Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

Is Policy Learning Overrated?: Width-Based Planning and Active Learning for Atari

Benjamin Ayton[*1], **Masataro Asai**[*2]
[1]MIT
[2]MIT-IBM Watson AI Lab
aytonb@mit.edu,masataro.asai@ibm.com

Abstract

Width-based planning has shown promising results on Atari 2600 games using pixel input, while using substantially fewer environment interactions than reinforcement learning. Recent width-based approaches have computed feature vectors for each screen using a hand designed feature set (Rollout-IW) or a variational autoencoder trained on game screens (VAE-IW), and prune screens that do not have novel features during the search. We propose Olive (Online-VAE-IW), which updates the VAE features online using active learning to maximize the utility of screens observed during planning. Experimental results across 55 Atari games demonstrate that it outperforms Rollout-IW by 42-to-11 and VAE-IW by 32-to-20. Moreover, Olive outperforms existing work based on policy-learning (π-IW, DQN) trained with 100 times the training budget by 30-to-22 and 31-to-17, and a state of the art data-efficient reinforcement learning (EfficientZero) trained with the same training budget and ran with 1.8 times the planning budget by 18-to-7 in the Atari 100k benchmark, without any policy learning. The source code and the appendix are available at github.com/ibm/atari-active-learning and arxiv.org/abs/2109.15310.

Introduction

Recent advancements in policy learning based on *Reinforcement Learning* (RL) (Sutton and Barto 2018) have made it possible to build an intelligent agent that operates within a stochastic and noisy image-based interactive environment, which has been one of the major goals of Artificial Intelligence. The Arcade Learning Environment (ALE) (Bellemare et al. 2013), which allows access to pixel and memory features of Atari games, is a popular testbed for testing modern RL approaches (Mnih et al. 2015). However, RL approaches are notorious for their poor sample efficiency; they require a huge number of interactions with the environment to learn the policy. This is especially true for sparse reward problems, such as Montezuma's Revenge, where successful cases could require 10^{10} interactions with the environment (Badia et al. 2020). Poor sample efficiency hinders real-world applicability when the number of interactions is restricted by practical constraints, such as the runtime for collecting data and the safety of the environment.

Meanwhile, the *planning setting* of the ALE provides an alternative benchmark where the focus is placed on efficient search. Significant progress in this setting has been made on Iterative-Width (IW) (Lipovetzky and Geffner 2012; Lipovetzky, Ramirez, and Geffner 2015) recently, which prunes states if they do not possess features that have not been seen during search, and has been demonstrated to outperform Monte Carlo tree search based on UCT (Kocsis and Szepesvári 2006). For example, p-IW (Shleyfman, Tuisov, and Domshlak 2016) modifies IW to maximize the reward per feature, DASA (Jinnai and Fukunaga 2017) learns and prunes actions that lead to identical states, Rollout-IW (RIW) (Bandres, Bonet, and Geffner 2018) improves anytime characteristics of IW, which revolutionalized the field with almost-real-time planning on pixel inputs, π-IW (Junyent, Jonsson, and Gómez 2019) performs informed search with a trainable policy represented by a neural network, and π-IW+ (Junyent, Gómez, and Jonsson 2021) also learns a value function.

Width-based approaches assume discrete inputs, such as the RAM state of Atari games, but can work on continuous variables using discretization (Frances et al. 2017; Teichteil-Königsbuch, Ramírez, and Lipovetzky 2020). π-IW discretizes the last hidden layer of a policy function as the input. Recently, VAE-IW (Dittadi, Drachmann, and Bolander 2021) obtains a compact binary representation by training a Binary-Concrete VAE offline.

Search-based approaches are in general not directly comparable against RL-based approaches because the latter are given significantly larger computational budgets than the former. One exception is π-IW, a hybrid approach that requires 2×10^7 interactions[1] to learn policies in Atari games. This suggests a better middle ground between one extreme (RL) that requires billions of interactions, and another (search) which does not improve in performance over time but is more sample efficient. Given that π-IW requires a large training budget, we hypothesized that learning a policy is significantly sample-inefficient. By building off of VAE-IW, which performs offline representation learning that is fixed once trained, and does not improve over time, we aim to build an agent which *does* learn over time, *without* expensive policy learning.

[*]These authors contributed equally.

[1]Updated in the Arxiv version from 4×10^7 in the ICAPS paper.

To this end, we propose Olive (Online-VAE-IW), a hybrid agent that collects new data and improves its state representation online by combining Iterative Width with *Active Learning* (Settles 2012). Active Learning is a group of approaches that optimize a data-dependent objective by exploring, selecting, and adding new observations to the dataset. Olive uses active learning in two senses: First, it improves the search with a *Multi-Armed Bandit* that balances data collection (exploration) and reward-seeking behavior (exploitation) based on expected rewards and their variances. Second, it improves the state representation with an *Uncertainty Sampling* (Settles 2012) that selects and adds novel screens between each planning episode based on the inaccuracy of the state representation. Active learning collects both reward information and screen information, but there is a non-trivial interaction between them. For example, reward-seeking behavior is also necessary for screen selection because it is not cost effective to accurately learn the representations of states that are unpromising in terms of rewards.

Our online learning agent plays the game until a specified limit on ALE simulator calls is reached. Each time the game finishes (an *episode*), the agent selects and adds a subset of the observed game screens to a dataset, then retrains a Binary Concrete VAE neural network that provides feature encodings of screens. Between each action taken in an episode, it constructs a search tree by performing rollouts, and selects the most promising action (giving the highest achievable reward in the expanded tree) at the root node using the information collected during the rollouts. States encountered during a rollout are pruned using the novelty criterion proposed by Rollout-IW. During a rollout, actions are selected by balancing the exploration and the exploitation of rewards using the Best Arm Identification algorithm called Top Two Thompson sampling (Russo 2020).

We show that after only a small number of training episodes, our approach results in markedly improved gameplay scores compared to state of the art width-based planning approaches and policy-based approaches with the same or a smaller number of simulator calls.

Background

We formalize the image-based ALE as a finite horizon MDP $P = \langle X, A, T, I, R, t^*, \gamma \rangle$, where X is a set of state observations (game screens) $\boldsymbol{x}$, A is a set of actions a, T is a deterministic transition function, $I \in X$ is an initial state, $R : X, A, X \to \mathbb{R}$ is a reward function, t^* is a planning horizon, and γ is a discount factor. A policy $\pi(\boldsymbol{x}) = \Pr(a \mid \boldsymbol{x})$ is a categorical distribution of actions given a state, and a score function $V_\pi(\boldsymbol{x})$ is the long-term reward of following π from $\boldsymbol{x}$, discounted by γ. Our task is to find an optimal policy π^* which maximizes $V^*_\pi(I)$. Formally,

$$\boldsymbol{x}_0 = I,\ \boldsymbol{x}_{t+1} = T(\boldsymbol{x}_t, a_t), \tag{1}$$

$$\pi^* = \arg\max_\pi V_\pi(I) = \arg\max_\pi \sum_{t=0}^{t^*} \gamma^t R(\boldsymbol{x}_t, a_t, \boldsymbol{x}_{t+1}).$$

In particular, we focus on a deterministic policy, i.e., a *plan*, or an action sequence $\pi^* = (a_0, \cdots a_{t^*})$, which is equivalent to $\Pr(a = a_t \mid \boldsymbol{x} = \boldsymbol{x}_t) = 1, \Pr(a \neq a_t \mid \boldsymbol{x} = \boldsymbol{x}_t) = 0$. Additionally, we define an *action-value* function

$$Q_\pi(\boldsymbol{x}, a) = R(\boldsymbol{x}, a, \boldsymbol{x}') + \gamma V_\pi(\boldsymbol{x}'),\ \boldsymbol{x}' = T(\boldsymbol{x}, a) \tag{2}$$

which satisfies $V_{\pi^*}(\boldsymbol{x}) = \max_a Q_{\pi^*}(\boldsymbol{x}, a)$. We will omit π, π^* for brevity hereafter.

Following existing work, the agent is assumed to have access to the ALE simulator, and we assign a fixed simulation budget to the planning process and force the agent to select an action each time the budget runs out. We assume that observed states can be cached, and the simulator can be set to any cached state during planning. We discuss more details on the experimental setting in Sec. .

Classical Novelty for Width-Based Planning

Width-based planning is a group of methods that exploit the *width* of a search problem, originally developed for classical planning problems, which are deterministic shortest path finding problems that are quite similar to MDPs in several aspects. To simplify the discussion, assume that each state $\boldsymbol{x} \in X$ is represented by a subset of boolean atoms F, so that the total state space is $X = 2^F$. A *(conjunctive) condition* over the states can be represented as a subset $\boldsymbol{c} \subseteq F$ where a state $\boldsymbol{x}$ *satisfies* $\boldsymbol{c}$ (denoted as $\boldsymbol{x} \models \boldsymbol{c}$) when $\boldsymbol{c} \subseteq \boldsymbol{x}$. Then a unit-cost classical planning problem can be seen as an MDP $\langle X, A, T, I, R, \infty, 1 \rangle$ where T is deterministic and, given a goal condition G, $R(\boldsymbol{x}, a, \boldsymbol{x}') = 0$ when $\boldsymbol{x}' \models G$, and $R(\boldsymbol{x}, a, \boldsymbol{x}') = -1$ otherwise. Unlike typical STRIPS/PDDL-based classical planning (Fikes, Hart, and Nilsson 1972), T could be a black-box in our setting.

Width in width-based planning (Lipovetzky and Geffner 2012) is defined as follows. Given a condition $\boldsymbol{c}$, we call the optimal plan to reach a state $\boldsymbol{x} \models \boldsymbol{c}$ to be the optimal plan to achieve $\boldsymbol{c}$. The *width* $w(\boldsymbol{c})$ of $\boldsymbol{c}$ is then the minimum size $|\boldsymbol{c}'|$ of a condition $\boldsymbol{c}'$ such that every optimal plan $\pi^* = (a_0, a_1 \ldots, a_N)$ to achieve $\boldsymbol{c}'$ is also an optimal plan to achieve $\boldsymbol{c}$, and that any prefix subsequence $(a_0, \ldots, a_n)$ of π^* ($n \leq N$) is also an optimal plan for some condition of size $|\boldsymbol{c}'|$. The width of a classical planning problem is then $w(G)$, the width of the goal condition. While the definition can be easily extended to categorical variables instead of boolean variables, as seen in SAS+ formalism (Bäckström and Nebel 1995), we focus on boolean variables as they are easily interchangeable.

Lipovetzky and Geffner (2012) found that, in a majority of planning problems in International Planning Competition (López, Celorrio, and Olaya 2015) benchmarks, each single goal atom $g \in G$ seen as a condition $\{g\}$ tends to have a small width, typically below 2. Based on this finding, they proposed *Iterative Width* (IW), a highly effective blind search algorithm that is driven by the intuition that the *original* problem also has a small width. IW performs a series of iterations over the width w in increasing order. The w-th iteration, denoted as IW(w), performs a breadth-first search that enumerates optimal plans for every condition whose width is less than or equal to w. IW(w) is guaranteed to solve a problem whose width is below w. While it runs in time exponential in w, low-w iterations run quickly and efficiently by keeping track of a *novelty* metric of each search state and

pruning the states with novelty larger than w. A state $\boldsymbol{x}$ is called *novel* with regard to w if there exists a condition $\boldsymbol{c}$ of size w such that $\boldsymbol{x} \models \boldsymbol{c}$, and no other state $\boldsymbol{x}'$ that has been seen so far satisfies $\boldsymbol{c}$. The novelty of a state is then defined as the minimum w for which it is novel.

Novelty can be used like a heuristic function (Lipovetzky and Geffner 2017) in traditional search algorithms such as Greedy Best First Search (Hoffmann and Nebel 2001). However, unlike traditional heuristics it does not utilize the transition model or the goal condition, and its value is affected by the set of states already expanded by the search algorithm. Let C be a CLOSE list; a database of states expanded during the search and their auxiliary information. We denote the novelty of state $\boldsymbol{x}$ by $n(\boldsymbol{x}, C)$ to emphasize the fact that it depends on the CLOSE list. Initially, $C = \emptyset$, but new states are added to C as more states are expanded. To determine whether a new state $\boldsymbol{x}$ is novel, check whether there is a condition $\boldsymbol{c}$ satisfied by $\boldsymbol{x}$ and not satisfied by any state in C. To implement a fast lookup, C is implemented with a set of conditions that are already achieved. IW(w) terminates when C contains all conditions of size w expressible under F.

Rollout-IW (RIW)

Width-based planning has been successfully applied to MDPs outside of classical planning. A boolean encoding $\boldsymbol{z} \in \{0,1\}^F$ is computed for each state, where each element z_j of the encoding indicates that the j-th atom in F is present in the state if and only if $z_j = 1$. A state is pruned if its novelty is larger than a specified width, and the policy with highest reward in the pruned state space is determined.

To apply width-based planning to Atari games with states derived from screen pixels, Bandres, Bonet, and Geffner (2018) proposed to compute $\boldsymbol{z}$ from a vector of pixels $\boldsymbol{x} \in X$ using the B-PROST feature set (Liang et al. 2016). However, even with width 1, the state space is too large to plan over using breadth first search under the time budgets consistent with real-time planning. To overcome this issue, they proposed Rollout-IW, which improves the anytime behavior of IW by replacing the breadth-first search with depth-first rollouts. They define a CLOSE list C^d specific to each tree depth d and use a depth-specific novelty $n(\boldsymbol{x}, C^{\leq D})$ for state $\boldsymbol{x}$ at depth D. The depth-specific novelty is the smallest w such that for some condition $\boldsymbol{c}$ with $|\boldsymbol{c}| = w$, $\boldsymbol{x} \models \boldsymbol{c}$ and $\boldsymbol{c}$ is not satisfied by any state in C^d with $d \leq D$. Rollouts are performed from the tree root, selecting actions uniformly at random until a non-novel state is reached. Novelty is reevaluated for states each time they are reached, since CLOSE lists at lower depths may change between different rollouts, and a state is pruned if it is ever found to not be novel.

VAE-IW

VAE-IW (Dittadi, Drachmann, and Bolander 2021) extends RIW to learn encodings from screen images. In a training stage, the game is run using RIW until a fixed number of screens are encountered and saved. A Binary-Concrete Variational Autoencoder (Jang, Gu, and Poole 2017; Maddison, Mnih, and Teh 2017) is trained on the saved data to produce a binary encoding vector $\boldsymbol{z} \in \{0,1\}^F$ from screen pixels, following work on learning a PDDL representation of image-based inputs (Asai and Fukunaga 2018). After the training, the game is played using RIW, with features obtained by the encoder.

A VAE consists of an encoder network that returns $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$, a decoder network that returns $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$, and a prior distribution $p(\boldsymbol{z})$. The VAE is trained to maximize the *evidence lower bound* (ELBO) of the saved screens, computed as

$$\log p_\theta(\boldsymbol{x}) \geq \text{ELBO}(\boldsymbol{x}) \tag{3}$$
$$= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\langle \log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\rangle - D_{\text{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{z})).$$

To obtain binary latent vectors, the latent variables are assumed to follow component-wise independent Bernoulli distributions.

$$p(\boldsymbol{z}) = \prod_{j=1}^{F} \text{Bern}(0.5), \quad q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) = \prod_{j=1}^{F} \text{Bern}(\mu_j) \tag{4}$$

where $\boldsymbol{\mu}$ are Bernoulli parameters that are obtained as the sigmoid of the output of the encoder network. To obtain deterministic features for planning, $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ is thresholded, using $z_j = 1$ if $\mu_j > 0.9$ and 0 otherwise.

Online Representation Learning for Atari

Learning the features from data permits the encoding to be tailored to a specific game, but generating a faithful encoding of a game is non-trivial, and using a static dataset is typically insufficient. For example, Dittadi, Drachmann, and Bolander (2021) save screens that are reached by a Rollout-IW agent using a hand-coded B-PROST feature set, and use those screens to train a Binary-Concrete VAE to produce a game-specific encoding. However, in order for the encoding to be representative of a game, the dataset must include screens from all visually distinct parts, such as separate levels. When the B-PROST/Rollout-IW agent lacks a degree of competence to reach level 2 and beyond, screens from the later levels are never included in the dataset, thus VAE-IW cannot perform well in later levels.

To build an efficient online planning agent that simultaneously learns the representation, we must tackle three challenges: **(1)** *How to automatically collect a diverse set of screens in the dataset*, which requires a metric that quantifies the diversity of screens. **(2)** *How to focus on improving the representations of states that return high rewards*. This is because learning the representation of the states with low rewards may not be worthwhile because the resulting agents will ultimately avoid such states. Finally, **(3)** *How to keep the dataset size small.* If the dataset gets too large, the computational effort for retraining between episodes becomes prohibitive.

To tackle these challenges, we propose *Olive* (Online-VAE-IW) (Alg. 1), an online planning and learning agent that performs dynamic dataset refinement and retraining of the VAE between each search episode (line 4-5). It addresses the challenges above from three aspects: **(1)** Pruning based on novelty, **(2)** search guided by Bayesian estimates of future rewards, and **(3)** active learning based on uncertainty sampling in games that change visually as the game progresses. Olive consists of 3 nested loops:

1. *Episode* (line 8): A period which is started and ended by a game reset. Each reset could be triggered by a limit on the maximum number of actions in an episode, a limit on simulator calls (training phase only), or an in-game mechanism (e.g., beating the game, running out of lives). During training, each episode is followed by augmenting the dataset with k screens observed in the episode and retraining the VAE. When evaluating Olive's performance after training, no additional screens are added.
2. *Planning and Acting* (line 13): A period in which the agent makes a decision by searching the state space using a simulator. It ends by hitting a limit on the maximum number of simulator calls for the period. When it reaches the limit, the agent performs an action irreversibly. The agent chooses the action that achieved the largest reward $Q_i(\boldsymbol{x}, a)$ on any rollout i from the root node (line 17), and then truncates the search tree at the new root.
3. *Rollout* (line 18): A lookahead that collects information using a black-box simulator. States are pruned if they are not novel (line 23). Actions are chosen by *Best Arm Identification* (BAI) algorithms (line 24), based on statistics from previous rollouts. Each recursion into a deeper rollout obtains a new empirical Q-value (line 27-28) and updates the statistics (line 29-30).

Prior Distributions for Reward

Let $Q(\boldsymbol{x}, a)$ be a random variable for the cumulative reward of a rollout from $\boldsymbol{x}$ starting with an action a, and $Q_i(\boldsymbol{x}, a)$ be the cumulative reward obtained from the i-th rollout launched from $\boldsymbol{x}$ with a. Assume that we have launched n rollouts from $\boldsymbol{x}$ with a so far, and we have a collection of data $\{Q_1(\boldsymbol{x}, a), \dots, Q_n(\boldsymbol{x}, a)\}$. We define a hierarchical Bayesian model where $p(Q(\boldsymbol{x}, a))$ is a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ with an unknown mean μ and a variance σ^2, which are further modeled as follows (Gelman et al. 1995):[2]

$$Q(\boldsymbol{x}, a) \mid \mu, \sigma^2 \sim \mathcal{N}(\mu, \sigma^2) \tag{5}$$

$$\mu \mid \sigma^2 \sim \mathcal{N}(\bar{\mu}, \sigma^2/n) \tag{6}$$

$$\sigma^2 \sim \text{Scaled-Inv}\chi^2(n+1, \bar{\sigma}^2) \tag{7}$$

where $\bar{\mu}$ is the empirical mean $\bar{\mu} = \frac{1}{n}\sum_{i=1}^{n} Q_i(\boldsymbol{x}, a)$ of the data, and $\bar{\sigma}^2$ is a *regularized* variance $\bar{\sigma}^2 = \frac{1}{n+1}(\sigma_0 + \sum_i (Q_i(\boldsymbol{x}, a) - \bar{\mu})^2)$ where we use $\sigma_0 = 0.2$ as a prior (hyperparameter), Scaled-Invχ^2 is a Scaled Inverse χ^2 distribution. Initializing $\bar{\sigma}^2$ to a non-zero value σ_0 when no rollouts are made ($n = 0$) and using $n + 1$ as the divisor instead of n is called a pseudo-count method, which avoids *zero-frequency problems*, i.e., when it is the first time to launch a rollout from the current node, we avoid computing Scaled-Invχ^2 with $\bar{\sigma}^2 = 0$, which is undefined.

This Bayesian statistical modeling can properly account for the uncertainty of the observed $Q(\boldsymbol{x}, a)$. The key idea is that the true mean/variance of $Q(\boldsymbol{x}, a)$ differs from the empirical mean/variance of $Q(\boldsymbol{x}, a)$ observed during finite rollouts. Since we do not know the true mean/variance, we use distributions to model which values of mean/variance are more likely. This distribution becomes more precise as more observations are made, e.g., the variance σ^2/n of μ decreases as the divisor n increases (more observations).

[2]This particular model is equivalent to a *Normal-Gamma* distribution in other literature.

Algorithm 1: Training and evaluation of Olive

```
1: procedure OLIVE-EXPERIMENT(X, A, T, I, R)
2:     Initialize VAE, dataset 𝒳 ← ∅
3:     for ep = 1, ..., N_ep do                          ▷ Training loop
4:         𝒳 ← 𝒳 ∪ {k screens observed in EPISODE}
5:         Retrain VAE with 𝒳
6:     for ep = 1, ..., 10 do                            ▷ Evaluation loop
7:         EPISODE
8: procedure EPISODE
9:     x_root ← RESETGAME
10:    while x_root is not terminal do                   ▷ e.g., until dead
11:        x_root ← PLANANDACT(x_root)
12:    return observed screens
13: procedure PLANANDACT(x_root)
14:    Initialize CLOSE list {C^0 ...}
15:    while budget remaining do                         ▷ Planning
16:        ROLLOUT(x_root, [], 0)
17:    return T(x_root, arg max_a max_i Q_i(x_root, a))
18: procedure ROLLOUT(x, π, d)                           ▷ state, path, depth
19:    z ← SIGMOID(ENCODE(x)) > 0.9
20:    if Parameters[π] is not present:                  ▷ new node
21:        Update C^d using x, initialize Parameters[π]
22:    if x is terminal | budget exceeded | n(x, C^{≤d}) > w:
23:        return 0
24:    a ← Best Arm Identification(Parameters)
25:    x' ← T(x, a), π' ← π + [a]
26:    (n, μ̄, σ̄²) ← Parameters[π']
27:    Q_{n+1}(x, a) ←
28:        R(x, a, x') + γ ROLLOUT(x', π', d + 1)
29:    μ̄' ← (nμ̄ + Q_{n+1}(x, a)) / (n+1)
30:    σ̄²' ← ((n+1)σ̄²)/(n+2) + ((Q_{n+1}(x, a) − μ̄)(Q_{n+1}(x, a) − μ̄'))/(n+2)
31:    Parameters[π'] ← (n + 1, μ̄', σ̄²')
32:    return Q_{n+1}(x, a)
```

To incorporate this Bayesian model, it is only necessary to store and update $n, \bar{\mu}, \bar{\sigma}^2$ incrementally on each search node (line 29-30) without storing the sequence $\{Q_1(\boldsymbol{x}, a), \dots, Q_n(\boldsymbol{x}, a)\}$ explicitly. See the appendix for the proof of these updates. Note the $n + 2$ being used as the divisor for the variance due to the pseudo-count, which adds 1 to the number of samples. In all our experiments, we initialize the parameters to $n = 0, \bar{\mu} = 0, \bar{\sigma}^2 = 0.2$. We refer to this set of parameters as $\left(n, \bar{\mu}, \bar{\sigma}^2\right) = \text{Parameters}[\pi']$, where π is an action sequence that reached $\boldsymbol{x}$ and $\pi' = \pi + [a]$.

Although Atari environments are deterministic, note that $Q(\boldsymbol{x}, a)$ still have uncertainty due to the depth-first algorithm which does not systematically search all successors.

Best Arm Identification

While Rollout-IW and VAE-IW have focused on uniform action selection within the pruned state space, we recognize

that more advanced action selection is fully compatible with width-based planning. We use the distributions over Q to apply a Best-Arm Identification (BAI) strategy at each node (Audibert, Bubeck, and Munos 2010; Karnin, Koren, and Somekh 2013).

BAI is a subclass of the multi-armed bandit (MAB) problem (Gittins, Glazebrook, and Weber 2011); a sequential decision making problem in which many unknown distributions (levers) are present, and one is selected to be sampled (pulled) from at each decision. MABs have frequently been studied with a minimum cumulative regret (CR) objective, which aims to maximize the expected reward of the samples *obtained during the experimentation*. For example, UCB1 (Auer, Cesa-Bianchi, and Fischer 2002) is the theoretical basis of the widely used UCT algorithm for MCTS (Kocsis and Szepesvári 2006), and is derived under a CR objective. In contrast to CR, BAI draws samples to minimize the *simple regret*, i.e., to maximize the expected rewards of a single decision (acting) made *after* the experimentation. BAI is thus more appropriate for our problem setting, as we wish to maximize the performance of acting and not the cumulative performance of rollouts. A number of works have recognized BAI as a more natural objective for tree search and have proposed algorithms based on BAI (Cazenave 2014; Kaufmann and Koolen 2017) or hybrids of BAI and cumulative regret minimization (Pepels et al. 2014). To our knowledge, these search procedures have not been applied within a state space pruned by width-based planning. Shleyfman, Tuisov, and Domshlak (2016) also point out that simple regret is better suited for this purpose. However, their Racing p-IW algorithm is not explicitly based on BAI.

In this paper, we consider Top-Two Thompson Sampling (Russo 2020, TTTS) as a BAI algorithm, and additionally consider UCB1, uniform random action selection, and maximum empirical mean for reference. Note that Rollout-IW and VAE-IW use uniform sampling for rollouts.

Top-Two Thompson Sampling (TTTS) is an algorithm that is based on Thompson Sampling (TS). TS runs as follows: For each a, we first sample σ^2 (Eq. 7). Next, using n, $\bar{\mu}$, and the sampled value of σ^2, we sample μ (Eq. 6). We similarly sample $Q(\boldsymbol{x}, a)$ (Eq. 5). TS returns $\hat{a} = \arg\max_a Q(\boldsymbol{x}, a)$ as the best action. TTTS modifies the action returned by TS with probability α (a hyperparameter, 0.5 in our experiments). TTTS discards action $\hat{a}$, then continues running TS until it selects a different action $a \neq \hat{a}$, which is returned as a solution. TTTS is shown to provide a better exploration than TS. Since we use an improper prior at $n = 0$, we first sample each action once at a new node.

Upper Confidence Bound (UCB1) selects the action which maximizes the metric $\bar{\mu}_a + \sqrt{2 \log N / n_a}$ where $\bar{\mu}_a$ is the statistic $\bar{\mu}$ for action a, n_a is the count n for a, and $N = \sum_a n_a$ is the sum of counts across actions.

Updating the Screen Dataset

Finally, we update the dataset $\mathcal{X}$ by adding a fixed number k of new observations after each episode. In principle, all screens that are observed during an episode could be added to a dataset to further train the VAE. However, this approach quickly exhausts the physical memory and prolongs the training time between episodes.

We address this issue through two strategies. The first approach is to select the k screens at random from those observed during the episode. This approach is more likely to include screens from later in the game as the agent improves, incorporating visual changes in the game into the learned representation. We refer to this approach as *Passive Olive*.

However, most of the new data are duplicates, or are quite close to data points already in the dataset. For example, the initial state and states nearby may be added to the dataset multiple times, and are likely to have similar appearances. As a result, the dataset contains more screens for the states near the initial states than those for the states deep down the search tree, resulting in an *imbalanced dataset*. Not only do these duplicate screens lack new information for the VAE to learn, but the imbalance also potentially prevents VAEs from learning new features, because machine learning models assume i.i.d. samples and tend to ignore rare instances in an imbalanced dataset (Wallace et al. 2011).

To tackle this challenge we employ *Uncertainty Sampling* (Settles 2012), a simple active learning strategy which selects data that the current model is most uncertain about. Retraining the model with those data points is expected to improve the accuracy of the model for those newly added inputs. In uncertainty sampling, we select k screens $\mathcal{X}_{\text{chosen}}$ from among the newly collected screens $\mathcal{X}_{\text{new}}$ with the lowest total probability under the current model:

$$\underset{\mathcal{X}_{\text{chosen}} \subset \mathcal{X}_{\text{new}}, |\mathcal{X}_{\text{chosen}}| = k}{\arg\min} \sum_{\boldsymbol{x} \in \mathcal{X}_{\text{chosen}}} \log p_\theta(\boldsymbol{x}).$$

In Olive, $p_\theta(\boldsymbol{x})$ represents the VAE. Since $p_\theta(\boldsymbol{x})$ is unknown, we approximate it with its lower bound, i.e., the ELBO that is also the loss function of the VAE (Eq. 3). Therefore, given a new set of observations $\mathcal{X}_{\text{new}}$, we compute their loss values using the VAE obtained in the previous iteration, then select the screens with the top-k highest loss values (minimum ELBO) as $\mathcal{X}_{\text{chosen}}$. Images with high loss values are images that the VAE is uncertain about, and which will be improved when added to the training set. We call the resulting configuration of Olive as *Active Olive*.

Empirical Evaluations

We evaluated Olive and existing approaches on a compute cluster running an AMD EPYC 7742 processor with nVidia Tesla A100 GPUs. We used 55 Atari 2600 games with a problem-dependent number of actions (Bellemare et al. 2013, the *minimal action* configuration) and the risk-averse reward setting (Bandres, Bonet, and Geffner 2018). For every configuration, we trained the system with 5 different random seeds, then evaluated the result for 10 episodes (playthroughs) for 50 evaluations total. Following existing work (Junyent, Jonsson, and Gómez 2019; Junyent, Gómez, and Jonsson 2021; Dittadi, Drachmann, and Bolander 2021), we use a discount factor of $\gamma = 0.99$ (rather than 0.995 in (Bandres, Bonet, and Geffner 2018)) in line 28. However, note that the reported final scores are undiscounted sums of rewards.

Resource Constraints: We repeated the setup of previous work (Lipovetzky and Geffner 2012; Bandres, Bonet,

and Geffner 2018; Dittadi, Drachmann, and Bolander 2021) where each action is held for 15 frames, which we refer to as taking a single action. We define a single *simulator call* as an update to the simulator when taking a single action. Improving the learned state representation requires screens to be observed by agents with increasing performance that can, for example, reach further in the game and experience later levels. In order to allow Olive to play a larger number of episodes during training, we limit the number of actions in each episode to 200. During evaluation, an episode ends if the game is not already finished once 18000 actions have been performed from the initial state (because some games can be played indefinitely).

To evaluate the sample efficiency, we limit the total number of simulator calls allowed for the training, or the *total training budget*. Approaches that contain machine learning with IW include VAE-IW, π-IW, and HIW (Dittadi, Drachmann, and Bolander 2021; Junyent, Jonsson, and Gómez 2019; Junyent, Gómez, and Jonsson 2021), but they use widely different budgets. The latter two use 4×10^7 or 2×10^7 simulator calls, while VAE-IW does not specify such a limit due to how they collect the data. VAE-IW collects 15000 images in total by indefinitely playing the game with B-PROST features, and randomly sampling 5 images in the search tree between taking each action. To normalize by budget, we modify VAE-IW to stop when it exhausts the budget, and sample $k = 15000$ images out of all past observed images using reservoir sampling. The total budget is 10^5 simulator calls, which is adequate because we use only 15000 images in total. The VAE is trained for 100 epochs, following the README file of the public source code of VAE-IW (not specified in the paper).

Olive, in contrast, plays the first episode with B-PROST features and collects $k = 500$ images from the observed screens. After the first iteration, it extends the dataset by $k = 500$ images in each episode, but by performing Rollout-IW using the features learned at the end of the previous episode. If the training budget is exhausted before completing 30 episodes (15000 images), the remaining images are sampled from the entire set of screens seen in the final episode.

Finally, to achieve "almost real-time" performance proposed in (Bandres, Bonet, and Geffner 2018), we limit the computational resources that can be spent between taking each action, or the "budget". An issue with the "almost real-time" setting of some existing work is that the budget between each action is limited by runtime. This lacks reproducibility because the score is affected by the performance of the compute hardware. Following (Junyent, Jonsson, and Gómez 2019; Junyent, Gómez, and Jonsson 2021), we instead limit the budget by the number of simulator calls, which is set to 100.

Evaluation Criteria: Since scores in Atari domains have varying magnitudes, we cannot compare them in different environments directly. Instead, we count the number of "wins/losses" between configurations. The wins are the number of environments where one configuration outperformed another with a statistical significance. To test the significance, we used Mann-Whitney's U test with $p < 0.05$.

win \ loss	uniform	max	UCB1	TTTS
uniform	-	9	3	5
max	5 (-4)	-	4	2
UCB1	4 (+1)	8 (+4)	-	6
TTTS	**10** (+5)	9 (+7)	2 (-4)	-

Table 1: Win/loss comparisons for different BAI algorithms in VAE-IW+ann. $X(\pm Y)$ indicates X wins and Y wins minus the losses. TTTS outperformed uniform and max in significantly more domains. UCB1 won against TTTS in more domains in direct comparison, but failed to win decisively against uniform (4 wins, 3 losses).

Ablation Study

We perform an ablation study of Olive to understand the effect of each improvement added to VAE-IW. We test the effects of VAE training parameters, the action selection strategy used in rollouts, and active learning of the dataset. Table 2 shows a summary of these comparisons.

VAE-IW + Temperature Annealing: Our VAEs are identical to those described in (Dittadi, Drachmann, and Bolander 2021), consisting of 7 convolutional layers in the encoder and the decoder. However, they elected not to anneal the temperature τ (Jang, Gu, and Poole 2017) when training the Binary-Concrete VAE, and instead kept the value low ($\tau = 0.5$). A BC-VAE becomes slower to train when τ is small because the activation function becomes closer to a step function, with small gradients away from the origin. We evaluated an exponential schedule $\tau = \tau_{\max}e^{-Ct}$ for an epoch t and an appropriate constant C which anneals $\tau = \tau_{\max} = 5.0$ to $\tau = 0.5$ at the end of the training ($t = 100$).

We compared the total number of significant wins of $\tau_{\max} = 5.0$ against $\tau_{\max} = 0.5$, thereby enabling/disabling annealing. $\tau_{\max} = 5.0$ won against $\tau_{\max} = 0.5$ in 17 domains, while it lost in 14 domains. From this result, we conclude that annealing causes a significant improvement in score. We call this variation VAE-IW+ann.

VAE-IW + ann + BAI Rollout: We next evaluated the effect of adding active learning in action selection to VAE-IW+ann using BAI. Table 1 shows the wins between BAI algorithms, including uniform (baseline), UCB1, TTTS, and max (a greedy baseline that selects the action with the maximum empirical mean reward). The results indicate that TTTS significantly outperforms uniform and max by winning in 10 and 9 games, and losing in 5 and 2 games, respectively. While UCB1 has 6 wins and 2 losses against TTTS, it did not outperform uniform decisively (4 wins and 3 losses). Since TTTS outperforms uniform more reliably, we conclude that VAE-IW+ann+TTTS is the best approach without online dataset updates.

Offline Learning vs. Passive Olive vs. Active Olive: Finally, we test the effect of online representation learning by comparing Passive and Active Olive to offline approaches in Table 2. Active Olive is more effective than Passive Olive (6-

to-4 against Passive), and wins more against offline learning (Active wins 7-to-5 while Passive wins 7-to-6, against the best offline approach, VAE-IW+ann+TTTS.)

The impact of Active Olive is most important for games whose visual features change significantly at higher scores. For example, in BankHeist, JamesBond, Pitfall, and WizardOfWor, the background and enemy designs change once an agent reaches far enough in the game, while in Amidar new enemy designs and color swaps occur. Montezuma's Revenge has multiple rooms, and the maximum score of Olive reached 2500 without policy learning, which outperforms 540 reported by 2BFS (Lipovetzky, Ramirez, and Geffner 2015). However, the impact of active online learning is game dependent. In games whose visual features are rather static, the VAE trained by VAE-IW can be sufficient.

We also see Passive Olive could outperform Active Olive in some games. This is due to the weakness of uncertainty sampling that, while it seeks for new screens, it does not consider the similarity between the k selected screens, i.e., selected screens can be visually similar to each other and can skew the dataset distribution. The more sophisticated Active Learning methods (e.g., mutual information maximization) address this issue, but we leave the extension for future work. Instead, to assess the best possible scores achievable with online learning, we analyzed a hypothetical, oracular portfolio, labeled as PortfolioOlive, which counts the domains in which either ActiveOlive or PassiveOlive wins against the baselines, and domains in which both lost against the baselines. As expected, this approach even more significantly outperformed existing baselines.

Main Results: Olive vs. π-IW vs. EfficientZero

Finally, Table 3 provides individual average scores of ActiveOlive across 55 Atari domains, alongside scores achieved by Rollout-IW and VAE-IW. In terms of average scores, ActiveOlive outperforms VAE-IW in **32-to-20**, and Rollout-IW in **42-to-11**. Olive is also **best in class in 30 domains**. Standard errors and maximum scores are included in the appendix. As a reference, we added human scores and VAE-IW scores from (Dittadi, Drachmann, and Bolander 2021) which uses hardware-specific 0.5 second planning budget and an unknown training budget.

Next, to understand the impact of a good representation relative to the impact of policy-learning, Table 3 also compares ActiveOlive against π-IW (Junyent, Jonsson, and Gómez 2019), DQN (Mnih et al. 2015) and EfficientZero (Ye et al. 2021). All results are obtained from the cited papers, except π-IW which is based on its Arxiv manuscript, as per its authors' request. π-IW is a width-based approach guided by a neural policy function which discretizes its intermediate layer as the feature vector for width-based search. This is an interesting comparison because π-IW has a significantly larger total training budget (2×10^7) compared to ActiveOlive (10^5), while operating under the same planning budget of 100. DQN is a state of the art model-free RL approach. EfficientZero is a recent state of the art in model-based data-efficient RL trained on a subset of Atari domains under the same 10^5 environment interactions. It uses a frameskip of 4 and 50 MCTS simulations per acting, instead of 15 and 100 in Olive, thus is allowed to see the screen $15/4 = 3.75$ times more frequently, and is given about a 1.8 times larger planning budget per second (equivalent to $50 \cdot \frac{15}{4} = 187.5$ simulations per acting using frameskip 15). **ActiveOlive outperforms π-IW by 30-to-22, DQN by 31-to-17, EfficientZero by 18-to-7.** VAE-IW also outperforms EfficientZero by 15-to-10. These wins/losses are counted by comparing the average results without the U-test due to the lack of the data on each run. Olive has average runtime per action of 0.38 sec., which is closely above real-time performance (15 frames, 0.25 sec.) that would be achievable by optimization and faster hardware.

Related Work

Incorporating Multi-Armed Bandit strategies to address exploration-exploitation trade off in search problems has a long history. Indeed, the initial evaluation of the planning-based setting of Atari was done based on UCB1 (Bellemare et al. 2013). However, from a theoretical standpoint, UCB1 is not appropriate in *simple regret* minimization scenario like online planning and acting. To our knowledge, we are the first to use a Bayesian BAI algorithm (TTTS) in MCTS. A limited body of work has used frequentist BAI in MCTS (e.g. Successive Halving in (Cazenave 2014)), but not within width-pruned search or the Atari domain.

π-IW (Junyent, Jonsson, and Gómez 2019) learns a Q-function which is then combined with Rollout-IW by replacing the uniform action selection with a softmax of Q values. However, it does not explicitly consider the uncertainty of the current estimate. In other words, the Q-function in π-IW provides a point estimate (mean) of $Q(\boldsymbol{x}, a)$, rather than its distributional estimate. Thus, while we focused on knowledge-free uninformed search as the basis of our method, our contribution of uncertainty-aware Bayesian methods is orthogonal to policy learning approaches. Extending π-IW to Bayesian estimates is future work.

Olive is limited to learning the state representation, requiring simulator interactions for rollouts. To further improve the sample efficiency, future work includes learning the environment dynamics, similar to model-based RL (Schrittwieser et al. 2020) but without expensive, sample-inefficient policy learning. Learning a PDDL/STRIPS model (Asai and Muise 2020) and leveraging off-the-shelf heuristic functions that provide a search guidance without learning is an interesting avenue of future work.

Conclusions

In this paper, we proposed Olive, an online extension of VAE-IW, which obtains a compact state representation of the screen images of Atari games with a variational autoencoder. Olive incrementally improves the quality of the learned representation by actively searching for states that are both new and rewarding, based on well-founded Bayesian statistical principles. Experiments showed that our agent is competitive against a state of the art width-based planning approach that was trained with more than 100 times larger training budget, demonstrating Olive's high sample efficiency.

win \ loss	RIW	VAE-IW	+ann	+ann+TTTS	**PassiveOlive**	**ActiveOlive**	**PortfolioOlive**
RIW	-	8	10	7	4	6	4
VAE-IW	31 (+23)	-	14	13	11	13	10
+ann	31 (+21)	17 (+3)	-	5	5	7	4
+ann+TTTS	38 (+31)	20 (+7)	10 (+5)	-	6	5	3
PassiveOlive	36 (+32)	21(+10)	6(+1)	7 (+1)	-	4	-
ActiveOlive	35 (+29)	20(+7)	7(±0)	7 (+2)	6 (+2)	-	-
PortfolioOlive	38 (+34)	26 (+16)	10(+6)	9(+6)	-	-	-

Table 2: Ablation study on Olive comparing wins/losses. In each $X(\pm Y)$, X indicates the number of wins, and Y indicates the number of wins minus losses. Adding annealing and a Bayesian BAI (TTTS) active learning significantly improves the performance. Active representation learning improved the performance over passive representation learning, but they are complementary. Their hypothetical, oracular portfolio (the best online configuration) shows future prospects.

train/plan budget	human	VAEIW NA/0.5s	RIW 0/100	VAEIW 10^5/100	Olive 10^5/100	Olive 10^5/100	π-IW 2×10^7/100	DQN 5×10^7/NA	EfficientZero 10^5/(187.5)	Olive sec./act.
Alien	6875	7744	6539	**7535**	5450	**5450**	3969.8	3069	1140.3	0.41
Amidar	1676	1380.3	537	928	**1390**	**1390**	950.4	739.5	101.9	0.4
Assault	1496	1291.9	1053	1374	**1477**	1477	1574.9	**3359**	1407.3	0.3
Asterix	8503	999500	919580	**999500**	**999500**	**999500**	346409.1	6012	16843.8	0.31
Asteroids	13157	12647	2820	33799	**41213**	**41213**	1368.5	1629	-	0.33
Atlantis	29028	1977520	62538	**2327026**	2266404	**2266404**	106212.6	85641	-	0.39
Bank Heist	734.4	289	241	219	**301**	301	**567.2**	429.7	361.9	0.34
Battle zone	37800	115400	42640	33760	**67660**	67660	**69659.4**	26300	17938	0.38
Beam rider	5775	3792	**4435**	2786	3785	3785	3313.1	**6846**	-	0.38
Berzerk	-	863	720	**847**	702	702	**1548.2**	-	-	0.33
Bowling	154.8	54.4	26	**64**	**64**	**64**	26.3	42.4	-	0.33
Boxing	4.3	89.9	**100**	99	98	98	**99.9**	71.8	44.1	0.44
Breakout	31.8	45.7	6	**58**	53	53	92.1	401.2	**406.5**	0.39
Centipede	11963	428451.5	103983	**623726**	129967	**129967**	126488.4	8309	-	0.36
Chopper command	9882	4190	13490	14842	**37248**	**37248**	11187.4	6687	1794	0.4
Crazy climber	35411	901930	95056	308970	**735826**	**735826**	161192	114103	80125.3	0.32
Demon attack	3401	285867.5	23882	128101	**128817**	**128817**	26881.1	9711	13298	0.37
Double dunk	-15.5	8.6	5	4	**8**	**8**	4.7	-18.1	-	0.39
ElevatorAction	-	40000	84688	**113890**	83826	**83826**	-	-	-	0.36
Enduro	309.6	55.5	1	**65**	25	25	**506.6**	301.8	-	0.41
Fishing derby	5.5	-20	-36	**-30**	-33	-33	**8.9**	-0.8	-	0.46
Freeway	29.6	5.3	7	**8**	7	7	0.3	**30.3**	21.8	0.49
Frostbite	4335	259	416	270	**509**	**509**	270	328.3	313.8	0.39
Gopher	2321	8484	4740	6925	**10073**	10073	**18025.9**	8520	3518.5	0.34
Gravitar	2672	1940	535	1358	**1655**	1655	**1876.8**	306.7	-	0.32
Ice hockey	0.9	37.2	30	34	**37**	**37**	-11.7	-1.6	-	0.39
James bond 007	406.7	3035	338	2206	**7637**	**7637**	43.2	576.7	459.4	0.36
Kangaroo	3035	1360	796	884	**1192**	1192	1847.5	**6740**	962	0.39
Krull	2395	3433.9	3428	**4086**	3818	3818	**8343.3**	3805	6047	0.41
Kung-fu master	22736	4550	**6488**	5764	4874	4874	**41609**	23270	31112.5	0.39
Montezuma	4367	0	0	0	**58**	**58**	0	0	-	0.38
Ms. Pac-man	15693	17929.8	15951	**19614**	16890	**16890**	14726.3	2311	1387	0.37
Name this game	4076	17374	14109	14711	**15804**	**15804**	12734.8	7257	-	0.36
Phoenix	-	5919	5770	**6592**	6165	**6165**	5905.1	-	-	0.32
Pitfall!	-	-5.6	-40	-50	**-12**	**-12**	-214.8	-	-	0.37
Pong	9.3	4.2	-6	**-5**	**-5**	-5	-20.4	18.9	**20.6**	0.36
Private eye	69571	80	57	**1004**	400	400	452.4	**1788**	100	0.4
Q*bert	13455	3392.5	1543	**7165**	6759	6759	**32529.6**	10596	15458.1	0.41
Riverraid	13513	6701	**7063**	6538	6303	6303	-	**8316**	-	0.39
Road Runner	7845	2980	13268	21720	**36636**	36636	**38764.8**	18257	18512.5	0.4
Robotank	11.9	25.6	**64**	35	61	**61**	15.7	51.6	-	0.45
Seaquest	20182	842	913	1785	**2469**	2469	**5916.1**	5286	1020.5	0.36
Skiing	1652	-10046.9	-29950	-29474	**-28143**	-28143	**-19188.3**	-	-	0.38
Solaris	-	7838	**7808**	5155	7378	**7378**	3048.8	-	-	0.45
Space invaders	1652	2574	2761	2682	**3301**	**3301**	2694.1	1976	-	0.34
Stargunner	10250	1030	2232	3164	**5234**	5234	1381.2	**57997**	-	0.3
Tennis	-8.9	4.1	-14	**8**	6	**6**	-23.7	-2.5	-	0.4
Time pilot	5925	32840	13824	**27946**	25180	**25180**	16099.9	5947	-	0.34
Tutankham	167.7	177	**158**	147	149	149	**216.7**	186.7	-	0.35
Up'n down	9082	762453	**834351**	729061	741694	**741694**	107757.5	8456	16095.7	0.54
Venture	1188	0	0	**6**	0	0	0	**380**	-	0.35
Video pinball	17298	373914.3	319596	437727	**464096**	464096	**514012.5**	42684	-	0.45
Wizard of wor	4757	199900	142582	175066	**197662**	**197662**	76533.2	3393	-	0.37
Yars' revenge	-	96053.3	69344	87940	**90778**	90778	**102183.7**	-	-	0.34
Zaxxon	9173	15560	6496	10472	**10594**	10594	**22905.7**	4977	-	0.36

Table 3: Comparison of the average scores. Best scores in each group in **bold**. We also included human and VAE-IW scores from (Dittadi, Drachmann, and Bolander 2021) as a reference. Scores of existing work are from the cited papers except π-IW (based on its Arxiv manuscript, as per authors' request). Hyphens indicate missing data. EfficientZero's planning budget is adjusted for frameskip 15. The rightmost column shows the average runtime of Olive between actions in seconds.

Acknowledgments

This work is also supported by the MIT-IBM Watson AI Lab, and its member company, Woodside.

References

Asai, M.; and Fukunaga, A. 2018. Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary. In *AAAI*, 6094–6101. AAAI Press.

Asai, M.; and Muise, C. 2020. Learning Neural-Symbolic Descriptive Planning Models via Cube-Space Priors: The Voyage Home (to STRIPS). In *IJCAI*.

Audibert, J.-Y.; Bubeck, S.; and Munos, R. 2010. Best Arm Identification in Multi-Armed Bandits. In *Proc. of the Conference on Learning Theory*, 41–53. Citeseer.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2-3): 235–256.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS+ Planning. *Computational Intelligence*, 11(4): 625–655.

Badia, A. P.; Piot, B.; Kapturowski, S.; Sprechmann, P.; Vitvitskyi, A.; Guo, Z. D.; and Blundell, C. 2020. Agent57: Outperforming the Atari Human Benchmark. In *ICML*, 507–517. PMLR.

Bandres, W.; Bonet, B.; and Geffner, H. 2018. Planning with Pixels in (Almost) Real Time. In *AAAI*, volume 32.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *J. Artif. Intell. Res.(JAIR)*, 47: 253–279.

Cazenave, T. 2014. Sequential Halving Applied to Trees. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(1): 102–105.

Dittadi, A.; Drachmann, F. K.; and Bolander, T. 2021. Planning from Pixels in Atari with Learned Symbolic Representations. In *AAAI*, volume 35, 4941–4949.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, 3(1-3): 251–288.

Frances, G.; Ramírez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely Declarative Action Representations are Overrated: Classical Planning with Simulators. In *IJCAI*, 4294–4301.

Gelman, A.; Carlin, J. B.; Stern, H. S.; and Rubin, D. B. 1995. *Bayesian Data Analysis*. Chapman and Hall/CRC.

Gittins, J.; Glazebrook, K.; and Weber, R. 2011. *Multi-Armed Bandit Allocation Indices*. John Wiley & Sons.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *J. Artif. Intell. Res.(JAIR)*, 14: 253–302.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. In *ICLR*.

Jinnai, Y.; and Fukunaga, A. 2017. Learning to Prune Dominated Action Sequences in Online Black-Box Planning. In *AAAI*, volume 31.

Junyent, M.; Gómez, V.; and Jonsson, A. 2021. Hierarchical Width-Based Planning and Learning. In *ICAPS*, volume 31, 519–527.

Junyent, M.; Jonsson, A.; and Gómez, V. 2019. Deep Policies for Width-Based Planning in Pixel Domains. In *ICAPS*, volume 29, 646–654.

Karnin, Z.; Koren, T.; and Somekh, O. 2013. Almost Optimal Exploration in Multi-Armed Bandits. In *ICML*, 1238–1246. PMLR.

Kaufmann, E.; and Koolen, W. 2017. Monte-Carlo Tree Search by Best Arm Identification. In *Neurips*, volume 30.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *ECML*, 282–293. Springer.

Liang, Y.; Machado, M. C.; Talvitie, E.; and Bowling, M. H. 2016. State of the Art Control of Atari Games Using Shallow Reinforcement Learning. In *AAMAS*, 485–493. ACM.

Lipovetzky, N.; and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In *ECAI*, volume 2012, 20th.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning . In *AAAI*.

Lipovetzky, N.; Ramirez, M.; and Geffner, H. 2015. Classical Planning with Simulators: Results on the Atari Video Games. In *IJCAI*.

López, C. L.; Celorrio, S. J.; and Olaya, Á. G. 2015. The Deterministic Part of the Seventh International Planning Competition. *Artificial Intelligence*, 223: 82–119.

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2017. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *ICLR*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-Level Control through Deep Reinforcement Learning. *Nature*, 518(7540): 529–533.

Pepels, T.; Cazenave, T.; Winands, M. H.; and Lanctot, M. 2014. Minimizing Simple and Cumulative Regret in Monte-Carlo Tree Search. In *Workshop on Computer Games at the European Conference on Artifical Intelligence*, 1–15. Springer.

Russo, D. 2020. Simple Bayesian Algorithms for Best-Arm Identification. *Operations Research*, 68(6): 1625–1647.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2020. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature*, 588(7839): 604–609.

Settles, B. 2012. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Shleyfman, A.; Tuisov, A.; and Domshlak, C. 2016. Blind Search for Atari-Like Online Planning Revisited. In *IJCAI*, 3251–3257.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

Teichteil-Königsbuch, F.; Ramírez, M.; and Lipovetzky, N. 2020. Boundary Extension Features for Width-Based Planning with Simulators on Continuous-State Domains. In *IJCAI*, 4183–4189.

Wallace, B. C.; Small, K.; Brodley, C. E.; and Trikalinos, T. A. 2011. Class Imbalance, Redux. In *ICDM*. IEEE.

Ye, W.; Liu, S.; Kurutach, T.; Abbeel, P.; and Gao, Y. 2021. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34.