Width-based Lookaheads with Learnt Base Policies and Heuristics Over the Atari-2600 Benchmark

Anonymous Author(s)
Affiliation
Address
email

Abstract

We propose new width-based planning and learning algorithms applied over the Atari-2600 benchmark. The algorithms presented are inspired from a careful analysis of the design decisions made by previous width-based planners. We benchmark our new algorithms over the Atari-2600 games and show that our best performing algorithm, RIW$_C$+CPV, outperforms previously introduced width-based planning and learning algorithms π-IW(1), π-IW(1)+ and π-HIW(n, 1). Furthermore, we present a taxonomy of the set of Atari-2600 games according to some of their defining characteristics. This analysis of the games provides further insight into the behaviour and performance of the width-based algorithms introduced. Namely, for games with large branching factors, and games with sparse meaningful rewards, RIW$_C$+CPV outperforms π-IW, π-IW(1)+ and π-HIW(n, 1).

1 Introduction

The Atari-2600 games provide useful environments for benchmarking autonomous agents due to the diversity of behaviour required across the different games. The Atari-2600 games can be accessed through the Arcade Learning Environment (ALE) [1] which provides a typical Reinforcement Learning (RL) environment interface where given a state, the agent selects an action and receives a resulting state and reward. The two main approaches that have been used by autonomous agents applied to the Atari-2600 games have been RL methods [2, 3, 4] and Planning methods [5, 6]. The RL approaches have had great success surpassing the performance of human players for many of the Atari-2600 games. However, the RL approaches require long training times in order to train the Neural Networks (NN) used for policy and value functions. Planning agents do not require training time and instead use a bounded, fixed computational budget to decide which action to take at each time step of the game. The budget allowed for planning for each action is set as part of the experimental setting and can be set in such a way that the agent can play a game in real-time. Through the ALE interface, the agent is not provided a description of the transition or reward functions as is the case of models described through languages such as the Planning Domain Description Language (PDDL) [7]. Planning agents applied to the Atari-2600 games are required to work with a simulator, treating the environment's transition and reward functions as a black-box [5].

Width-based planning agents have been shown to be particularly successful on the Atari-2600 games when compared to other planning agents [5, 6]. Width-based planners prioritise search effort on states deemed to be novel. The novelty of a state can be defined in a number of ways. Previously, novelty tests have been extracted from the RAM of the game [5], handcrafted features extracted from the pixels of the screen [6] and learnt features extracted from the screen pixels through a NN [8, 9, 10]. In this paper we consider planners which use novelty measures directly over the values of screen pixels.

Recent approaches have combined the RL and planning methods into single agents that are applied to the Atari-2600 games [8, 11, 10]. Junyent et al. [8] combined a width-based planner with a learnt

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

policy defined over a NN in order to guide the planner to promising areas of the search space. The learnt NN was also used to extract features from which the novelty of states were defined over. In this paper we introduce new width-based planning and learning methods that learn both policy and value networks using a methodical learning schedule.

Through analysing previous width-based methods we construct and benchmark new width-based approaches for the Atari-2600 games. We also classify the Atari-2600 games according to their particular characteristics. These different groupings of the games helps us to gain insight into the performance of the algorithms we propose and benchmark. The paper contributions are: (1) an analysis of the previous width-based planning methods that have been applied to the Atari-2600 games, (2) introducing new width-based planning and learning approaches for playing the Atari-2600 games, (3) defining a methodical learning schedule and (4) identifying characteristics of the Atari-2600 games that influence the performance of different width-based planning approaches.

2 Background

2.1 MDPs

We model the Atari games as Markov Decision Processes (MDPs). We formalise MDPs, as described by Geffner and Bonet [12], as the tuple of $M = (\mathcal{S}, s_0, A, T, R)$, where $\mathcal{S} \subseteq \mathbb{R}^d$, $s_0 \in \mathcal{S}$ is the initial state, A provides the sets of applicable actions such that $A(s)$ is a set of actions applicable in $s \in \mathcal{S}$, T is a set of distributions such that $T(s, a, s')$ gives the probability of the transition from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$ given action $a \in A(s)$, and R is the reward function such that $R(a, s)$ returns the reward for performing action $a \in A(s)$ from state $s \in \mathcal{S}$. In this work we will be considering a special case, finite-horizon MDPs, where accumulated rewards need to be maximised over a given number of stages equal to the horizon H, starting at a fixed initial state, s_0. Terminal states in finite-horizon MDPs are absorbing states. That is, if s is a terminal state and we are at time step k, every action a will map (s, k) into $(s, k + 1)$ and will be reward-free i.e. $R(a, s) = 0$. Without loss of generality, we consider only *discrete* action sets and deterministic action transitions, that is $T(s, a, s')$ can only equal 1 for one state s' and 0 for any other state s'', $s, s', s'' \in \mathcal{S}$, $s' \neq s''$ and $a \in A(s)$.

Our goal is to produce a policy, π, that maps any given state into an action, such that it maximises the expected accumulated reward received for an episode of the MDP,

$$\text{argmax}_{\pi} E\left\{ \sum_{k=0}^{H-1} \gamma^k R(\pi(s_k), s_k) \right\} \tag{1}$$

where the expectation is over $s_{k+1} \sim T(s_k, \pi(s_k), s_{k+1})$, and γ is the discount factor. As we are dealing with finite horizon problems we set γ to 1.

We assume that we have access to a simulator of the environment that given any state-action pair (s, a), where $s \in \mathcal{S}$ and $a \in A(s)$, the simulator returns the reward $R(a, s)$, a resulting state s' following the probability distribution $T(s, a, s')$ and whether s' is a terminal state.

2.2 Online Planning over simulators

In this paper we explore online planning over simulators by considering width-based lookahead algorithms for the Atari-2600 games. Lookaheads use a simulator of the environment to consider rewards from different action trajectories from the current state into the future. An example of this is shown in Figure 1 where a lookahead tree is illustrated. We define the lookahead as,

Definition 1 (Lookahead Tree)**.** A lookahead tree is defined as L = (N, C, s_r) where N is a set of nodes/states, $s_r \in N$ is the root node and C is a function that given a node $s \in N$ and an action $a \in A(s)$ returns the children of s, that is $s_c \in C(s, a)$ and $s_c \in N$.

Through backing up the rewards for each node in the lookahead tree an expected value can be found for each action applicable at the current state. The operation of backing up the rewards and selecting which action to execute is defined as,

Definition 2 (Action selection of lookahead)**.** Given a lookahead L= (N, C, s_r) (Definition 1) the action to execute a is selected at the root s_r by $\text{argmax}_a\{Q(s_r, a)\}$, where $Q(s_r, a) = R(a, s_r) +$

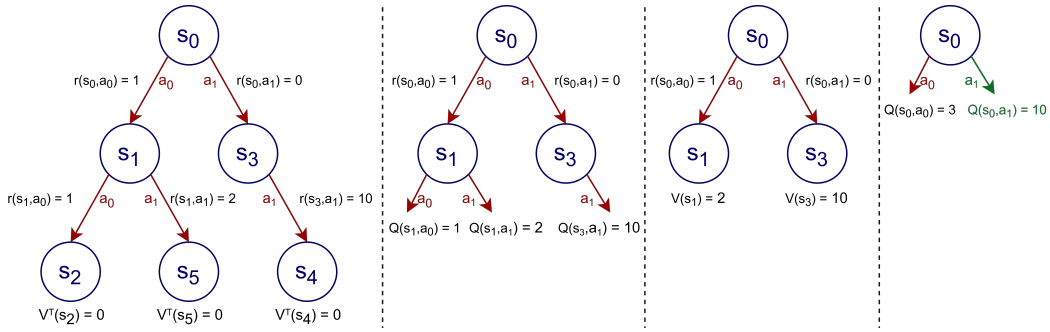

Figure 1: Example lookahead tree showing the action selection process for the root node s_0 following Definition 2. The transition with action a_1 from s_0 shown in green in the last diagram on the right is the transition added to the lookahead's critical path (Definition 3). The V, Q, and V^T functions are as defined in Definition 2.

$\sum_{s \in C(s_r,a)} T(s_r, a, s)V(s)$, where $V(s) = V^T(s)$, with $V^T(s)$ being a termination cost, when $C(s) = \emptyset$, otherwise $V(s) = \max_a \{R(a, s) + \sum_{s' \in C(s,a)} T(s, a, s')V(s')\}$.

Once an action is selected for execution, the lookahead is updated to have its root at the selected action's resulting node and the lookahead continues being constructed from the new root node. We define the action selected by the agent as a part of its *critical path*. That is,

Definition 3 (Critical Path)**.** Given the action selected a_t at each time step $t = 0, 1, \ldots, n$ following Definition 2, the critical path ρ is the sequence of state-action pairs $\rho = (s_0, a_0, s_1, a_1, \ldots, a_{n-1}, s_n)$, such that $s_{i+1} \sim T(s_i, a_i, \cdot)$ for $i = 0, \ldots, n-1$.

3 Related Work

3.1 MuZero

Schrittwieser et al. [11] followed up on the AlphaGo [13] and AlphaZero [14] algorithms with MuZero. AlphaZero is a planning and learning agent that achieved state-of-the-art performance on the games of Go, Chess and Shogi [15]. AlphaZero learns a policy and value network that are used within a Monte Carlo Tree Search (MCTS) lookahead through sampling actions according to the policy network and evaluating the states within the lookahead with the value function. Unlike AlphaZero, MuZero does not require a simulator or model of the game environment but instead learns a model of the environment through interaction. MuZero achieved state-of-the-art performance in the Atari-2600 games when compared to existing model-free RL algorithm performances.We also explore using learnt value and policy networks within a lookahead but consider width-based methods as opposed to MCTS. MuZero's experimental setting is different to the one considered in this paper as we require access to a simulator in the planning phase.

3.2 Width-based Planners on Atari

Here we will provide an overview of the different width-based planners which have been applied to the Atari-2600 problems. In the next Section we go into the design and implementation details that each of the following planners use.

Width-based planners [16] prioritise adding states to the lookahead with novel valuations of features that are defined over the states. *IW(1)* is a width-based *breadth-first search* that is guaranteed to run in *linear time and space* as it only expands *novel* states. *IW(1)* considers a state in the lookahead as *novel* if it is the *first* state within the lookahead to make a particular feature within the feature set true. The feature set used is an experimental setting. Width-based planners were first applied to the Atari-2600 games by Lipovetzky et al. [5], where they applied IW(1) over the RAM values of the game state. Lipovetzky et al. showed that IW(1) outperforms breadth-first search and UCT [17] planners.

Bandres et al.[6] introduced a *depth-first* version of the IW(1) planner, named Rollout-IW(1) (RIW). RIW(1) aims to contain the same nodes that are expanded by the IW(1) planner. As RIW(1) performs

Algorithm 1: Overview of the RIW(1) Algorithm

Input : A lookahead tree $L = (N, C, s_r)$
Output : Updated lookahead tree L and the termination cost function V^t

1 **while** $\neg$ *has_solved_label(s_r)* **do**
2 $\quad s \leftarrow s_r$
3 $\quad$ **while** *is_novel(s)* $\wedge \neg$ *is_terminal(s)* **do**
4 $\quad\quad s', a \leftarrow$ `sample_unsolved_child`(s, π_b)
5 $\quad\quad L \leftarrow$ `update_lookahead`(L, s, a, s'), $s \leftarrow s'$
6 $\quad$ **end**
7 $\quad$ `update_solved_labels`(s), $V^t(s) \leftarrow 0$ `// updating the termination cost of s`
8 **end**

depth-first search it was argued by Bandres et al. that it has better any-time performance than IW(1). The hypothesis for the better any-time performance of RIW(1) is that its search visits states that are further away from the initial state earlier in the search than its breadth-first search counter-part IW(1). Algorithm 1 provides an overview of RIW(1) using the base policy π_b. RIW(1) uses a random uniform base policy π_b. The function `sample_unsolved_child`, samples an action $a \sim \pi_b(s, \cdot)$ and a transition $s' \sim T(s, a, \cdot)$ provided that s' has not been marked as solved. If the selected transition does not already exist in the lookahead tree `update_lookahead` adds it. The `update_solved_labels` function adds a solved label to the given state and back-propagates the solved label to its parents if the parent's children have all been marked as solved. Bandres et al. results showed that RIW(1) outperformed IW(1) greatly when almost real-time budgets for planning were applied.

Junyent et al. [8] follow up on the use of a random base policy in Line 4 of Algorithm 1 with π-IW(1), an algorithm that replaces π_b with a trained policy defined over a NN. The intended effect is to orient the lookahead to promising areas of the state space. The NN from the trained policy was also used to extract features from the screen pixels for the computation of state novelty. Recently Junyent et al. [10] introduced π-IW(1)+ and π-HIW(n, 1) as follow ups of π-IW(1). π-IW(1)+ modifies π-IW(1) random breaking of ties for the action selection (Definition 2) to select the action with the branch of the lookahead tree that contains the most nodes. π-IW(1)+ also adds a learnt value function, $\tilde{V}$, which is used in the action selection (Definition 2) by modifying $V(s)$ to be $\max\{\tilde{V}(s), \bar{V}(s)\}$, where $\bar{V}(s)$ is $V(s)$ as described in Definition 2. π-HIW(n, 1) is a hierarchical algorithm that has a high-level planner which uses a coarse down-sampling of the screen pixels as a feature set and a low-level planner which uses π-IW(1)+ with the feature set defined through the policy network as previously described. The high-level planner uses a modified stochastic exploration policy, that selects actions with probability inversely proportional to state visitation counts.

4 Width-based Planning and Learning for the Atari Games

In this Section we step through different design considerations when constructing a width-based planning and learning algorithm. We compare the design decisions made by previous works and propose new algorithms to test over the Atari-2600 games.

4.1 Definition of Novelty

The novelty definition of a width-based lookahead dictates which states to prune. In Algorithm 1 the novelty definition determines the output of the `is_novel` function. We refer to IW(1)'s novelty definition as the "Classic" definition and define it as,

Definition 4 ("Classic" Novelty)**.** Given a feature set $F = \{f_1, \ldots, f_i, \ldots, f_k\}$ s.t. $f_i : S \rightarrow \{\top, \bot\}$, and a lookahead L= (N, C, s_r), a node n is novel, if n contains the first state generated s.t. $f(n) = \top$ for some $f \in F$, that is, $\forall n' \in N$, $f(n') = \bot$ and $n' \neq n$.

The "Depth" novelty definition introduced for RIW(1) by Bandres et al. [6] is,

Definition 5 ("Depth" Novelty)**.** Given a lookahead L= (N, C, s_r), we say that a newly generated node, $n \notin N$, reached after doing $d(n)$ actions from s_r, is novel, if $f(n) = \top$ for some $f \in F$, and $\forall n' \in N$, such that $d(n') \leq d(n)$, $f(n') = \bot$.

In what follows, we refer to RIW(1) where one replaces the Bandres et al. "Depth" novelty definition with the "Classic" one as RIW_C, and use RIW_D to refer to the Bandres et al. configuration of RIW(1).

4.2 Novelty Features

Width-based methods require a feature set F to be defined over the observable state-space S. There are two types of observations that can be used for the Atari-2600 games, the internal states of the Atari-2600 machine (the RAM), and the colours of screen pixels. Either of these enable features to be defined, as arbitrary Boolean functions over the observable variables. For the internal state observables we have b $\times$ x variables, where b is the size of the Atari memory word (8 bits) and x is the size of the physical RAM given by the number of distinct memory addresses (128 addresses). There are c $\times$ w $\times$ h screen observable variables, where c is 128, w is 160, and h is 210, corresponding to the colour depth, and the number of screen pixels along the horizontal and vertical directions.

When RIW(1) was introduced [6], Bandres et al. stated that features capturing "meaningful structure" would yield better results than using raw features. Hence, Bandres et al. mapped the observable screen variables into the feature set B-PROST, first proposed by Liang et al. [3]. The B-PROST feature set attempts to capture temporal and spatial relationships between the past and present screen pixel values. In order to compute the set of B-PROST features there are a number of steps required. First a set of basic features needs to be computed through dividing the screen into 16×14 tiles comprised of 10×15 pixels. For each tile, (w, h), where $w \in \{1, \ldots, 16\}$ and $h \in \{1, \ldots, 14\}$, there are K features where K is equal to the colour depth of the Atari-2600 pixels (128). The basic feature in the B-PROST set is $f_{w,h,c}$, where $c \in \{1, \ldots, K\}$ is true if the tile (w, h) contains at least one pixel with the colour value c. A second tier of features, the Basic Pairwise Relative Offsets in Space (B-PROS) set, is computed from the basic ones. A B-PROS feature $f_{c_1,c_2,i,j}$, is true if $f_{w,h,c_1} \wedge f_{w+i,h+j,c_2}$ for any w, h. Finally, a third tier of features, the Basic Pairwise Relative Offsets in Time (B-PROT) set, are computed. A B-PROT feature considers the current screen's tiles (w, h) and the previous game screen's tile (w', h') so that a feature $f^t_{c_1,c_2,i,j}$ is true if $f_{w,h,c_1} \wedge f_{w'+i,h'+j,c_2}$ for any w, h where $w' = w$ and $h' = h$. The B-PROST set is the union of basic, B-PROS, and B-PROT feature sets.

The feature set can also be defined dynamically through a NN [8]. π-IW, π-IW(1)+ and the lower level planner of π-HIW(n, 1) use a feature set that is defined as the output values of the rectified linear units from the last hidden layer of the policy NN treating zero values as $\perp$ and positive as $\top$. The policy NN input are the last four screens, processed to map colours to a suitably defined grayscale, and down sampled to a size of 84×84. While the policy network is being trained the features extracted through it will also change.

We define our feature set directly over the values of the current down sampled 84×84, 8-bit grayscaled, observable screen variables. Each feature is defined as $f_{i,j,c}$ and is true if the downsampled pixel (i, j) has the grayscaled colour c, where $i, j \in \{1, \ldots, 84\}$ and $c \in \{1, \ldots, 256\}$.

4.3 Base Policies and Value Functions

We propose a new algorithm based on Algorithm 1, RIW(1) with Critical Path Value Function approximation or RIW+CPV for short, that incorporates both a policy and value function network. We change the base policy in Algorithm 1, which we define as a policy network, similarly as π-IW does. Besides that, we use a cost-to-go approximation which we implement with a NN as a value function, which we evaluate at the non-terminal leaf nodes of the lookahead. Using cost-go-approximations has been shown to significantly improve the performance of width-based lookaheads over stochastic shortest paths [18], but rather than using simulations to obtain the cost-to-go estimates, we rely on a learnt heuristic function. That is, instead of assigning termination costs, V^T, of 0 as done in line 7 of Algorithm 1, if the state is not terminal the valuation of a learnt value function network is used. We note that unlike π-IW(1)+ we do not use the learnt value function to modify the $V(s)$ definition as defined in Definition 2.

4.4 Training data

The policy network of RIW+CPV uses the state action pairs, (s_j, a_j), of previous episodes performed by RIW+CPV with NN parameters, $<\theta^{\pi}_{i-1}, \theta^{V}_{i-1}>$, in order to train new NN parameters θ^{π}_{i}. This is similar to how π-IW(1) trains its policy function, except for the fact that π-IW(1) uses the Q values

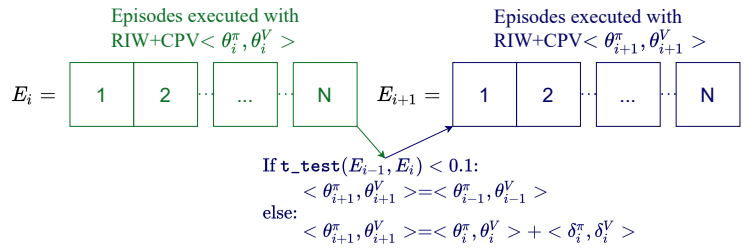

Figure 2: Illustration of the schedule for the network parameter updates, $<\theta^{\pi}, \theta^{V}>$, of RIW+CPV. The E arrays contain episodes executed by RIW+CPV and each episode in the array is represented by a square in the diagram. The `t_test` function returns the p-value for Welch's t-test [20] of the episode rewards executed with the old $i-1$ parameters being better than the new ones. δ^{π}_{i} and δ^{V}_{i} are real vectors of the same dimensions as θ^{π}_{i} and θ^{V}_{i} respectively. δ^{π}_{i} and δ^{V}_{i} are functions over E_i.

within the lookahead tree. If multiple actions in π-IW's lookahead have the same Q value for a given state, instead of the training vector assigning a probability of one to the executed action, π-IW uniformly distributes the probability across the actions with the same Q values. We have taken the simpler approach of just using 1-hot encodings for the single selected action along the critical path (Definition 3) of the RIW+CPV algorithm. Curating the training dataset in this way also means RIW+CPV does not need access to the internal data structures of the planning agent itself but instead can externally observe any agent interacting with the environment in order to acquire the training data.

Influential deep RL algorithms such as DQN [2] which have been applied to the Atari-2600 games rely on evaluating an ϵ-greedy policy defined over the parameters of its network in order to sample transitions and use Q-learning updates on the parameters of the network. We follow this strategy and perform Temporal Differential (TD) [19] learning to train the value network. The selection of the transitions that are used for the TD learning determines what the value function is estimating. That is, TD's task is to estimate the expected accumulated rewards from the given state following the policy which underlies the transitions that it is trained on. In RIW+CPV the transitions within the lookahead follow the base policy which is being learnt by aiming to mimic the actions selected by RIW+CPV in the previous training iteration. However, the end result of a RIW+CPV execution is not necessarily equivalent to the base policy execution, as actions are selected based on the information within the lookahead following Definition 2. Therefore it does not make sense to approximate the expected accumulated reward of the lookahead with the expected accumulated reward of the base policy. Instead for the training data we select only the transitions on the critical path of the lookahead (Definition 3), that is, the transitions selected by the RIW+CPV. This results in the value function approximating the expected accumulated reward of executing RIW+CPV from a given state.

4.5 Learning schedules

RIW+CPV has a set of network parameters for its policy network, θ^{π} and value function network, θ^{V}. As illustrated in Figure 2, RIW+CPV tests whether to accept or reject network parameter updates $<\theta^{\pi}_{i}, \theta^{V}_{i}>$ for the parameters $<\theta^{\pi}_{i-1}, \theta^{V}_{i-1}>$ according to their performance when used within RIW+CPV. RIW+CPV performs a Welch's t-test [20] on its performance with the new i parameters vs. the old $i-1$ parameters and rejects the update if the t-test suggests, with a p-value of less than 0.1, that performance deteriorates with the parameter update. This training and parameter update schedule allows learning steps to be completed at each time step like done by π-IW [8], except that the updated parameters are not used for data generation by RIW+CPV until they have been accepted by the proposed learning schedule. Our proposed learning schedule is similar to AlphaZero [14] which evaluates each new set of network parameters θ' that are trained against the current set of network parameters θ to ensure θ' improves AlphaZero's performance.

5 Experimental Study

We benchmark width-based planning and learning methods with variations of the design decisions explained in the previous section. Here we explain our experimental methodology and provide results across the different algorithms over the Atari-2600 games.

5.1 Methodology

Given vast compute resources it would be preferable to conduct a full ablation study over the Atari-2600 benchmark on each element of a width-based planning and learning algorithm discussed in the previous Section. However, due to computational constraints we instead select four width-based planners we believe will provide the most insight. Two of the planners we evaluate are based on RIW(1) without learnt policy or value function networks. One of the planners uses the "Depth" definition of novelty (Definition 5) and the other uses the "Classic" definition (Definition 4), we name these version RIW_D and RIW_C respectively. Note that RIW_D is as described in Bandres et al.[6], except that the features are defined directly over the screen's pixel values as discussed in the previous Section. The other 2 width-based planners we benchmark are two versions of RIW+CPV, as previously described. Again we test both the "Classic" and "Depth" novelty definitions (Definitions 4, 5), and refer to the planning and learning algorithms as RIW_C+CPV and RIW_D+CPV. For the policy networks of RIW_C+CPV and RIW_D+CPV we use the same architecture used by Mnih et al. [2]. The value network uses the same architecture as the policy network except that instead of the output layer being a dense softmax layer with an output for each action, the output layer of the value network is a dense linear layer with a single output value.

We compare RIW_D, RIW_C, RIW_C+CPV, and RIW_D+CPV to the π-IW, π-IW(1)+ and π-HIW(n, 1) planners with the results as given by Junyent et al. [10]. We do not directly compare our results to those given in the original RIW(1) planner as Bandres et al. [6] use a different experimental design. That is, Bandres et al. and previous works such as Lipovetzky et al. [5] benchmarked their width-based planners over the Atari-2600 games all using the full action set of 18 different actions per state. We found in the code provided for the π-IW(1) work that it was benchmarked against the games using the minimal action set for each game. Using the minimal action set results in many games having much smaller branching factors, for example, instead of Breakout having a branching factor of 18, it has a branching factor of just 4. Due to computational constraints we could not benchmark our algorithms over the games with both the full and minimal action sets. We decided to benchmark using the minimal action set as Junyent et al. do.

The results are also not directly compared with those from MuZero due to discrepancies in the evaluation protocols and computing resource requirements. For example, MuZero uses a smaller frameskip for the environment time steps and uses a longer allowed episode length of 108,000 frames compared to the 18,000 frame maximum episode length we imposed. While our experiments run on a single vCPU for each trial for both training and evaluation, MuZero required 40 third generation Google Cloud TPUs for each run, 8 for training and 32 for its self-play. Furthermore the results for MuZero on each domain were only made available for a 20 billion frame training budget. We do however provide comparisons of our results with the RL algorithms DQN [2] and Rainbow [4], along with all the necessary caveats, in the Supplementary Material.

For the evaluation of each algorithm on each game we run 5 independent trials. Once training has completed we evaluate each trial over 10 episodes. Following previous width-based planning papers [6, 8, 10] we use a frameskip of 15. We keep our experimental settings the same as Junyent et al. [10] including a training budget of 2×10^7 simulator calls and allowing 100 simulator calls at each planning time step, which allows almost real-time planning. Note that previous width-based algorithms have varied in how they apply planning budgets, Lipovetzky et al. enforce a budget of 3,000 simulator calls with a frameskip of 5, while Bandres et al. enforce time budgets of 0.5 and 32 seconds with a frameskip of 15. We ran 80 independent trials at once over 80 Intel Xeon 2.10GHz processors with 720GB of shared RAM, limiting each trial to run on a single vCPU. The average vCPU run-time per time step needed across both the planning and learning steps were 1.28 and 1.11 seconds for RIW_C+CPV, and RIW_D+CPV respectively, resulting in each trial taking just under 3 days to complete. For RIW_C and RIW_D, which do not require any learning steps or evaluation of NN, the average run-times per step were 0.55 and 0.54 seconds respectively. Note that given Atari operates at 60 frames per second and we use a frameskip of 15 a real-time planner would be required to execute with a run-time of 0.25 seconds per time step.

The transitions within the lookahead tree are cached inline with previous width-based planners [5, 6, 8, 10]. That is, when the search revisits any transition previously added to the lookahead the simulator does not need to be recalled and hence does not affect the simulator budget. Also following previous work, transitions that are cached from previous time steps are not considered by the Novelty Definitions 4, and 5, and hence will never be pruned.

Table 1: A pairwise comparison of the width-based planning algorithms over the full benchmark set made up of 53 Atari Games. Numbers represent the number of games an algorithm had a higher average evaluation score over the 5 learning trials than the algorithm it is being compared to.

	Compared to							
	RIW$_D$	RIW$_C$	RIW$_D$+CPV	RIW$_C$+CPV	π-IW(1)	π-IW(1)+	π-HIW(n,1)	**Total**
RIW$_D$		29	6	5	20	18	17	**95**
RIW$_C$	23		4	3	19	18	17	**84**
RIW$_D$+CPV	46	48		26	32	38	30	**220**
RIW$_C$+CPV	47	49	26		32	39	32	**225**
π-IW(1)	32	33	20	20		30	25	**160**
π-IW(1)+	35	35	15	14	23		23	**145**
π-HIW(n,1)	36	36	23	21	28	30		**174**

Table 2: Same as Table 1 but for games with a Branching Factor $\geq$ 10 (33 Games).

	Compared to							
	RIW$_D$	RIW$_C$	RIW$_D$+CPV	RIW$_C$+CPV	π-IW(1)	π-IW(1)+	π-HIW(n,1)	**Total**
RIW$_D$		16	3	3	16	16	13	**67**
RIW$_C$	16		1	2	15	16	13	**63**
RIW$_D$+CPV	29	31		18	23	27	21	**149**
RIW$_C$+CPV	29	30	14		23	28	22	**146**
π-IW(1)	16	17	9	9		19	14	**84**
π-IW(1)+	17	17	6	5	14		11	**70**
π-HIW(n,1)	20	20	12	11	19	22		**104**

5.2 Results

Table 1 summarises the results of RIW$_D$, RIW$_C$, RIW$_D$+CPV, RIW$_C$+CPV, π-IW(1), π-IW(1)+ and π-HIW(n, 1). Using the pairwise comparison of the different algorithms across the 53 games it is clear that RIW$_D$+CPV and RIW$_C$+CPV are the most performant. Comparing the "Depth" vs. "Classic" novelty definition methods as RIW$_D$ vs. RIW$_C$, the former performs better than the latter. The superiority of the "Depth" over the "Classic" definition of novelty does not follow when using our CPV method. The "Classic" method of RIW$_C$+CPV slightly outperforms the "Depth" method RIW$_D$+CPV, with Table 1 indeed showing that RIW$_C$+CPV is the best performing algorithm overall.

To better understand the performance of the algorithms we segment the benchmark set according to a couple of different characteristics. The game characteristics we examine are the *branching factor*, and the *sparseness* of meaningful reward feedback (SMRF). We consider rewards as meaningful when they provide information to a player about how to maximise the accumulated reward of an episode. For a given game, SMRF is determined by executing a random policy and a Real-Time Dynamic Planner (RTDP) [21] over each of the games. RTDP is an online planner that uses a one step lookahead tree (Definition 1) and an approximation for the termination cost V^t at each of the leaf nodes. For the approximation of $V^t(s')$ we use the accumulated reward from a random policy executed from s' for 10 time steps. We run both the random policy and RTDP for 50 time steps (750 frames). If the results from the RTDP planner are not better than the random policy according to Welch's t-test [20] with $p < 0.1$, the game is classified as having SMRFs. The results of the random policy vs. RTDP can be found in the Supplementary Material. For example, in the game of Pong, RTDP will be able to discover states through its 10 step approximation of $V^t(s')$ where either player has scored. Using information from $V^t(s')$ about which players have scored, RTDP will be able to have a better informed policy than the random policy, so Pong would not be considered a SMRF game. In a game like Skiing, where a skier is required to ski down a mountain and pass through gates on its path down, RTDP will not discover any meaningful rewards in its 10 step rollouts. This is because in Skiing, despite a constant negative reward at each time step, there is no meaningful reward feedback until the skier reaches the bottom of the mountain where a negative reward is applied for each gate the skier missed.

Table 2 groups the games according to their branching factor. Comparing Table 2 and 1 we can see that for games with larger branching factors, the relative performance gap between our RIW$_D$+CPV and RIW$_C$+CPV planners and Junyent et al.'s π-IW(1), π-IW(1)+ and π-HIW(n, 1) increases as the branching factor increases. For example RIW$_D$+CPV and RIW$_C$+CPV perform better than π-IW(1)+ in 27/33 (81.8%) games and 28/33 (84.8%) games respectively for the games with a branching factor greater or equal to 10. However, for games with a branching factor less than 10, RIW$_D$+CPV and RIW$_C$+CPV only perform better than π-IW(1)+ in 11/20 (55%) and 11/21 (55%) games respectively.

Table 3: Same as Table 1 but for SMRF games (12 Games).

	Compared to							
	RIW$_D$	RIW$_C$	RIW$_D$+CPV	RIW$_C$+CPV	π-IW(1)	π-IW(1)+	π-HIW(n,1)	**Total**
RIW$_D$		5	2	1	6	3	4	**21**
RIW$_C$	6		2	1	6	3	4	**22**
RIW$_D$+CPV	9	9		4	7	9	6	**44**
RIW$_C$+CPV	10	10	7		7	9	7	**50**
π-IW(1)	5	5	4	4		4	3	**25**
π-IW(1)+	9	9	3	3	8		5	**37**
π-HIW(n,1)	8	8	6	5	9	7		**43**

Table 3 compares the pairwise performance for games classified as SMRF games. Table 3 clearly shows that "Classic" novelty (Definition 4) methods outperform "Depth" novelty (Definition 5). This observation, that contradicts previous claims in the literature, required careful analysis. We observed that the "Classic" method prunes states more aggressively, meaning it is more likely to reach states that are further away from the root node compared with the "Depth" definition. By definition, high rewards for SMRF games have a higher probability of being further away than games with dense rewards. Therefore, the "Classic" method is more likely to discover them by searching deeper in the lookahead tree. Interestingly RIW$_C$+CPV still outperforms π-IW(1)+ and π-HIW(n, 1) on the SMRF games, despite π-IW(1)+ and π-HIW(n, 1) being motivated by such domains.

6 Discussion

We have found significant discrepancies in the experimental settings used in the previous width-based planning papers for evaluating their algorithms over the Atari-2600 games. We believe a clear and consistent evaluation protocol should be set out for planning based algorithms applied to the Atari-2600 games to facilitate the direct comparison of their results. This could be similar to the evaluation protocol for the Atari-2600 games set out by Machado et al. [22], which was mainly focused towards RL agents and included recommendations on episode termination, setting of hyper-parameters, measuring training data, summarising learning performance and injecting stochasticity. However Machado et al. do not discuss evaluation settings that are vital to the deterministic planning setting we have explored in this paper, such as planning budgets, and caching of transitions within lookahead trees. We hope that by having addressed some of the discrepancies in the experimental settings of previous width-based algorithms, such as the size of the action set and the planning budget used, future research in planning agents for the Atari-2600 games can be more easily assessed. We were able to observe interesting patterns in the relative performance of algorithms through segmenting the Atari-2600 games by their different game characteristics. We are not aware of other works that analyse the performance of agents in regards to the characteristics of specific Atari-2600 games. We believe this taxonomy will provide useful insights into the behaviour of agents on the Atari-2600 games.

In this paper we have focused on width-based planning methods that have been applied over the Atari-2600 games. It is important to note though that these algorithms are defined in a general way to operate over MDPs. We proposed new width-based planning and learning algorithms through the examination of different design decisions made by previous implementations of width-based planners. These new algorithms, particularly RIW$_C$+CPV, are simpler implementations than the previously introduced width-based planning and learning algorithms π-IW(1)+ and π-HIW(n, 1). RIW$_C$+CPV defines its features directly over the grayscaled pixel colours of the game screen and uses a simplified novelty definition. Furthermore, RIW$_C$+CPV learns a value function which is only used for cost-to-go approximations at the leafs of the lookahead search tree. RIW$_C$+CPV also uses a methodical learning schedule we introduced for training both its policy and value networks. We found RIW$_C$+CPV to outperform π-IW(1), π-IW(1)+ and π-HIW(n, 1) not only across the Atari-2600 benchmark, but also over subsets of games with large branching factors and games with sparse meaningful rewards. These results show that RIW$_C$+CPV's integration of planning and learning pays-off for almost real-time planning over hard problems.

References

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518:529, 2015.

[3] Yitao Liang, Marlos C. Machado, Erik Talvitie, and Michael H. Bowling. State of the art control of atari games using shallow reinforcement learning. In Catholijn M. Jonker, Stacy Marsella, John Thangarajah, and Karl Tuyls, editors, *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, pages 485–493. ACM, 2016.

[4] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[5] Nir Lipovetzky, Miquel Ramirez, and Hector Geffner. Classical planning with simulators: results on the atari video games. In *Proc. of Int'l Joint Conf. in Artificial Intelligence (IJCAI)*, pages 1610–1616, 2015.

[6] Wilmer Bandres, Blai Bonet, and Hector Geffner. Planning with pixels in (almost) real time. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2018.

[7] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

[8] Miquel Junyent, Anders Jonsson, and Vicenc Gomez. Deep policies for width–based planning. In *Proc. of the Int'l Conf. in Automated Planning and Scheduling (ICAPS)*, 2019.

[9] Andrea Dittadi, Frederik K Drachmann, and Thomas Bolander. Planning from pixels in atari with learned symbolic representations. In *ICAPS Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning.*, 2020.

[10] Miquel Junyent, Vicenç Gómez, and Anders Jonsson. Hierarchical width-based planning and learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 519–527, 2021.

[11] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[12] Hector Geffner and Blai Bonet. A concise introduction to models and methods for automated planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(1):1–141, 2013.

[13] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and others. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[14] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, and others. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354, 2017.

[15] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[16] Nir Lipovetzky and Héctor Geffner. Width and serialization of classical planning problems. In *Proc. of European Conference in Artificial Intelligence (ECAI)*, pages 540–545, 2012.

[17] Levente Kocsis and Csaba Szepevari. Bandit based monte carlo planning. In *Proc. of European Conference in Machine Learning (ECML)*, pages 282–293, 2006.

[18] Stefan O'Toole, Miquel Ramirez, Nir Lipovetzky, and Adrian Pearce. Width-based lookaheads augmented with base policies for stochastic shortest paths. In *ICAPS Workshop on Heuristics and Domain Independent Planning (HSDIP)*, 2019.

[19] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[20] B. L. Welch. The generalization of 'student's' problem when several different population variances are involved. *Biometrika*, 34(1/2):28–35, 1947.

[21] Andy G. Barto, S. J. Bradtke, and S. P. Singh. Real–time learning and control using asynchronous dynamic programming. *Artificial Intelligence Journal*, 72:81–138, 1995.

[22] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] discussed throughout, for example we discuss our requirement of simulator when comparing to MuZero in related work, and the performance on small branching factor games being similar to baselines in the results Section.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] there are no obvious negative societal impacts of the work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] included in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] included in the supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] included 90% confidence intervals in Table 2 of the supplementary material.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] in Section 5.1.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Arcade Learning Environment is cited in the introduction. All result data used from previous works is also cited.
   (b) Did you mention the license of the assets? [Yes] In supplementary material.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Code for algorithms introduced is in supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]