as input an MDP $\mathcal{M}$ and a current state s, and returns a next action: $a = \textsc{Plan}(\mathcal{M}, s)$.[3]

Before being presented with the test task, the agent may first interact with the training tasks W_{train}, perhaps compiling useful knowledge that it can deploy at test time, such as a task-conditioned policy. Then, it is given the test task $\omega_{\text{test}} = (s_{0,\text{test}}, R_{\text{test}})$, and its goal is to *efficiently* produce actions that accrue high cumulative reward in the test MDP $\mathcal{M}_{\omega_{\text{test}}}$. We formalize this trade-off via the objective:

$$J(\pi, \omega) = \mathbb{E}\left[\sum_{t=0}^{H} R(s_t) - \lambda \cdot \textsc{ComputeCost}(\pi, s_t)\right], \tag{1}$$

where $\omega = (s_0, R)$, $\textsc{ComputeCost}(\pi, s) \geq 0$ denotes the cost (e.g., wall-clock time) of evaluating the policy $\pi(s)$, $\lambda \geq 0$ is a trade-off parameter, and the expectation is over stochasticity in the transitions. Note that COMPUTECOST includes the cost of both computation performed before the agent starts acting and any computation that might be performed on each timestep after the first.

We seek to find $\pi_{\text{test}} = \text{argmax}_\pi\, J(\pi, \omega_{\text{test}})$. One possible approach is to call PLAN on the full test MDP, that is, $\pi_{\text{test}}(s) = \textsc{Plan}(\mathcal{M}_{\omega_{\text{test}}}, s)$. This method would yield high rewards, but it may also

[3] Planners generally return a policy or a sequence of actions; we suppose that the planner is called at every timestep to simplify exposition. In our experiments, we replan in domains that have stochastic transitions.

incur a large COMPUTECOST. Another possibility is to learn (at training time) and transfer (to test time) a task-conditioned reactive policy; this can have low COMPUTECOST at test time, but perhaps at the expense of rewards if the policy fails to generalize well to the test task (Figure 2).

3 Context-Specific Abstract Markov Decision Processes (CAMPs)

The objective formulated in Equation 1 trades off the computational cost of planning with the resulting rewards. In this section, we present an approach to optimizing this trade-off that lies between the two extremes of pure planning and pure policy learning [10]. Rather than planning in the full test task, we propose to *learn* to generate an abstraction [5, 6] of the test task, in which we can plan efficiently.

An *abstraction* over state space $\mathcal{S}$ and action space $\mathcal{A}$ is a pair of functions (σ, τ), with $\sigma : \mathcal{S} \mapsto \mathcal{S}'$ and $\tau : \mathcal{A} \mapsto \mathcal{A}'$, where $\mathcal{S}'$ and $\mathcal{A}'$ are *abstract* state and action spaces. We are specifically interested in abstractions that are projections: $\sigma([s^1, s^2, \ldots, s^n]) = [s^{i_1}, s^{i_2}, \ldots, s^{i_{n'}}]$ and $\tau([a^1, a^2, \ldots, a^m]) = [a^{j_1}, a^{j_2}, \ldots, a^{j_{m'}}]$. This has the effect of dropping $n - n'$ state variables and $m - m'$ action variables; the i and j superscripts refer respectively to the state and action variables that are not dropped.

The *relevant-variables projection* is a simple projective abstraction that has been studied in prior work (under different names) [13, 14, 15]. It drops all irrelevant variables, in the following sense:

Definition 1 (Variable relevance)**.** *Given a factored* MDP *with variables V and reward variables S_{rew}, any $V^i \in V$ is* relevant *iff $\exists\, V^j \in S_{rew}$, $t \in \{0, \ldots, H\}$, and $t' \in \{t+1, \ldots, H\}$ s.t. $V_t^i \not\perp\!\!\!\perp V_{t'}^j$.*

Intuitively, a variable is relevant if there is *any* possibility that its value at some timestep will have an eventual influence, directly or indirectly, on the value of the reward. Unfortunately, as identified by Baum et al. [13], relevance is often too strong of a property for the relevant-variables projection to yield meaningful improvements in practice — most variables typically have *some* way of influencing the reward, under *some* sequence of actions taken by the agent. In search of greater flexibility, we now define a generalization of variable relevance that is conditioned on a particular context (§2.1).

Definition 2 (Context-specific variable relevance)**.** *Given a context $(C, \mathcal{C})$ and a factored* MDP *with variables V and reward variables S_{rew}, any $V^i \in V \setminus C$ is* relevant in the context $(C, \mathcal{C})$ *iff $\exists\, V^j \in S_{rew}$, $t \in \{0, \ldots, H\}$, and $t' \in \{t+1, \ldots, H\}$ s.t. $V_t^i \not\perp\!\!\!\perp V_{t'}^j \mid (C, \mathcal{C})$.*

Each possible context $(C, \mathcal{C})$ induces a projection that drops variables which are irrelevant in $(C, \mathcal{C})$; let $\text{proj}_{C,\mathcal{C}}$ denote this abstraction. We now define a CAMP, an abstract MDP associated with $\text{proj}_{C,\mathcal{C}}$.

Definition 3 (Context-Specific Abstract MDP (CAMP))**.** *Consider an* MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, H)$ *and a context $(C, \mathcal{C})$. Let $proj_{C,\mathcal{C}} = (\sigma, \tau)$ with right inverses (σ^{-1}, τ^{-1}). Let $\perp$ be a new sink state, such that $\perp \notin \sigma(\mathcal{S})$. The context-specific abstract* MDP*, $\mathcal{M}'$, for $\mathcal{M}$ and $(C, \mathcal{C})$ is $(\sigma(\mathcal{S}) \cup \{\perp\}, \tau(\mathcal{A}), T', R', H)$, where T' and R' are defined as follows: $\forall s'_t, s'_{t+1} \in \sigma(\mathcal{S}), a'_t \in \tau(\mathcal{A})$,*

1. $T'(\perp, a'_t, \perp) = 1$
2. $T'(s'_t, a'_t, \perp) = 1$ *if* $(\sigma^{-1}(s'_t), \tau^{-1}(a'_t))$ *is not in the context;*
3. $T'(s'_t, a'_t, s'_{t+1}) = T(\sigma^{-1}(s'_t), \tau^{-1}(a'_t), \sigma^{-1}(s'_{t+1}))$ *if* $(\sigma^{-1}(s'_t), \tau^{-1}(a'_t))$ *is in the context;*
4. $R'(\perp) = -\infty$
5. $R'(s'_t) = R(\sigma^{-1}(s'_t))$

We may also say that $\mathcal{M}'$ is $\mathcal{M}$ with context $(C, \mathcal{C})$ imposed.

Intuitively, a CAMP imposes a projective abstraction that drops the variables that are irrelevant *under the given context*, and also imposes that any transition in violation of the context leads the agent to $\perp$, an absorbing sink state with reward $-\infty$. In practice, the right inverses σ^{-1} and τ^{-1} can be obtained by assigning arbitrary values to the dropped variables; the choice of value is inconsequential by Definition 2. For a graphical example of a CAMP, see Appendix C.

A CAMP is usually *not* optimality-preserving, because the context restricts the agent to a subregion of the state and action space [16]. However, context-specific relevance is much weaker than relevance: it only requires a variable to be relevant under the given context. For example, to a robot operating in a home, the weather outside is irrelevant as long as it remains in the context of staying indoors.

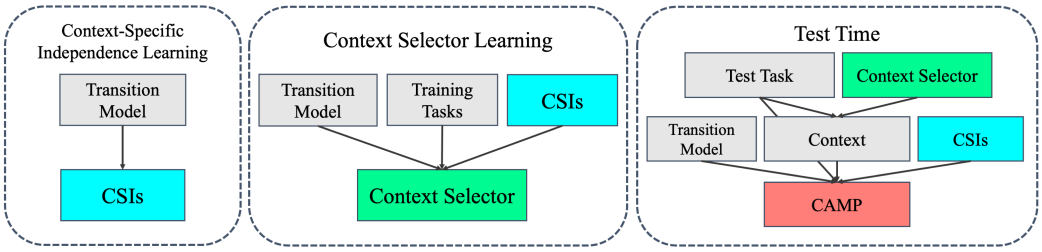


Figure 3: Data-flow diagram for our method during learning and test time. (Left) Approximate context-specific independence (CSI) learning derives the relevant state and action variables under each context. (Middle) A context selector is learned by optimizing the objective on the training tasks. The transition model and approximate CSIs are used to evaluate the objective. (Right) Given a test task, the agent selects a context to impose using the learned context selector. The relevant variables for this context are calculated from the learned CSIs. From the context, relevant variables, test task, and transition model, the agent derives a CAMP, and can plan in it to obtain a policy for the test task.

CAMPs offer a way to solve the test task (§2.2) that lies between the extremes of pure planning and pure policy learning. Namely, given a test MDP $\mathcal{M}_{\omega_{\text{test}}}$, we select a context, compute the relevant variables under that context via backward induction, generate the CAMP, and finally plan in this CAMP to obtain a policy π_{test} for $\mathcal{M}_{\omega_{\text{test}}}$. We have therefore reduced the problem of optimizing $J(\pi, \omega_{\text{test}})$ to that of determining the best context $(C, \mathcal{C})$ to impose. To address this issue, we now turn to learning.

4 Learning to Generate CAMPs

We now have the ability to generate a context-specific abstract MDP (CAMP) when given a context and the associated context-specific independences. However, contexts and their associated independences are *not* provided in our problem. In this section, we describe how to learn approximate context-specific independences and a context selector, for use at test time. Figure 3 gives a data-flow diagram.

4.1 Approximating the Context-Specific Independences

Recall that the agent is given MDPs with factored states and variables, but only query access to the (shared) transition model. For example, the transition model may be a black-box physics simulator, as in two of our experimental domains. In order to approximately determine the context-specific independences that are latent within a factored MDP, we propose a sample-based procedure. Given a context, the algorithm examines each pair of state or action variables (V^i, V^j) and tests for empirical dependence, that is, whether any sampled value of V_t^i induces a change in the distribution of V_{t+1}^j, conditioned on the sampled values of the remaining variables. For full pseudocode, see Appendix A.

The runtime of this algorithm depends on the size of the domain and the number of samples used to test dependence. In theory, the number of samples required to identify all independences could be arbitrarily large. In practice, for the tasks we considered in our experiments, including robotic manipulation and NAMO, we found this algorithm to be sufficient for detecting a useful set of context-specific independences. Moreover, our method is robust to errors in the discovered independences, which in the worst case will simply exclude some candidate abstractions from consideration.

Where Does the Space of Contexts Come From? Our approximate algorithm allows us to estimate independences *given a context*; this raises the question of which contexts should be evaluated. We propose a simple method for deriving a space of possible contexts that works well across our varied experimental domains. From the set of variables V, we consider conjunctions and disjunctions up to some length (a hyperparameter), excluding any terms whose involved variables have joint domain size less than some threshold (another hyperparameter). Note that for any finite threshold, this procedure immediately excludes contexts involving continuous variables. While this family of contexts has the benefit of being fairly general to describe, we emphasize that other choices, e.g., more domain-specific context families, may be used as well [1, 17, 18]. Importantly, the space of considered contexts should always include the trivial universal context so that CAMPs can reduce to planning in the original problem when no useful abstractions are available.

4.2 Learning the Context Selector

The performance of a CAMP depends entirely on the selected context; if the context constrains the agent to a poor region of plan space, or induces independences that make important variables irrelevant, the resulting policy could get very low rewards. However, if the context is selected judiciously, the CAMP may exhibit substantial efficiency gains with minor impact on rewards.

We now describe an algorithm for learning to select a context that optimizes the objective (Equation 1). Pseudocode is presented in Appendix B. Given each training task $\omega^{(i)} \in W_{\text{train}}$, we first identify the best possible context $(C^{(i)}, \mathcal{C}^{(i)})^*$ according to the objective in Equation 1. This process sets up a supervised multiclass classification problem that maps the featurized representation of a task $\theta_{\omega^{(i)}}$ to the best context $(C^{(i)}, \mathcal{C}^{(i)})^*$ to impose on that task. We solve this classification problem by training a neural network with cross-entropy loss, resulting in a context selector $f_\alpha(\theta_\omega) = (C, \mathcal{C})^*$, where α denotes the parameters of the neural network. At test time, we choose a context by calling $f_\alpha(\theta_{\omega_{\text{test}}})$, generate the associated CAMP, and plan in this CAMP to efficiently obtain a policy for the test task.

5 Experiments and Results

Our experiments aim to answer the following key questions:

- How does planning with learned CAMPs compare to pure planning and pure policy learning across a varied set of domains, both discrete and continuous? (§5.1), (§5.3)
- To what extent is the performance of CAMPs planner-agnostic? (§5.1), (§5.3), (Appendix H)
- How does the performance of CAMPs vary with the choice of λ (Equation 1)? (§5.4)
- How does the performance of CAMPs vary with the number of training tasks? (Appendix F)

We overview the experimental setup in (§5.1) and (§5.2), and provide details in Appendix E. Then, we present our main results in (§5.3), with additional results in (§5.4) and Appendix F–H.

5.1 Domains and Planners

We consider four domains and five planners (four online, one offline). Full details are in Appendix D.

Domain D1: Gridworld. A maze-style gridworld in which the agent must navigate across rooms to reach a goal location, while avoiding or destroying stochastically moving obstacles. Task features are top-down images of the maze layout. The state is a vector of the current position and room of each obstacle, the agent, and the goal. The actions are moving up, down, left, right; and destroying each obstacle. For planning in this domain, we consider Monte Carlo tree search (**MCTS**), breadth-first graph search with replanning (**BFSReplan**), and value iteration (**VI**). VI results are in Appendix H.

Domain D2: Classical planning. A deterministic dinner-making domain written in PDDL [19], with three different possible meals to make. Preparing each meal requires a different number of actions. The relative rewards for making each meal are the only source of variation between tasks; task features are simply these rewards. States are binary vectors describing which logical fluents hold true, and actions are logical operators. We use an off-the-shelf classical planner (Fast-Downward [8]).

Domain D3: Robotic navigation among movable obstacles (NAMO), simulated in PyBullet [20]. Task features are overhead images. The state is a vector of the current pose of each object and the robot, and the robot's current room. The actions are moving the robot to a target pose, and clearing an object in front of the robot. We use a state-of-the-art TAMP planner [9], which is *not* NAMO-specific.

Domain D4: Robotic sequential manipulation, simulated in PyBullet [20]. Task features are a vector of the object radii and occupied bins. The state is a vector of the current pose of each object, the grasp style used by the robot, and the current held object (if any). The actions are moving the robot to a target base pose and grasping at a target gripper pose, and moving the robot to a target base pose and placing at a target placement pose. The planner for this task is the same as in NAMO.

5.2 Methods and Baselines

We consider the following methods:

- CAMP. Our full method.

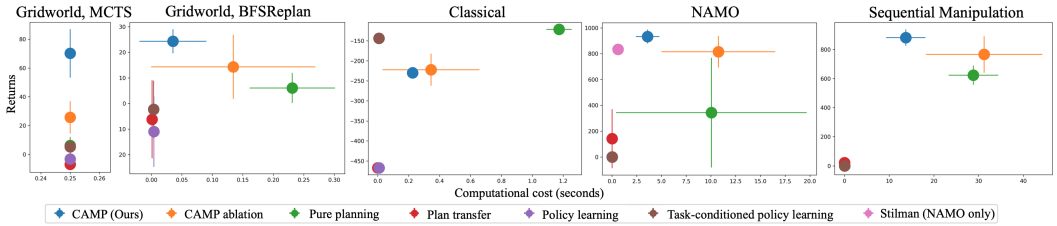


Figure 4: Mean returns versus computation time on the test tasks, for all domains and methods. All points report an average over 10 independent runs of training and evaluation, with lines showing per-axis standard deviations. CAMPs generally provide a better trade-off than the baselines: the blue points are usually higher than pure policy learning (CAMPs accrue more reward) and to the left of pure planning (CAMPs are more efficient). For the left-most plot, only the returns vary because MCTS is an anytime algorithm, so we run it with a fixed timeout. See (§5.3) for discussion on these results.

- CAMP ablation. An ablation of our full method in which the CAMP only sends the agent to a sink state for context violation, but does *not* project away irrelevant variables.
- Pure planning. This baseline does not use the training tasks, and just solves the full test task.
- Plan transfer. This baseline solves each training task to obtain a plan, and at test time picks actions via majority vote across the training task plans.
- Policy learning. This baseline solves each training task to obtain a plan, then trains a state-conditioned neural network policy to imitate the resulting dataset of state-action trajectories, using supervised learning. This policy is used directly to choose actions at test time.
- Task-conditioned policy learning. This baseline is the same as policy learning, but the neural network also receives as input the features of the task, in addition to the current state.
- (Domain D3 only) Stilman's planning algorithm [11] for NAMO problems, named ResolveSpatial-Constraints, which attempts to find a feasible path to a target location by first finding feasible paths to any obstructing objects and moving them out of the way.

In all our domains, every variable is relevant under *no* context. For this reason, the pure planning baseline can also be understood as an ablation of CAMP that does not account for contexts.

5.3 Main Results

Figure 4 plots the mean returns versus computation time on the test tasks, for all domains and methods. Table 1 in Appendix G shows the corresponding objective values (Equation 1). All results report an average over 10 independent runs of context selector training and test task evaluation.

Discussion. CAMPs outperform every baseline in all but Domain D2 (classical planning). CAMPs fare better than task-conditioned policy learning because the latter fails to generalize from training tasks to test tasks. This failure manifests in low test task rewards, and in a substantial difference between the training and test objective values. In classical planning, however, policy learning outperforms CAMPs; both achieve high task rewards, but the policy is faster to execute. This is because this domain involves little variation between instances, in stark contrast to the other domains. In Appendix F, we unpack this result further by analyzing performance versus number of training tasks.

Another clear conclusion from the main results is that CAMPs outperform pure planning across all experiments, consistently achieving lower computational costs. In several cases, including NAMO and manipulation, CAMPs also achieve higher *rewards* than pure planning does, since the latter sometimes hits the 60-second timeout before discovering the superior plans found very quickly by CAMPs.

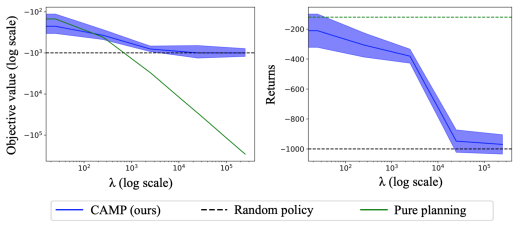
Results for the CAMP ablation show that imposing contexts alone provides clear benefits, focusing the planner on a promising region of the search space. This result is consistent with prior work showing that learning to impose constraints reduces planning costs [1, 2, 3]. However, the difference between CAMP and the ablation shows that dropping irrelevant variables provides even greater benefits.

A final observation is that CAMPs perform comparably to Stilman's NAMO algorithm [11]. This is notable because Stilman's algorithm employs NAMO-specific assumptions, whereas the planner we use does not; in fact, we can see that the pure planner is strongly outperformed by Stilman's algorithm. In our method, the context selector learns to constrain the robot to stay in emptier rooms, meaning it

must move comparatively few objects out of the way. This leads to efficient planning, making the computational cost of CAMPs almost as good as that of Stilman's algorithm; additionally, it leads CAMPs to obtain higher rewards than Stilman's algorithm, because it often reaches the goal faster. The NAMO results also show that CAMPs are able to learn to impose useful contexts even when there are multiple good options, e.g., multiple "room paths" with similar numbers of obstacles (Figure 1).

5.4 Performance as a Function of λ

The plots on the right illustrate how the objective value (left) and returns (right) accrued by the CAMP policy vary as a function of λ, in Domain D2 (classical planning). The right figure shows the returns from CAMP interpolate between those obtained by pure planning (when $\lambda = 0$, the agent is okay with spending a long time planning out its actions) and those obtained by a random policy (when $\lambda \to \infty$, the agent spends as little time as possible choosing actions). The green line is dashed because pure planning does not use λ, so its returns are unaffected by the value of λ. The left figure (note the log-scale y-axis) shows objective values. We see that CAMP never suffers a lower objective value than that of a random policy, while pure planning drops down greatly as λ increases. This is because as $\lambda \to \infty$, our context selector learns to choose contexts that induce very little planning (but get low returns).



6 Related Work

Our work falls under the broad research theme of learning to make planners more efficient using past planning experience. A fundamental question is deciding what to predict; for instance, it is common to learn a policy and/or value function from planning experience [21, 22, 23, 24]. In contrast, we learn to predict *contexts*. Recent work leverages a given set of contexts to represent planning problem instances in "score space" [1], but does not consider the resulting CSIs, which we showed experimentally to yield large performance improvements. Other methods predict the feasibility of task plans or motion plans [2, 3, 17], which can also be seen as learning constraints on the search space. These methods can be readily incorporated into the CAMP framework.

We have formalized CAMPs as a particular class of MDP abstractions. There is a long line of work on deriving abstractions for MDPs, much of it motivated by the prospect of faster planning or more sample-efficient reinforcement learning [5, 14, 25, 26, 27]. One common technique is to *aggregate* states and actions into equivalence classes [28, 29, 6], a generalization of our notion of projective abstractions. Other work has learned to select abstractions [30, 31]; a key benefit of CAMPs is that the contexts induce a structured hypothesis space of abstractions that greatly improve planning efficiency.

CAMPs identify and exploit CSIs [4] in factored planning problems. In graphical models, CSIs can be similarly used to speed up inference [32, 33, 34]. Stochastic Planning using Decision Diagrams (SPUDD) is a method that adapts these insights for planning with CSIs [35]. These insights are orthogonal to CAMPs, but could be integrated to yield further efficiencies. SPUDD is a pure planning approach that considers *all* contexts, whereas we *learn* a context selector that induces abstractions.

7 Conclusion

In this work, we have presented a method for learning to generate context-specific abstractions of MDPs, achieving more efficient planning while retaining high rewards. There are several clear directions for future work. On the learning side, one interesting question is whether factorizations of initially unfactored MDPs can be automatically discovered in a way that leads to useful CAMPs. Another direction to pursue is learning the task featurizer ϕ, which we assumed to be given in our problem formulation. Following [1], it could also be useful to extend the methods we have presented here so that multiple contexts can be imposed in succession at test time, using the performance of previous contexts to inform the choice of future ones. However, note that such a method would lead to an increase in computational cost, possibly to the detriment of the overall objective we formulated.

Acknowledgments

We would like to thank Kelsey Allen for valuable comments on an initial draft. We gratefully acknowledge support from NSF grant 1723381; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; from the Honda Research Institute; from MIT-IBM Watson Lab; and from SUTD Temasek Laboratories. Rohan and Tom are supported by NSF Graduate Research Fellowships. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

[1] B. Kim, L. P. Kaelbling, and T. Lozano-Pérez. Learning to guide task and motion planning using score-space representation. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2810–2817. IEEE, 2017.

[2] D. Driess, J.-S. Ha, and M. Toussaint. Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image. In *Proc. of Robotics: Science and Systems (R:SS)*, 2020.

[3] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki. Learning feasibility for task and motion planning in tabletop environments. *IEEE Robots and Automation Letters*, 2018.

[4] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in bayesian networks. In *Proceedings of the Twelfth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1996.

[5] L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for mdps. In *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, pages 531–539, 2006.

[6] G. Konidaris and A. Barto. Efficient skill learning using abstraction selection. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[7] 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*, 4(1):1–43, 2012.

[8] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[9] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE, 2014.

[10] T. M. Moerland, A. Deichler, S. Baldi, J. Broekens, and C. M. Jonker. Think too fast nor too slow: The computational trade-off between planning and reinforcement learning, 2020.

[11] M. Stilman and J. J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, 2(04):479–503, 2005.

[12] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

[13] J. Baum, A. E. Nicholson, and T. I. Dix. Proximity-based non-uniform abstractions for approximate planning. *Journal of Artificial Intelligence Research*, 43:477–522, 2012.

[14] N. Hernandez-Gardiol. *Relational envelope-based planning*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2008. URL http://hdl.handle.net/1721.1/43028.

[15] C. Boutilier. Correlated action effects in decision theoretic regression. In *UAI*, pages 30–37, 1997.

[16] D. Abel, N. Umbanhowar, K. Khetarpal, D. Arumugam, D. Precup, and M. L. Littman. Value preserving state-action abstractions, 2019.

[17] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint. Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[18] J. Carpentier, R. Budhiraja, and N. Mansard. Learning feasibility constraints for multi-contact locomotion of legged robots. In *Robotics: Science and Systems*, page 9p, 2017.

[19] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL-the planning domain definition language, 1998.

[20] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub repository*, 2016.

[21] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 2016.

[22] B. Kim, L. P. Kaelbling, and T. Lozano-Pérez. Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[23] B. Kim and L. Shimanuki. Learning value functions with relational state representations for guiding task-and-motion planning. *Conference on Robot Learning*, 2019.

[24] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel. Guided search for task and motion plans using learned heuristics. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 447–454. IEEE, 2016.

[25] N. K. Jong and P. Stone. State abstraction discovery from irrelevant state variables. In *IJCAI*, volume 8, pages 752–757, 2005.

[26] K. A. Steinkraus. *Solving large stochastic planning problems using multiple dynamic abstractions*. PhD thesis, Massachusetts Institute of Technology, 2005.

[27] D. Abel, D. E. Hershkowitz, and M. L. Littman. Near optimal behavior via approximate state abstraction. *arXiv preprint arXiv:1701.04113*, 2017.

[28] J. C. Bean, J. R. Birge, and R. L. Smith. Aggregation in dynamic programming. *Operations Research*, 35(2):215–220, 1987.

[29] D. P. Bertsekas, D. A. Castanon, et al. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, 1988.

[30] N. Jiang, A. Kulesza, and S. Singh. Abstraction selection in model-based reinforcement learning. In *International Conference on Machine Learning*, pages 179–188, 2015.

[31] G. Konidaris. Constructing abstraction hierarchies using a skill-symbol loop. In *IJCAI: proceedings of the conference*, volume 2016, page 1648. NIH Public Access, 2016.

[32] N. L. Zhang and D. Poole. On the role of context-specific independence in probabilistic inference. In *IJCAI*, volume 1, page 9, 1999.

[33] D. L. Poole. Context-specific approximation in probabilistic inference. *arXiv preprint arXiv:1301.7408*, 2013.

[34] C. Domshlak and S. E. Shimony. Efficient probabilistic reasoning in bns with mutual exclusion and context-specific independence. *International journal of intelligent systems*, 19(8):703–725, 2004.

[35] J. Hoey, R. St-Aubin, A. J. Hu, and C. Boutilier. Spudd: Stochastic planning using decision diagrams. In *UAI*, 1999.

[36] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[37] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[38] J. Hoffmann. FF: The fast-forward planning system. *AI magazine*, 22(3):57–57, 2001.

[39] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A Pseudocode: Approximate Context-Specific Independence Learning

The following pseudocode describes our algorithm for learning approximate context-specific independences (CSIs). See Figure 5B for an example output, and see (§4.1) for discussion.

Input: State and action variables $V = \{S^1, \ldots, S^n\} \cup \{A^1, \ldots, A^m\}$
Input: Black-box transition model T
Input: Context $(C, \mathcal{C})$
Input: Number of samples k_1, k_2 `// Hyperparameters`
Returns: Approximate CSIs $\{(V^i, V^j) : V^j_{t+1} \perp\!\!\!\perp V^i_t \mid (C, \mathcal{C})\}$, for arbitrary t
`// Initialize all pairs of variables to be independent`
Initialize: CSIs $\leftarrow V \times V$
`// Sample` k_1 `state and action assignments in the context`
$U \leftarrow$ SAMPLEINCONTEXT$(V, C, \mathcal{C}, k_1)$
`// Test pairs of variables for dependence`
for $V^i, V^j \in V \setminus C$ **do**
  for $u \in U$ **do**
    for *up to* k_2 *samples* v^i *of* V^i **do**
      if $T(V^j_{t+1} \mid V_t = u) \neq T(V^j_{t+1} \mid V_t \setminus \{V^i_t\} = u_{-i}, V^i_t = v^i)$ **then**
        `//` V^j `is dependent on` V^i`; remove this pair from CSIs`
        CSIs $\leftarrow$ CSIs $\setminus \{(V^i, V^j)\}$
return CSIs

B Pseudocode: Context Selector Learning

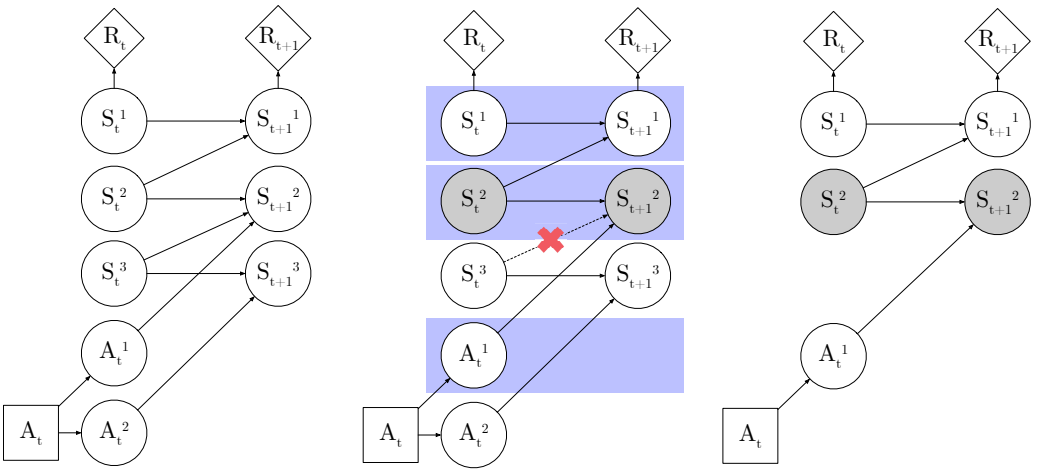
The following pseudocode describes our algorithm for learning a context selector model, given training tasks and their context-specific independences (CSIs). See (§4.2) for discussion.

Input: Training tasks $W_{\text{train}} = \{\omega^{(i)}\}_{i=1}^N$ with features $\{\theta_{\omega^{(i)}}\}_{i=1}^N$
Input: Black-box transition model T
Input: Set of contexts $\{(C, \mathcal{C})\}$
Input: All learned CSIs $(C, \mathcal{C}) \to \{(V^i, V^j) : V^j_{t+1} \perp\!\!\!\perp V^i_t \mid (C, \mathcal{C})\}$
Returns: Context selector $f_\alpha(\theta_\omega) = (C, \tilde{\mathcal{C}})^*$ `// Neural network with parameters` α
Initialize: Inputs for supervised learning $X \leftarrow [\theta_{\omega^{(1)}}, \ldots, \theta_{\omega^{(N)}}]$
Initialize: Targets for supervised learning $Y \leftarrow []$
for $\omega^{(i)} \in W_{train}$ **do**
  `// See Subroutine below`
  $Y[i] \leftarrow \text{argmax}_{(C,\mathcal{C})}$ SCORECONTEXT$(\omega^{(i)}, (C, \mathcal{C}), T,$ CSIs for $(C, \mathcal{C}))$
`// Perform supervised learning (multiclass classification)`
$\alpha^* \leftarrow \text{argmin}_\alpha$ CROSSENTROPYLOSS$(X, Y; \alpha)$
return f_{α^*}

Subroutine SCORECONTEXT
  Input: Training task $\omega = (s_0, R)$
  Input: Black-box transition model T
  Input: Context $(C, \mathcal{C})$
  Input: Learned CSIs $\{(V^i, V^j) : V^j_{t+1} \perp\!\!\!\perp V^i_t \mid (C, \mathcal{C})\}$
  Returns: A score
  $\mathcal{M}' \leftarrow$ CREATECAMP$(T, R, (C, \mathcal{C}),$ CSIs$)$ `// See (§3)`
  $\pi(s) \triangleq$ PLAN$(\mathcal{M}', s)$ `// Plan in the CAMP`
  return $J(\pi, \omega)$ `// See Equation 1`

C CAMP Graphical Example

Figure 5 provides an example of a CAMP. Note that standard influence diagrams [36] cannot capture context-specific independence, so we use a dotted line in the second panel to denote this concept.



(A) Original MDP
all variables relevant

(B) MDP with Imposed Context
A^2 and S^3 now irrelevant

(C) Context-Specific Abstract MDP
irrelevant variables dropped

Figure 5: (A) Example of a factored MDP represented as an influence diagram [36]. As seen in the diagram, $S_{\text{rew}} = \{S^1\}$. With no contexts imposed, all variables are relevant. (B) Imposing contexts can induce new independences. In this example, a context involving S^2 is imposed, inducing an independence between S^2 and S^3 (red $\times$). Variables A^2 and S^3 are irrelevant under the imposed context; relevant variables are highlighted in blue. Note that relevance is a time-independent property. (C) Dropping the irrelevant variables leads to a CAMP, an abstraction of the original MDP.

D Domain and Planner Details

Domain D1: Gridworld. The first domain we consider is a simple maze-style gridworld in which the agent must navigate across rooms to reach a goal location, while avoiding obstacles that stochastically move around at each timestep. The agent has available to it REMOVE(OBJ) actions, which remove the given obstacle from the world so that the agent can no longer collide with it, but these actions can only be used when the agent is adjacent to the obstacle. Whenever the agent collides with an obstacle, it is placed back in its initial location. Each obstacle remains within a particular room, and so the agent can impose a context of not entering particular rooms, allowing it to ignore the obstacles that are in those rooms, and also not have to consider the action of removing those obstacles. Across tasks, we vary the maze layout. We train on 50 task instances and test on 10 held-out instances.

Planners. We consider the following planners for this domain: Monte Carlo tree search (MCTS), breadth-first graph search with replanning (BFSReplan), and asynchronous value iteration (VI). Both MCTS and BFSReplan are online planners, while VI is offline. As such, VI computes a policy over the full state space, and thus is only tractable in this relatively small (about 100,000 states) domain.

Representations. The features of each task are a top-down image of the maze layout. The state is a vector of the current position and room of each obstacle, the agent, and the goal. The actions are moving up, down, left, right; and removing each obstacle in the environment.

Domain D2: Classical planning. We next consider a deterministic classical planning domain in which an agent must make a meal for dinner, and has three options: to stay within the living room to make ramen, to go to the kitchen to make a sandwich, or to go to the store to buy and prepare

a steak. Making any of these terminates the task. The steak gives higher terminal reward than the sandwich, which in turn gives higher terminal reward than the ramen. However, planning to go to the store for steak requires reasoning about many objects that would be irrelevant under the context of staying within the home (for a sandwich or ramen), and planning to go to the kitchen for a sandwich requires reasoning about many objects that would be irrelevant under the context of staying within the living room (for ramen). There is also a timestep penalty, incentivizing the agent to finish quickly. Optimal plans may involve 2, 16, or 22 actions depending on the relative rewards for obtaining the ramen, sandwich, and steak. These rewards are the only thing that varies between task instances; there is thus small variation between task instances relative to the other domains. We train on 20 task instances and test on 25 held-out instances.

Planner. We use an off-the-shelf classical planner (Fast-Downward [8] in A* mode with the *lmcut* heuristic). The various rewards are implemented as action costs. As this domain is deterministic, we only run the planner once per task; it is guaranteed to find a reward-maximizing trajectory.

Representations. The features of each task are a vector of the terminal rewards for each meal. The state is a binary vector describing which logical fluents hold true (1) versus false (0). The actions are logical operators described in PDDL, each containing parameters, preconditions, and effects.

We also use this domain as a testbed for additional experiments into the impact of λ (the trade-off parameter in Equation 1), and the number of training tasks on our method. See (§5.4) and (§F).

Domain D3: Robotic navigation among movable obstacles (NAMO). Illustrated in Figure 1A, this domain has a robot navigating through rooms with the goal of reaching the red object in the upper-right room. Roughly 20 blue obstacles are scattered throughout the rooms, and like in the gridworld, the robot may impose the context of not entering particular rooms; it may also pick up the obstacles and move them out of its way. Across tasks, we vary the positions of all objects. We train on 50 task instances and test on 10 held-out instances. This domain has continuous states and actions, and as such is extremely challenging for planning. Though the obstacles do not move on their own (like they do in the gridworld), the difficulty of this domain stems from the added complexity of needing to reason about geometry and continuous trajectories. We simulate this domain using PyBullet [20]. The reward function is sparse: 1000 if the goal location is reached and 0 otherwise.

Planner. Developing planners for robotic domains with continuous states and actions is an active area of research. For this domain, we use a state-of-the-art task and motion planner [9], which is *not* specific to NAMO problems. We use the RRT-Connect algorithm [37] for motion planning and the Fast-Forward PDDL planner [38] for task planning.

Representations. The features of each task are a top-down image of the scene. The state is a vector of the current pose of each object and the robot, and the robot's current room. The actions are moving the robot base to a target pose, and clearing an object in front of the robot.

Domain D4: Robotic sequential manipulation. Illustrated in Figure 1C, this domain has a robot manipulating the two red objects that start off on the left table to be placed into the bins on the right table. The fifteen blue objects on the left table serve as distractors, with which the robot must be careful not to collide when grasping the red objects; the green objects in the bins indicate that certain bins are already occupied. Across tasks, we vary the positions of all objects, and which bins are occupied by green objects. We also vary the radii of the red objects. We train on 50 task instances and test on 10 held-out instances. We again simulate this domain using PyBullet [20]. As in Domain D3, the reward function is sparse: 1000 if the goal location is reached and 0 otherwise.

Broadly, there are two types of contexts that are useful to impose in this domain. (1) If the robot chooses to constrain its *grasp style* to only allow top-grasping the red objects, then it need not worry about colliding with the blue objects, and can thus ignore them. However, this does not always work, since not all geometries are amenable to being top-grasped; for instance, sometimes an object's radius may be too large. Note, however, that to place the red objects into the bins upright, a side-grasp is necessary, and so we provide the robot a *regrasp* operator in addition to the standard move, pick, and place. Importantly, this regrasp operator is never *necessary*, but including it can allow the robot to simplify its planning problem by ignoring the blue objects (see Equation 1). (2) If the robot chooses to constrain which bins it will place the red objects into, then it need not worry about the green objects in the other bins, simplifying the planning problem.

Planner. Same as in Domain D3 (NAMO).

Representations. The features of each task are a vector of the object radii and occupied bins. The state is a vector of the current pose of each object, the grasp style used by the robot, and the current held object (if any). The actions are moving the robot to a target base pose and grasping at a target gripper pose (which requires an empty gripper), and moving the robot to a target base pose and placing at a target placement pose (which requires an object to be currently held).

E Experimental Details

In all experiments, computational cost is measured in wall-clock time (seconds). We use the following values of λ: 0 for MCTS[4], 100 for BFSReplan, 250 for FastDownward, and 100 for TAMP. Every domain uses horizon $H = 25$. Additionally, to ensure that shorter plans are preferred in general, all domains use a discount factor $\gamma = 0.99$, except for Domain D2 which uses a timestep penalty as previously discussed.

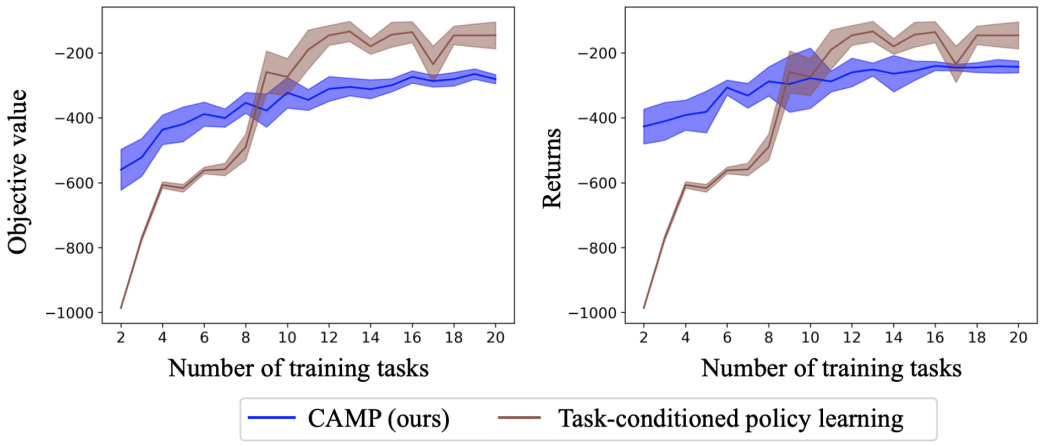
To properly evaluate our objective (Equation 1), we would need to run every method until it completes, which can be extremely slow, e.g. for the pure planning baseline, or when our context selector picks a bad context. To safeguard against this, we impose a timeout of 60 seconds on all planning calls.

All neural networks are either fully connected for vector inputs or convolutional for image inputs. Fully connected networks have hidden layer sizes [50, 32, 10]. Convolutional networks use a convolutional layer with 10 output channels and kernel size 2, followed by a max-pooling layer with stride 2, and then fully connected layers of [32, 10]. Neural networks are trained using the Adam optimizer [39] with learning rate 10^{-4}, until the loss on the training dataset reaches 10^{-3}.

To generate the spaces of contexts, we use the method described in (§4.1). In Domains D1, D2, and D3, we consider disjunctive and single-term (a single variable and a single value in its domain) constraints only, while in Domain D4 we also consider conjunctive constraints. All contexts only consider the discrete variables in the domain. Our parameters k_1 and k_2 (Appendix A) are: $k_1 = k_2 = 50$ for Domain D1, $k_1 = k_2 = 40$ for Domain D2, and $k_1 = k_2 = 25$ for Domains D3 and D4.

F Performance as a Function of Number of Training Tasks

The following plots illustrate how the objective value (left) and returns (right) accrued by the CAMP policy vary as a function of the number of training tasks, in Domain D2 (classical planning):



Discussion. As following a policy in the test task requires near-zero computational effort (our neural networks are small enough that inference is very fast), the red curves in both plots are nearly identical. Interestingly, in the regime of fewer training tasks (≤ 8), CAMP outperforms policy learning, despite policy learning performing better with the full set of 20 tasks. This leads us to believe that in domains where policy learning would perform well when given a lot of data, generating and planning in

[4]Since MCTS is an anytime algorithm, we give it a timeout of 0.25 seconds. With $\lambda = 0$, the objective then reflects the best returns found within this timeout.

Method	Test Task Objective Value (St. Dev.)				
	D1 (Grid), MCTS	D1 (Grid), BFSReplan	D2 (Classical)	D3 (NAMO)	D4 (Manip)
CAMP (ours)	**70** (16)	**21** (10)	-286 (9.6)	**896** (63)	**744** (94)
CAMP ablation	25 (11)	0.9 (24)	-308 (52)	707 (154)	453 (237)
Pure planning	6 (5)	-17 (11)	-414 (20)	242 (385)	335 (86)
Plan transfer	-7 (0.4)	-6 (15)	-467 (0.02)	141 (227)	21 (34)
Policy learning	-3 (4)	-11 (13)	-469 (0.2)	-0.2 (0.01)	-0.2 (0.01)
Task-conditioned	5 (5)	-2 (11)	**-145** (0.4)	-0.3 (0.01)	-0.2 (0.02)
Stilman's [11]	-	-	-	826 (36)	-

Table 1: Compilation of test task objective values on all our domains and methods. Objective values are computed using the same values of λ that were used during training. All table entries report an average over 10 independent runs of both context selector training and test task evaluation. Stilman's algorithm [11] is NAMO-specific and so is only run on the NAMO domain.

a CAMP may be a more viable strategy when data is limited. As shown in the main results, this disparity between CAMPs and policy learning is more dramatic for the other three domains, where task instances are far more varied, so much so that CAMPs sharply outperform policy learning for any reasonable number of training environments that we were able to test.

G Objective Values for Main Experiments

Table 1 complements Figure 4 in the main text, showing the objective values obtained for all domains, planners, and methods with the values of λ that were used during training. See (§5.3) for further analysis and discussion.

H Performance with an Offline Planner

The table on the right shows test task objective values with an offline planner (asynchronous value iteration), in Domain D1 (gridworld).

Discussion. This result mirrors the trend found in the main results: CAMPs strongly outperform both pure policy learning and pure planning, for reasons of generalization error and high computational cost, respectively. CAMP's reduction of the state space leads to substantial benefits for offline planning, because offline planners find a policy over the entire state space. Nonetheless, CAMP remains primarily motivated by online planning for robotics, where the continuous states and actions make offline planning completely infeasible in practice.

Method	Objective (SD)
CAMP (ours)	**25** (3)
CAMP ablation	4 (11)
Pure planning	-1 (2)
Plan transfer	-
Policy learning	7 (0.01)
Task-conditioned	7 (0.01)