Learning to Prune Dominated Action Sequences in Online Black-box Planning *

Yuu Jinnai Alex Fukunaga

Graduate School of Arts and Sciences
The University of Tokyo

Abstract: Black-box domains where the successor states generated by applying an action are generated by a completely opaque simulator pose a challenge for domain-independent planning. The main computational bottleneck in search-based planning for such domains is the number of calls to the black-box simulation. We propose a method for significantly reducing the number of calls to the simulator by the search algorithm by detecting and pruning sequences of actions which are dominated by others. We apply our pruning method to Iterated Width and breadth-first search in domain-independent black-box planning for Arcade Learning Environment, adding our pruning method significantly improves upon the baseline algorithms.

1 Introduction

Planning is the task of deciding a sequence of actions for an agent such that the agent achieves some set of goals, where the goals can include the optimization of some objective function. Much of the previous work in AI planning assumes that a model of the world and its dynamics is fully specified and available to the planner. For example, in classical planning with the standard STRIPS or SAS+ models, each action a available to the agent is specified precisely in terms of its preconditions (constraints on the state vector required to execute a) and effects (changes in the state vector as a result of executing a). Such *transparent* domain models have enabled the development of effective search algorithms which exploit the structure exposed by the model, e.g., heuristic functions that use the model to estimate the cost of achieving the goals from a given state.

Recently, planning in *black-box* domains with much more *opaque* domain models has attracted attention, spurred by interest in developing a game-independent AI playing algorithm for video games [1]. In black-box planning, a state vector and a set of actions are available, as well as an objective function for evaluating states. However, the only way to compute the successor state s' resulting from applying an action a to state s is to execute $s' = Simulate(s, a)$, a black-box simulation function for which the internal dynamics are inaccessible.

Such black-box domains present a challenge for search-based planning, because the pruning techniques which enabled effective search in domains with transparent domain models are not applicable, leaving us only with brute-force methods such as breadth-first search. However, it was recently shown that Iterative Width (IW) [2], a search strategy which prunes the search space by focusing only on states which are "novel" compared to previously expanded nodes, can be used as the basis for a successful, search-based planner for black-box planning [3].

[1]This is a preliminary version of a paper which will appear in AAAI-17.

In most traditional planning/search domains, successor state generation is relatively cheap, while state evaluation (e.g., calling a heuristic function) tends to be expensive. For example, in classical planning, powerful but expensive heuristic functions are often the bottleneck. In contrast, in black-box domains, *successor state generation is expensive*, since the successor computation ($Simulate$) can be a complex simulation algorithm. State evaluation can also be slow in black-box domains (it might require running another simulation), or evaluation might be cheap (e.g., in the Arcade Learning Environment (ALE) [1], evaluation is cheap because the score is computed as a side-effect of state generation).

Slow state generation introduces a significant bottleneck for black-box planning when the state space is a graph. When searching in standard domains using algorithms such as A* or breadth-first search, duplicate detection is relatively cheap (e.g., a hashing operation) and since state generation is fast, the presence of detectable duplicate states is usually not a critical issue. In contrast, in black-box domains with slow state generation, the duplicate check itself is fast (hashing), but by the time a duplicate check is performed, it is too late – a large cost has been incurred when the duplicate state has been generated. To avoid this overhead, *we must avoid generating duplicate states*. Previously proposed duplicate avoidance mechanisms (c.f. [4]), are inapplicable because they depend on the ability to identify duplicate action sequences by analyzing the underlying transparent domain models.

In this paper, we investigate duplicate avoidance in *online planning* settings for black-box domains, as exemplified by the on-line planning for the ALE [1], where an agent plays video games by repeating the loop: (1) solving a planning problem with a very limited resource budget, and (2) execute an action. Because this setting poses a series of related planning episodes, there is an opportunity to improve planner performance over time by learning a duplicate avoidance strategy.

Specifically, we seek to eliminate actions (and sequences of actions) which are dominated by others (and lead to duplicate states). For example, in the ALE,

which simulates an retro arcade game machine, 18 actions are always available (the joystick has 9 states – up/down/left/right/4 diagonals/"neutral", and the "fire" button has 2 states, 9x2=18). Previous work in search-based planning for the ALE treats all 18 actions as applicable at every state [3, 5]. However, in any particular game, many of these 18 actions are dominated ("useless"): First, some actions are trivially dominated because they are completely ignored, or the program always treats them as being equivalent to other actions (e.g., in some games, the state of the "fire" button is irrelevant). Second, some actions are *conditionally* dominated because, in a given context, the action results in the same state as another action (e.g., in a maze-based game, if the agent is stuck against a wall to the left, then the "left" action is useless because (in some games) it results in the same state as "no action"). More generally, *sequences of actions* can be useless. For example, some actions can have *cooldown periods*, i.e, after action a is used, executing a again has no effect for the next t seconds (e.g. firing missiles in shooting games).

Given the exogenous events are deterministic, dominated action sequences lead to duplicate states, so this paper focuses on avoiding dominated action sequences. If the domain is transparent, dominated actions can be detected trivially by analyzing domain models, and dominated action sequences can be pruned using methods such as duplicate action sequence detection [4], symmetry detection [6, 7], and strong stubborn sets [8]. However, in black-box planning, pruning dominated actions and action sequences is nontrivial because we can not be certain whether an action is truly dominated, or merely appears to be dominated due to the context provided by the current game state.

We propose Dominated Action Sequence Detection (DASD), an approach to detecting actions which are likely to be dominated by other action(s) in the course of online planning. DASD can be applied to online planning using any standard search algorithm (e.g., breadth-first search, IW) in order to prune dominated action sequences. We first propose Dominated Action Sequence Pruning (DASP), which assumes a static environment and classifies actions as being either dominated or non-dominated. Next, we propose Dominated Action Sequence Avoidance (DASA), a method to detect context-dependent (conditional) dominated action sequences, which learns the ratio of duplicate nodes generated by actions. DASA assigns a low probability of expanding action sequences that result in more duplicates (i.e., dominated actions), in order to avoid wasting search resources on unfruitful actions, and instead invests more resources into more promising (non-dominated) actions. We evaluate DASA and DASP applied to p-IW(1), IW(1) and breadth first search (BrFS) on 53 games in the ALE, and show that on all three search methods, DASA improved the performance compared to the baseline search method as well as the baseline method using a hand-coded (human-generated), game-specific, restricted action set.

2 Background

A black box planning problem is a tuple $B = (V, A, Sim, I, U)$. V is a set of variables, each with a discrete domain $D(v_i)$. A state is a combination of v_i for each V. A is the set of *available actions*. Any action in A can be applied at any state. The effects of $a \in A$ are computed using the black box simulation function Sim. Sim is a function which takes two parameters. $Sim(a, s)$ returns the state resulting from applying a to state s. I is an initial assignment of values to V. U is a utility function. The objective is to find a state s which maximizes $U(s)$ (satisficing, black-box planning uses a U which is maximal when desired (goal) attributes are satisfied, zero otherwise). In this paper, we assume that the environment is deterministic, i.e., Sim is deterministic.

The lack of useful knowledge makes search difficult. Brute-force, exhaustive search algorithms such as breadth-first search can be applied, but does not scale well [1]. More focused search techniques which do not depend on heuristics are needed. Iterative Width [3] is breadth-first search with novelty-based pruning: a newly generated state is pruned if it does not make a new atom true. IW(1) has been shown to perform well in classical planning [2], games in the ALE environment [3], and General video game playing [9]. p-IW [5] further improves the pruning by considering the reward in addition to novelty.

Online, black-box planning is a real-time search problem [10], where we are given an initial black-box planning instance B_0, and a resource limit (e.g., time limit, limit on number of node generations, etc.). An agent for online black-box planning behaves as follows: [Step 1, initialization]: I is initialized to I_0. [Step 2, termination check]: If some termination condition has been met, then terminate. [Step 3, planning episode]: The agent applies a planning algorithm P until the resource limit is exhausted, at which point the agent selects an action a to execute. [Step 4, world update]: The agent executes a, resulting in an updated world state $s' = Apply(a, s)$. In black-box domains where the simulator Sim is a perfect model of the actual world inhabited by the agent, then $Apply(a, s) = Sim(a, s)$. [Step 5]: Set $I = s$, and go to step 2.

In step 3 (planning episode), after the planning algorithm is terminated, the selection of the action to execute in step 4 can be implemented in many different ways. In a satisficing problem, if a path has been found to a goal (maximal utility) state, then the first step on that path should be selected. However, in most cases, such a path is unavailable, so the action is chosen based on the search space that has been explored so far, e.g., choose the first step in the path with the highest utility frontier node. For example, in the ALE domain, previous work selects the first step which leads to the highest discounted accumulated reward [1, 3]. Planning episodes can re-use work (search) performed in previous planning episodes, and all nodes generated in previous planning episodes can be cached.

Taylor and Korf (1993) proposed a standard method for dominated action sequence elimination in deterministic domains with transparent models, based on the following criterion for determining dominance: An action se-

quence S1 dominates S2 if and only if (1) $Cost(S1) \leq Cost(S2)$, (2) $S1$ is applicable whenever $S2$ is applicable, and (3) Applying $S1$ and $S2$ to state s always result in the same resulting state s'. In black-box planning, all available actions are always "applicable" – any action can be given as input to the simulator. However, action effects can not be predicted without executing the simulation, i.e., any conclusions about the equivalence of actions is only valid with respect to states which have actually been generated, and are not guaranteed to hold for states which have not yet been generated. Thus, it is not possible be certain whether Taylor and Korf's dominance criterion holds for any two action sequences in a black-box domain. Also, their method, which was originally designed for a single planning episode, learns a finite state machine (FSM) which detects and prunes sequences of actions leading to dominated states in a preprocessing step prior to the search; the FSM is used to prune actions during the single planning episode. In an online planning setting, instead of a single preprocessing/learning step, we can continuously apply learning, constantly improving a learned model across planning episodes. Therefore, we propose an approach which is adapted for online, black-box planning.

3 Dominated Action Sequence Pruning (DASP)

To improve the performance of planning episodes (Step 3) of the online black-box planning, we propose Dominated Action Sequence Pruning (DASP), a method to eliminate dominated action sequences in black-box planning.

The set of all actions which can be executed by an agent is the set of *available actions* (A_{avail}). For example, there are 18 available actions in ALE games. The *available action sequence set* A^L_{avail} is the set of sequences of available actions with the maximum length of L. Out of these, many are "useless" for a given domain that the resulting state of the action is always duplicated. We refer to such action sequences as *dominated action sequences*. More formally:

Definition 1. An action sequence set A^N dominates action sequence $\mathbf{a}$ if for all s there exists an action sequence $\mathbf{a}' \in A^N$ such that $succ(\mathbf{a}, s) = succ(\mathbf{a}', s)$

where $succ(a, s)$ is the successor generator function and $succ(\mathbf{a}, s)$ returns the state which results from applying the sequence of actions $\mathbf{a} = (a_0, a_1, ..., a_{n-1})$ to state s, i.e., $succ(\mathbf{a}, s) = s_n, s_i = succ(s_{i-1}, a_{i-1}), s_0 = s$.

A set of action sequences of maximum length L (A^L) is *sufficient* if it dominates all action sequences in A^L_{avail}.

Fact 1. Let A^L be a sufficient action sequence set. For every state s, a set of successor states of s generated by A^L is equal to a set of successor states of s generated by A^L_{avail}.

Proof. For all $\mathbf{a}' \in A^L_{avail} \setminus A^L$, as $\mathbf{a}'$ is dominated by A^L, $\{s' | s' = succ(\mathbf{a}, s), \mathbf{a} \in A^L\} = \{s' | s' = succ(\mathbf{a}, s), \mathbf{a} \in A \cup \{\mathbf{a}'\}\} = \{s' | s' = succ(\mathbf{a}, s), \mathbf{a} \in A^L \cup (A^L_{avail} \setminus A^L)\}$. □

Therefore, a sufficient action sequence set is sufficient to search the entire search space for the planning domain. To reduce the number of calls to $succ$, it is always beneficial to have smaller action sequence set. A *minimal* action sequence set is a sufficient action sequence set with the smallest cardinality.

Algorithm 1. [Find minimal action sequence set]

1. Initialize A^L_{min} to the set of all action sequences which generate one or more non-duplicate nodes.
2. Let $G = (V, E)$ be a hypergraph where $v_i \in V$ represents an action sequence $\mathbf{a}_i$ with no non-duplicate search nodes, and hyperedge $e(v_0, v_1, .., v_n) \in E$ if there exist one or more duplicate search nodes generated by all of $\mathbf{a}_0, \mathbf{a}_1, ..., \mathbf{a}_n$ but not by any other action sequences.
3. Add the minimal vertex cover of G to A^L_{min}.

During the first k planning episodes, DASP performs no pruning. After the k-th planning episode, L-step DASP finds an action sequence set A^L using Algorithm 1, based on the search tree explored in previous planning episodes. The input of Algorithm 1 is a set of sets of action sequences which generated one or more common duplicate nodes from the same state in any of the previous planning episodes. Based on this input, Algorithm 1 generates its hypergraph G with hyperedge $e(v_0, v_1, .., v_n)$ iff action sequences $a_0, a_1, a_2, ..$ corresponding to $v_0, v_1, v_2...$ generated common duplicate nodes from some state s, but not by any other actions sequences. In this way, Algorithm 1 returns a minimal set of action sequences which generated nodes by these action sequences covers all generated nodes in previous episodes (minimal set cover).

DASP only uses A^L for the rest of the planning episodes, except for the first node expansion in each episode. That is, when expanding a state s, which is reached by a trajectory $(a_0, a_1, ..., a_n)$, an action a is applied only if all of $(a), (a_n, a), (a_{n-1}, a_n, a), ...$ are in A^L. As a heuristic mechanism for recovering from incorrectly pruned action sequences, for the first node expansion in each episode, we apply all the available actions including dominated action sequences. If a dominated action sequence generates a non-duplicate node, then the action sequence will be put in A^L for the next planning episode by Algorithm 1.

Given a complete knowledge of whether a set of action sequences have duplicate nodes in common, Algorithm 1 returns a minimal action sequence set (Lemma 1).

Lemma 1. Assume for all subset of $A^L \subseteq A^L_{avail}$, we know whether there exists a state s that for all $\mathbf{a}, \mathbf{a}' \in A^L$, $succ(\mathbf{a}, s) = succ(\mathbf{a}', s)$. Algorithm 1 returns a minimal action sequence set.

We call it a *static* environment if all pair of action sequences either constantly generate duplicated nodes or never have them. In a static environment, expanding all action sequences from one node is sufficient to obtain a minimal action sequence set by Algorithm 1.

Definition 2. An environment (domain) is *static* if for all pair of action sequences $\mathbf{a}, \mathbf{a}'$, if $\exists\ s_0$ s.t. $succ(\mathbf{a}, s_0) = succ(\mathbf{a}', s_0)$, then $succ(\mathbf{a}, s) = succ(\mathbf{a}', s)$ holds $\forall s$.

Theorem 1. Assume a static environment. If an input to Algorithm 1 includes all generated nodes from node s_i

($succ(\mathbf{a}, s_i), \mathbf{a} \in A^L_{avail}$), then Algorithm 1 returns a minimal action sequence set.

Proof. In a static environment, for all s $succ(\mathbf{a}, s) = succ(\mathbf{a}', s)$ if $succ(\mathbf{a}, s_i) = succ(\mathbf{a}', s_i)$ (Definition 2). Therefore we have a complete and correct knowledge of action sequences which generates one or more non-duplicate nodes, and all set of action sequences with one or more duplicate search nodes in common. As we have a complete knowledge, by Lemma 1 , Algoirthm 1 returns a minimal action sequence set. □

Theorem 2. In a static environment, if an input includes a node s_i that we expanded all of $succ(\mathbf{a}, s_i), \mathbf{a} \in A^L_{avail}$, then Algorithm 1 returns a minimal action sequence set.

Proof. In a static environment, since the input to the Algorithm 1 in the first planning episode includes a node s_i for which we expanded all of $succ(\mathbf{a}, s_i), \mathbf{a} \in A^L_{avail}$, then by Theorem 2, Algorithm 1 returns a minimal action sequence set A^L. By the reordering criteria, all action sequences $\mathbf{a}(\in A^L) < \mathbf{a}'(\in A^L_{avail} \backslash A^L)$. As A^L is minimal and sufficient, every action sequences $\mathbf{a} \in A^L$ generates a new node with probability of 1, so $p(\mathbf{a}, 2) = 1$. Action sequences in a minimal action sequence set A^L always generate a new node as they are ordered in front, thus DASA expands them with probability of 1. As action sequences not in A^L are all dominated by A^L, and ordered after sequences in A^L, they always generate old nodes. Therefore, they are applied with a probability of $\epsilon(< 1)$. Therefore a set of action which DASA expands with probability of 1 is a minimal action sequence set. □

In many cases, an environment is not static. However, even in a dynamic environment, using Algorithm 1, we get an action sequence set smaller or equal to the size of a minimal action sequence set.

Fact 2. Let A^L be the action sequence set returned by Algorithm 1. $|A^L|$ is smaller or equal to the size of a minimal action sequence set.

Proof. Action sequences which generate one or more non-duplicate nodes can be modeled as a node with a self-loop edge in G. The size of a minimal vertex cover only decreases when we remove an edge from G. □

4 Dominated Action Sequence Avoidance (DASA)

DASP assumes a static environment where actions are either effective all the time or not at all. However, in many domains, most of the actions are conditionally effective, an action has a unique outcome for *some* states, but not for all states in the domain. We define *conditional dominance* as:

Definition 3. A set of action sequences A^L conditionally dominates an action sequence $\mathbf{a}$ if there exists a state s and an action sequence $\mathbf{a}' \in A^L$ s.t. $succ(\mathbf{a}, s) = succ(\mathbf{a}', s)$.

For example in pacman, the *up* action is only effective when there is no obstacle above the pacman. Thus, *up* is conditionally dominated by *neutral*.

Unfortunately, DASP does not work well on conditionally dominated action sequences. First, DASP may preemptively prune conditionally dominated action sequence out of the search completely if the action sequence is dominated in the first k planning steps (i.e., false positives). The only mechanism in DASP to recover an incorrectly pruned action sequence is when it is applicable from the first node in the search. Figure 1b shows the maximum size of action set detected as non-dominated actions throughout the game (from the k-th planning episode to the end of the game) in the ALE environment by 1-step DASP, compared to the human-generated restricted action set. This suggests that many of the actions are preemptively pruned and never recovered.

Second, even if an action sequence is almost always dominated, the inability of DASP to identify conditionally dominated action sequences means that such action sequences are always generated, significantly increasing the number of calls to the expensive $succ$ function.

Therefore, it should be more beneficial to consider whether an action is valid in the current context rather than to consider whether an action is used in the whole domain. To this end, we propose DASA, a method to estimate the probability of actions being dominated in the next planning episode.

Definition 4. Let "$<$" denote a total ordering on an action sequence set. $s' = succ(\mathbf{a}, s)$ is *new* if there is no action sequence $\mathbf{a}_m (< \mathbf{a}_n)$ such that $succ(\mathbf{a}_n, s) = succ(\mathbf{a}_m, s)$. A node is *old* if it is not new.

Let $p(\mathbf{a}, t)$ be the fraction of new nodes generated by action sequence $\mathbf{a}$ in the t-th planning episode. We define $p^*(\mathbf{a}, t+1)$ as:

$$p^*(\mathbf{a}, 0) = 1,$$
$$p^*(\mathbf{a}, t+1) = (p(\mathbf{a}, t) + \alpha p^*(\mathbf{a}, t))/(1+\alpha), \quad (1)$$

where α is a discount factor. $p^*(\mathbf{a}, t+1)$ is an estimate for the ratio of new nodes by action sequence $\mathbf{a}$ on $t+1$-th planning episode based on the experience of previous t planning episodes. If the number of nodes generated (including new/old nodes) by action sequence $\mathbf{a}$ in the t-th planning episode is 0, then $p^*(\mathbf{a}, t+1) = p^*(\mathbf{a}, t)$.

Unlike DASP which prunes sequences only after k-th planning episode, DASA prunes action sequences from the second episode using p^* value. DASA applies actions with higher p value more frequently, and action with lower p value less frequently. The trajectory to reach state s is given as $T = (a_0, a_1, ..., a_n)$. For each node expansion, action a is applied and $s' = succ(a, t)$ is generated with a probability:

$$P(a,t) = (1-\epsilon)s(p^*(a,t))s(p^*((a_{n-1},a),t)) \\ ...s(p^*((a_{n-l},...,a_n,a),t)) + \epsilon, \quad (2)$$

where $s(x)$ is an activation function, $\epsilon(< 1)$ is a parameter for the minimal probability for applying an action, and l is the length of the longest dominated action sequences to detect. We used a product of estimated proportions because longer paths have lesser training data, thus

unreliable. To avoid preemptive pruning based on scarce training data, we consider the p value of shorter paths, in addition to epsilon to smooth the probability.

As the definition of new/old depends on the ordering of actions, the action sequences should properly be ordered. DASA finds an action sequence set A^L using Algorithm 1. DASA orders the action sequences preferring sequences in A^L and breaking ties in favor of higher p value.

The additional overhead on the runtime due to DASP/DASA should be negligible when node expansion is slow enough, which is likely in many blackbox domains. For example, DASA(L=2) took 0.16 seconds on average per planning episodes in ALIEN while the planning per episode took 6 seconds on average with 10000 frame budget.

5 Experimental Evaluation

We evaluate our proposed dominated-action sequence detection strategies on a set of 53 single-player ALE games in the Arcade Learning Environment (ALE) [1]. The 53 games selected are the same as the set used in previous work [5]. Each game in ALE is a different, blackbox domain. Although ALE games are stochastic, ALE interface provides deterministic environment. The state of the game is represented by a fully observable, 128-byte RAM array. The meaning of the contents of this RAM is not known to the agent. However, ALE provides an API call which, for all games, returns the score for any state (higher is better). Following previous work on search-based planning on the ALE [3], we represent the state vector as 128 variables, with domain of 256 values. ALE provides a an API which returns the *restricted* action set, which is a *hand-coded* set of minimal actions for the game. We use this hand-coded restricted action set as one of the baselines against which our DASD strategies are evaluated.

We evaluated p-IW(1) [5], IW(1)[2], and Breadth-first search algorithms with the following action set determination policies.

- default: use all available action set
- restricted: use (hand-coded) restricted action set
- DASP1: 1-step DASP ($L = 1$).
- DASA1: 1-step DASA ($L = 1$).
- DASA2: 2-step DASA ($L = 2$).

Following [3], we discounted the accumulated reward as $R(s') = R(s) + \gamma^{d(s)+1} r(s, a)$ where s is the parent node of s', and the discount factor was set to $\gamma = 0.995$. A maximum budget of 10,000 simulated frames is applied. This simulated frames limit is roughly equivalent to limiting the lookahead time, as most of the time in the planning is spent in calls to the emulator. For example, in ALIEN, p-IW(1) with DASA2 used >99% of the time for running simulator.

As with previous work [1, 3], all algorithms cache simulation results, so simulations are not executed for cached states, and cached states do not count against the simulation frame budget. As the cost of reusing cached frame is negligible, we do not apply DASD pruning to cached states. We apply DASD pruning only to new frames which requires simulatins. Following previous work, all algorithms select an action every 5 frames [1, 3]. That is, a successor node $succ(a, s)$ in a search tree is a state after executing $s \leftarrow Apply(a, s)$ five times to the parent node, thus (simulation frame budget)/5 = max. # nodes generated.

For DASP we use all available actions until 12 planning frames (5x12=60 in-game frames = 1 second). As the minimal vertex cover is NP-hard [11], We calculate the optimal vertex cover if there are ≤ 5 nodes, and otherwise use a greedy algorithm which adds a vertex with the highest number of uncovered edges one by one. For DASA, we used a sigmoid function $s(x) = \frac{1}{1+e^{-5(x-0.5)}}$, minimal chance of applying action sequence $\epsilon = 0.04$, discount factor for p value $\alpha = 0.95$. To reduce the variance, each game was played 5 times, with the reported results averaged across these runs.

Table 1 and row "p-IW(1)" in Table 2 show the performance of DASA and DASP applied to p-IW(1). Overall, DASA2 outperformed the default action set and restricted action set. DASA and DASP successfully pruned dominated actin sequences and expanded more nodes than baseline. Although the maximum number of generated nodes is fixed for all methods, DASA and DASP spent fewer resources for unnecessary node generation, resulting in deeper search and therefore, better scores. Note that cached nodes are included in the number of expanded nodes, and the depth of the search tree also includes the cached nodes.

Figures 1a show the average number of actions applied per state using DASA2 compared to the size of restricted action set. Figures 1b show the maximum number of actions detected as non-duplicate using DASP compared to the size of restricted action set. For DASA, the number of actions applied is expected to be higher because it every action is appied with at least the minimal probability ϵ, so the number of applied actions exceeds the restricted action set size when the restricted action set size is small. The number of action applied is smaller when it successfully detects and prunes. DASP tends to have smaller action set size compared to restricted action set, especially when the restricted action set size is large. This is because DASP is aggressively pruning conditionally dominated actions before the states on which these actions have novel effects are discovered.

Table 2 shows the results for a smaller simulation frame budget (2000) on p-IW(1). With 2000 simulator frames, the maximum # of nodes generated (excluding cached nodes) is 400. Even with small search trees and thus less training data for learning, DASA and DASP successfully improves upon the baseline p-IW(1). DASA and DASP also improves the performance on IW(1) and BrFS (Table 2), showing that DASD is effective on wide range of search algorithms.

Results with Extended Action Set: To see the scalability of DASA and DASP with a larger available action set, we implemented an extended action set which adds two spurious buttons with no effect to the ALE controller (1 directional joystick, a fire button, and two spurious buttons), for a total of 72 actions. The additional buttons have no effect, so for every action in the ALE action set, there are 4

Table 1: Performance of p-IW(1) with different action sets on 53 ALE games using 10000 simulator frames per planning episode. "default" uses all actions available to the agent. "restricted" uses the human-made minimal action set provided by ALE. Scores are averaged over 5 runs with the same set of different random seeds for all algorithms. "expanded" shows the average number of nodes expanded including cached nodes for each planning, "depth" shows the average depth of the search tree including cached nodes.

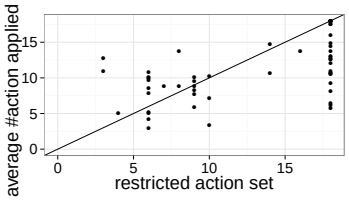
game	p-IW(1) DASA2			p-IW(1) DASA1			p-IW(1) DASP1			p-IW(1) default			p-IW(1) restricted		
	score	expanded	depth	score	expanded	depth	score	expanded	depth	score	expanded	depth	score	expanded	depth
ALIEN	**14018**	**406.0**	**116.1**	10992	258.1	73.3	3596	115.1	32.1	3596	115.2	33.8	3596	115.2	33.8
AMIDAR	**1504**	**401.6**	**162.8**	1223	279.5	80.4	939	116.7	39.5	296	116.3	36.6	1010	203.4	57.3
ASSAULT	1300	**314.0**	**60.7**	1365	209.5	42.5	1211	115.1	25.9	**1571**	114.7	26.1	1341	290.1	53.6
ASTERIX	234500	**313.1**	**107.2**	272600	197.2	67.5	288500	118.6	42.0	**315700**	118.7	42.0	292400	230.3	75.6
ASTEROIDS	**36354**	**216.1**	**56.1**	19720	145.8	47.0	9120	115.7	40.8	10192	115.5	40.8	34210	146.7	47.8
ATLANTIS	151400	323.5	214.3	182740	322.5	188.2	169700	121.6	64.3	183280	121.1	63.2	**190120**	**457.9**	**261.7**
BANK HEIST	**749**	**262.2**	**67.6**	342	171.1	47.6	232	114.7	33.0	232	114.7	33.0	232	114.7	33.0
BATTLE ZONE	12000	**208.1**	**26.9**	**19000**	160.7	21.4	7800	112.9	16.2	7800	112.9	16.2	7800	112.9	16.2
BEAM RIDER	**8130**	**333.1**	**159.6**	7513	264.9	83.3	3210	120.3	37.7	2825	120.8	35.7	5424	233.8	67.3
BERZERK	**528**	**172.9**	**52.0**	526	125.4	35.5	368	113.1	35.0	368	113.1	35.0	368	113.1	35.0
BOWLING	56	222.1	174.2	47	172.9	126.9	38	140.0	73.9	38	140.0	73.9	**71**	**329.5**	**254.3**
BREAKOUT	121	195.5	**172.8**	353	212.5	121.9	509	121.7	44.4	**603**	121.6	42.7	595	**251.5**	120.2
CARNIVAL	4760	284.9	**153.8**	5668	243.4	94.9	4830	123.4	44.8	4462	120.0	39.7	**6128**	**346.3**	130.9
CENTIPEDE	138036	**218.5**	**65.6**	151729	141.2	42.8	**156930**	114.9	34.9	154167	114.9	34.9	154167	114.9	34.9
CHOPPER COMMAND	2600	**158.7**	**27.5**	**4620**	129.7	24.1	3280	113.5	21.7	3280	113.5	21.7	3280	113.5	21.7
CRAZY CLIMBER	94920	151.1	63.7	128780	156.6	58.3	33720	109.3	38.8	125980	108.9	39.4	**131680**	**203.7**	**66.0**
DEMON ATTACK	24101	**431.8**	**129.2**	26523	248.4	63.7	**31860**	116.0	32.8	30348	115.8	32.2	30138	338.9	81.4
DOUBLE DUNK	**-7**	**152.2**	**25.0**	-14	121.6	21.1	-14	114.6	19.8	-14	114.6	19.8	-14	114.6	19.8
ELEVATOR ACTION	7340	**161.6**	**57.9**	**8720**	121.3	28.0	0	113.8	25.5	4480	114.1	26.1	4200	114.2	26.1
ENDURO	0	**338.6**	**45.6**	0	205.8	30.3	0	113.7	19.3	0	113.5	19.2	0	224.6	32.3
FISHING DERBY	-10	**262.7**	**49.8**	**-6**	179.0	27.3	-21	115.1	19.4	-31	114.4	18.9	-31	114.4	18.9
FREEWAY	**32**	267.5	116.0	32	191.8	73.4	30	125.0	43.5	29	125.3	43.7	32	**537.2**	**268.5**
FROSTBITE	**1114**	**162.3**	**61.2**	998	128.3	44.7	272	125.4	43.3	272	125.4	43.3	272	125.4	43.3
GOPHER	20144	**375.1**	**189.4**	23940	239.4	117.1	25073	118.7	57.9	**26444**	118.8	56.6	24932	256.4	119.5
GRAVITAR	**1570**	**182.0**	**27.8**	1420	139.6	22.3	990	115.7	19.7	990	115.7	19.7	990	115.7	19.7
HERO	**2176**	**164.8**	**46.3**	2052	123.5	36.1	1078	113.8	31.1	1078	113.8	31.1	1078	113.8	31.1
ICE HOCKEY	**1**	**152.7**	**33.6**	-3	122.5	26.3	-3	114.7	24.5	-5	114.7	24.5	-5	114.7	24.5
JAMESBOND	**170**	**276.2**	**48.4**	0	163.7	29.6	67	113.9	22.5	60	113.9	22.5	60	113.9	22.5
JOURNEY ESCAPE	-2500	**187.6**	**36.9**	-6820	139.0	28.3	-7180	114.8	24.2	-3120	114.5	23.9	**-160**	128.0	26.3
KANGAROO	4980	**217.6**	**168.1**	**5200**	189.3	89.7	1511	129.8	43.4	1600	130.2	43.8	1600	130.2	43.8
KRULL	7128	**189.7**	**54.8**	7104	176.9	52.5	2410	116.5	39.8	**10726**	115.2	36.4	**10726**	115.2	36.4
KUNG FU MASTER	53620	**216.6**	**77.0**	70440	171.5	54.7	68000	118.0	36.0	70880	117.8	36.0	**74960**	149.5	44.1
MONTEZUMA REVENGE	0	**205.3**	**45.6**	**20**	144.8	30.5	0	114.0	24.4	0	114.0	24.4	0	114.0	24.4
MS PACMAN	**29241**	**363.8**	**159.3**	29009	271.6	118.5	12068	116.2	54.8	7594	115.7	48.6	12826	227.4	87.6
NAME THIS GAME	12768	255.6	**114.4**	14618	158.9	48.7	**15110**	117.1	36.3	14550	117.1	36.3	13848	**340.7**	93.7
PONG	14	299.3	**119.9**	19	245.1	58.4	19	115.4	22.7	-2	115.0	22.0	**21**	**382.9**	97.4
POOYAN	11193	317.4	**207.8**	14604	282.7	151.1	15155	121.0	60.7	**15604**	120.5	60.1	14362	**340.4**	182.8
PRIVATE EYE	**100**	**154.0**	**26.2**	80	119.3	21.0	-500	112.6	19.9	-140	112.6	19.9	-140	112.6	19.9
QBERT	4670	167.5	153.2	13880	301.4	**159.0**	14160	121.4	53.5	10435	119.9	50.6	**16945**	**323.7**	130.5
RIVERRAID	**8182**	**193.5**	**51.0**	6920	146.8	41.7	8026	115.8	32.1	7972	115.8	32.1	7972	115.8	32.1
ROAD RUNNER	**13700**	**246.2**	**46.4**	9620	156.5	30.9	0	115.8	25.0	0	115.8	25.0	0	115.8	25.0
ROBOTANK	**3**	**145.1**	**30.2**	2	120.1	26.0	1	114.8	26.4	3	114.8	26.4	3	114.8	26.4
SEAQUEST	**1822**	**188.0**	**60.7**	1136	135.9	39.6	908	115.5	36.1	908	115.5	36.1	908	115.5	36.1
SPACE INVADERS	2261	259.0	65.8	2517	178.7	47.3	1646	116.7	32.3	1646	116.7	32.3	**2813**	**338.6**	**86.8**
STAR GUNNER	**1280**	**173.5**	**61.9**	1160	124.6	46.5	1211	117.5	44.8	1220	117.5	44.8	1220	117.5	44.8
TENNIS	**24**	**292.5**	**127.3**	20	196.6	71.4	11	120.9	38.2	11	120.9	38.2	11	120.9	38.2
TIME PILOT	44020	**313.0**	**160.6**	**44400**	237.7	88.9	32960	119.0	42.3	32960	119.1	42.9	38780	206.2	75.3
TUTANKHAM	203	191.3	**67.4**	**221**	148.6	31.8	195	114.1	24.2	202	114.0	23.8	150	**254.2**	49.8
UP N DOWN	12080	306.0	**111.7**	85140	237.7	66.1	58070	116.1	36.5	29400	115.8	36.5	**132320**	**338.4**	84.0
VENTURE	0	**286.8**	**94.7**	0	187.2	48.4	0	115.6	30.8	0	115.4	30.0	0	115.4	30.0
VIDEO PINBALL	197072	**418.2**	**111.6**	375420	225.6	58.1	**595012**	115.5	34.9	381514	115.4	34.7	442129	226.4	59.4
WIZARD OF WOR	50020	**302.9**	**104.0**	**69580**	202.5	52.2	31700	116.3	27.2	16040	115.3	24.8	36500	204.1	43.4
ZAXXON	**13540**	**178.9**	**36.4**	5320	124.5	25.7	22	113.2	23.5	40	113.2	23.5	40	113.2	23.5
Average	-	**254.9**	**82.8**	-	191.1	59.5	-	119.9	34.6	-	119.6	34.1	-	234.0	40.8

equivalent actions in the extended action (but an agent has no prior knowledge of this). Row "p-IW(1) (extd)" in Table 2 compares the scores using the extended action set on p-IW(1). We observed that p-IW(1) without DASD suffers from the sheer amount of available actions, and the number of nodes expanded was about 27% on average compared to the standard 18-action setting. Although DASA2 performed worse compared to the original 18 action setting, we observed that the number of applied actions per state is still decreasing at the end of the game.Therefore, given enough data for learning, we expect the performance of DASA2 to improve.

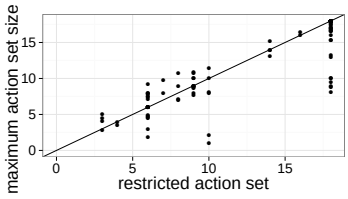
6 Conclusion

We proposed DASD, an approach to speeds up search in black-box planning domains with expensive node generations by avoiding the generation of duplicate nodes. DASD identifies action sequences which result in the same resulting state, and learns a minimal set of non-dominated actions (and action sequences), which is then used to restrict node generation so that duplicate states are not generated. We first proposed DASP, which learns a static minimal action set which is valid throughout the course of a game. We then proposed DASA, which learns conditional minimal action sets which are dependent on the current context of the game. We evaluated DASP and DASA on 53 games in the ALE arcade game environment, and showed that DASD significantly improves the performance of black-box planning in these domains compared to baseline algorithms without DASD. DASP was shown to yield performance comparable to using a human-generated set of minimal actions for a game, and DASA, by exploiting conditionally dominated actions, significantly outperformed both DASP and the human-generated minimal action set.

In this paper, we focused on eliminating short (1 or



(a) DASA2



(b) DASP1 (jittered)

search method	DASA2	DASA1	DASP1	default	restrict
p-IW(1)	**22**	10	4	6	10
p-IW(1) (2000)	**24**	14	6	5	7
IW(1)	**22**	9	7	7	8
BrFS	**18**	11	11	6	11
p-IW(1) (extd)	**39**	22	19	16	-

Table 2: #Best game

Figure 1: Figure 1a shows average number of actions applied for each node expansion using DASA2. Figure 1b shows the maximum size of action set detected as non-dominated in DASP1. Table 2 is comparison of the number of games scored the best. #Best includes ties, but excludes when all 5 algorithms have the same score. p-IW(1) (2000) is limited to 2000 simulation frames, while all others are limited to 10000 frames. For "extd", agent has two additional button in addition to the ALE controller, total of 72 available actions. The additional buttons have no effect on the game, but an agent has no prior knowledge of them. Overall, DASA2 outperformed other methods in all search algorithm.

2 step) dominated action sequences, and showed that this was sufficient for obtaining significant speedups. Many domains have many, longer dominated action sequences, but accurately learning longer sequences requires significantly more training data (longer / more numerous planning episodes). Future work will address this scaling issue.

Shleyfman et al. reported that p-IW(1) and IW(1) typically exhaust a search node and did not use the entire budget of 150000 simulated frames per planning episode (2016). In this case, DASP/DASA would not improve the score. However, p-IW(1) and IW(1) with 150000 frames run much slower than real-time. DASP/DASA helps close the gap with real-time because it enables searching deeper with a much lower budget.

In this paper, we evaluated DASD on the ALE environment, which has 18 available actions. However, the ALE games is a relatively simple environment, and modern gaming systems have significantly more complicated controllers (i.e., many more available actions). For example, Retro Learning Environment [12] has at least 720 available actions, and the Playstation3 game controller has at least 9x9x2x2x2x2 = 1296 available actions. In such complex environments search-based approaches will be overwhelmed by dominated actions without effective DASD. Our results for the extended action set (2 additional buttons) showed that the gap between DASD and the default grows significantly even with a modest increase in the number of available actions. Evaluation of DASD in more complex environments is future work.

In stochastic domains, one can repeatedly apply the same action to see the distribution of the outcome [9]. DASA should be able to extend to stochastic domains by redefining $succ(\mathbf{a}, s)$ as a set of possible resulting states. Applying DASA to stochastic domains (e.g. GVG-AI) is future work.

References

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

[2] Nir Lipovetzky and Hector Geffner. Width and serialization of classical planning problems. In *Proc. ECAI*, pages 540–545, 2012.

[3] Nir Lipovetzky, Miquel Ramirez, and Hector Geffner. Classical planning with simulators: Results on the Atari video games. In *Proc. IJCAI*, pages 1610–1616, 2015.

[4] Larry A. Taylor and Richard E. Korf. Pruning duplicate nodes in depth-first search. In *Proc. AAAI*, pages 756–761, 1993.

[5] Alexander Shleyfman, Alexander Tuisov, and Carmel Domshlak. Blind search for Atari-like online planning revisited. In *Proc. IJCAI*, pages 3251–3257, 2016.

[6] Maria Fox and Derek Long. The detection and exploitation of symmetry in planning problems. In *Proc. IJCAI*, volume 2, pages 956–961, 1999.

[7] Nir Pochter, Aviv Zohar, and Jeffrey S Rosenschein. Exploiting problem symmetries in state-based planners. In *Proc. AAAI*, pages 1004–1009, 2011.

[8] Martin Wehrle, Malte Helmert, Yusra Alkhazraji, and Robert Mattmüller. The relative pruning power of strong stubborn sets and expansion core. *Proc. ICAPS*, pages 1–9, 2013.

[9] Tomas Geffner and Hector Geffner. Width-based planning for general video-game playing. In *The IJCAI-15 Workshop on General Game Playing*, pages 15–21, 2015.

[10] Richard E. Korf. Real-time heuristic search. *Artif. Intell.*, 42(2-3):189–211, 1990.

[11] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[12] N. Bhonker, S. Rozenberg, and I. Hubara. Playing SNES in the Retro Learning Environment. *ArXiv e-prints*, November 2016.