GROUPING NODES WITH KNOWN VALUE DIFFERENCES: A LOSSLESS UCT-BASED ABSTRACTION ALGORITHM

Anonymous authors
Paper under double-blind review

ABSTRACT

A core challenge of Monte Carlo Tree Search (MCTS) is its sample efficiency, which can be addressed by building and using state and/or state-action pair abstractions in parallel to the tree search, such that information can be shared among nodes of the same layer. On the Go Abstractions in Upper Confidence bounds applied to Trees (OGA-UCT) is the state-of-the-art MCTS abstraction algorithm for deterministic environments that builds its abstraction using the Abstractions of State-Action Pairs (ASAP) framework, which aims to detect states and state-action pairs with the same value under optimal play by analysing the search graph. ASAP, however, requires two state-action pairs to have the same immediate reward, which is a rigid condition that limits the number of abstractions that can be found and thereby the sample efficiency. In this paper, we break with the paradigm of grouping value-equivalent states or state-action pairs and instead group states and state-action pairs with possibly different values as long as the difference between their values can be inferred. We call this abstraction framework Known Value Difference Abstractions (KVDA), which infers the value differences by analysis of the immediate rewards and modifies OGA-UCT to use this framework instead. The modification is called KVDA-UCT, which detects significantly more abstractions than OGA-UCT, introduces no additional parameter, and outperforms OGA-UCT on a variety of deterministic environments and parameter settings.

1 INTRODUCTION

Research into non-learning-based decision-making algorithms such as Monte Carlo Tree Search (MCTS) (Browne et al., 2012; Kocsis & Szepesvári, 2006) is an active field which is still worth dedicating research effort too. On the one hand MCTS can be used for applications where a general on-the-fly applicable decision-making algorithm is needed such as Game Studios which rarely use Machine Learning (ML) based AI as they would have to be retrained whenever the game and its rules are updated (e.g. during development or patches). And on the other hand, though not the scope of this paper, foundational work in MCTS might potentially translate to improvements of ML algorithms such as Alpha Zero (Silver et al., 2017) that are built on MCTS.

One way to improve MCTS is to reduce the search space by grouping states and actions in the current MCTS search tree to enable an intra-layer information flow (Jiang et al., 2014; Anand et al., 2015; 2016) by averaging the visits and returns of all abstract action nodes in the same abstract node used for the Upper Confidence Bounds (UCB) formula in the tree policy, which increases the sample efficiency. One key strength of one of the state-of-the-art abstraction algorithms On the Go Abstractions in Upper Confidence bounds applied to Trees (OGA-UCT) (Anand et al., 2016) is its exactness in the sense that if OGA-UCT groups two state-action pairs in a search tree where all possible successors of each state-action pair have been sampled, only state-action pairs are grouped that have the same Q^* value, i.e., they have the same value under subsequent optimal play. This exactness condition, however, comes at the cost that state or state-action pairs that only differ slightly in their Q^* value cannot be detected. This issue was slightly alleviated with the introduction of $(\varepsilon_\mathrm{a}, \varepsilon_\mathrm{t})$-OGA (Schmöcker et al., 2025), which is equivalent to $(\varepsilon_\mathrm{a}, 0)$-OGA in deterministic environments that allows for small errors in the immediate reward or transition function when building an abstraction. However, this also brings two downsides with it. Firstly, it introduces two parameters, which makes

tuning harder, and secondly, grouping non-value equivalent state-action pairs might even be harmful to the performance, as they can make convergence to the optimal action impossible.

In this work we propose Known Value Differences Abstractions UCT (KVDA-UCT), that relaxes the strict abstraction conditions of OGA-UCT to detect almost as many abstractions as $(\varepsilon_a, 0)$-OGA in deterministic environments but without losing the exactness condition. The novel idea that makes this possible is to deliberately group states or state-action pairs that do not have the same value if we know the difference between their values which is inferred by analysis of the search tree's immediate rewards. When the abstractions are used, instead of averaging state-action pair values directly, their difference-accounted values are averaged. The contributions of this paper can be summarized as follows:

1. We introduce the Known-Value-Difference (KVDA) abstraction framework that extends the Abstractions of State-Action Pairs (ASAP) framework used by OGA-UCT. Fig. 1 is an example of a simple state-transition graph in which KVDA finds three non-trivial abstractions, while ASAP would detect none.

2. We propose and empirically evaluate KVDA-UCT, a modification of OGA-UCT that introduces no parameters and uses and builds KVDA abstractions. We show that KVDA-UCT outperforms OGA-UCT in most of the here-considered deterministic environments. We also compare KVDA-UCT with $(\varepsilon_a, 0)$-OGA and show that it either performs equally well or better than a parameter-optimized $(\varepsilon_a, 0)$-OGA agent.

3. We also consider the stochastic setting where we generalize KVDA-UCT to ε_t-KVDA which allows for errors in the transition function when building the abstraction. Furthermore, we compare it to $(\varepsilon_t, \varepsilon_a)$-OGA and show that, unlike in the deterministic setting, KVDA rarely performs better than $(\varepsilon_t, \varepsilon_a)$-OGA in stochastic environments.

The paper is structured as follows. In **Section** 2, the theoretical groundwork for automatic state and state-action pair abstractions is laid. In particular, we define OGA-UCT and $(\varepsilon_a, \varepsilon_t)$-OGA. Then, in **Section** 3, our novel KVDA framework and both KVDA-UCT and ε_t-KVDA are introduced. Afterwards, in **Section** 4, the experimental setup is defined, then in **Section** 5, the experimental results using this setup are shown and discussed. The paper is concluded by a discussion of the limitations of KVDA-UCT and avenues for future work in **Section** 6.

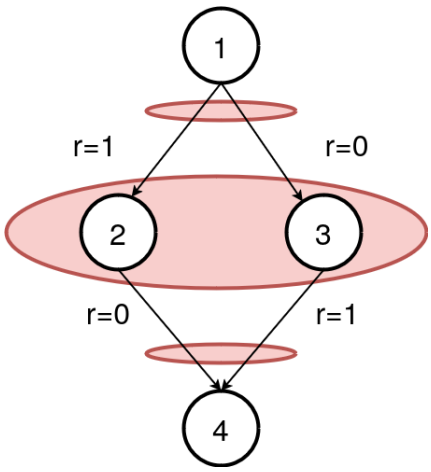


Figure 1: An example of an MDP state-transition graph where the state-of-the-art abstraction framework ASAP (Anand et al., 2015) would detect no abstractions while **our method Known-Value-Difference-Abstractions (KVDA)** detects three non-trivial abstractions. In this example, circles represent states, arrows represent deterministic state-transitions and arrow annotations denote the immediate transition reward. All actions or states that are intersected by a red ellipse will be abstracted by KVDA.

2 FOUNDATIONS OF AUTOMATIC ABSTRACTIONS

In this section, we will be laying the theoretical groundwork for this paper as well as introducing related work. For a comprehensive overview of non-learning-based abstractions, we refer to the survey paper by Schmöcker and Dockhorn (Schmöcker & Dockhorn, 2025).

2.1 PROBLEM MODEL AND OPTIMIZATION OBJECTIVE

For our purposes, finite Markov Decision Processes (Sutton & Barto, 2018) are used as the model for sequential, perfect-information decision-making tasks. $\Delta(X)$ denotes the probability simplex of a finite, non-empty set X and the power set of X is denoted by $\mathcal{P}(X)$.

Definition: An *MDP* is a 6-tuple $(S, \mu_0, T, \mathbb{A}, \mathbb{P}, R)$ where the components are as follows:

- $S \neq \emptyset$ is the finite set of states.
- $\mu_0 \in \Delta(S)$ is the probability distribution for the initial state.
- $\emptyset \neq T \subsetneqq S$ is the (possibly empty) set of terminal states.
- $\mathbb{A}\colon (S \setminus T) \mapsto A$ maps each state s to the available actions $\emptyset \neq \mathbb{A}(s) \subseteq A$ at state s where $|A| < \infty$.
- $\mathbb{P}\colon (S \setminus T) \times A \mapsto \Delta(S)$ is the stochastic transition function where $\mathbb{P}(s' |\, s, a)$ is used to denote the probability of transitioning from $s \in (S \setminus T)$ to $s' \in S$ after taking action $a \in \mathbb{A}(s)$ in s.
- $R\colon (S \setminus T) \times A \mapsto \mathbb{R}$ is the reward function.

From hereon, let $M = (S, \mu_0, T, \mathbb{A}, \mathbb{P}, R)$ be an MDP. We define $P \coloneqq \{(s, a) \mid s \in (S \setminus T), a \in \mathbb{A}(s)\}$ as the set of all legal state-action pairs. The goal is to find an agent π that is modelled as a mapping from states to action distributions $\pi\colon S \mapsto \Delta(A)$, such that π maximizes the expected episode's return where the (discounted) return of an episode $s_0, a_0, r_0, \ldots, s_n, a_n, r_n, s_{n+1}$ with $s_{n+1} \in T$ is given by $\gamma^0 r_0 + \ldots + \gamma^n r_n$.

2.2 ABSTRACTIONS OF STATE-ACTION PAIRS (ASAP)

For MCTS-based abstraction research, the goal has been to detect state-action pairs with the same Q^* value (the value under subsequent optimal play) in the search graph to increase sample efficiency by an intra-layer information flow (Jiang et al., 2014; Anand et al., 2015; 2016). In general, by abstractions of either the states of state-action-pairs we refer to equivalence relations over the state set S or state-action pair set P. The equivalence classes are abstract states or state-action pairs.

The current state of the art is the Abstraction of State-Action Pairs in UCT (ASAP) abstraction framework (Anand et al., 2015) that proposes rules to detect value-equivalent states and state-action pairs given an MDP transition graph and applies it to the current MCTS search graph (for details on MCTS, see Section A.7)). The core idea of ASAP is to alternatingly construct a state abstraction given a state-action pair abstraction and a state-action pair abstraction given a state abstraction.

Assume one is given a state abstraction $\mathcal{E}' \subseteq S \times S$. The corresponding ASAP state-action pair abstraction $\mathcal{H} \subseteq P \times P$ is defined as grouping those state-action pairs with the same immediate reward and equal abstract successor distribution. Concretely, any state-action-pair $(s_1, a_1), (s_2, a_2)$ is equivalent i.e. $((s_1, a_1), (s_2, a_2)) \in \mathcal{H}$ if and only if

$$\begin{aligned} F_{\text{a}} &\coloneqq |R(s_1, a_1) - R(s_2, a_2)| = 0 \\ F_{\text{t}} &\coloneqq \sum_{x \in \mathcal{X}} \left| \sum_{s' \in x} \mathbb{P}(s' |\, s_1, a_1) - \mathbb{P}(s' |\, s_2, a_2) \right| = 0, \end{aligned} \tag{1}$$

where $\mathcal{X}$ are the equivalence classes of $\mathcal{E}'$. And given a state-action pair abstraction $\mathcal{H}' \subseteq P \times P$, the corresponding ASAP state abstraction $\mathcal{E}$ groups all states whose actions can be mapped to each other, concretely:

$$\begin{aligned} &(s_1, s_2) \in \mathcal{E} \iff \\ &\forall a_1 \in \mathbb{A}(s_1)\, \exists a_2 \in \mathbb{A}(s_2) : ((s_1, a_1), (s_2, a_2)) \in \mathcal{H}_{i+1} \\ &\forall a_2 \in \mathbb{A}(s_2)\, \exists a_1 \in \mathbb{A}(s_1) : ((s_1, a_1), (s_2, a_2)) \in \mathcal{H}_{i+1}. \end{aligned} \tag{2}$$

OGA-UCT: Anand et al. (2016) proposed OGA-UCT, which builds an ASAP-like abstraction in parallel to running MCTS. When building the abstraction OGA starts with the initial state abstraction that groups all terminal states of the same layer and puts the remaining states in their own singleton equivalence class. Furthermore, when building the ASAP abstraction on the current search graph, OGA ignores non-yet-sampled successors of state-action pairs that appear in Equation 2. To make the frequent recomputation of the ASAP abstraction feasible, OGA keeps track of a recency counter for each Q-node and once it surpasses a certain threshold, recomputes its abstraction. If the abstraction changed, the parent states are recomputed too (and possibly their Q-node parents if their abstraction changed). By only locally checking for errors in the abstraction, OGA is able to keep track of an ASAP-like abstraction that is always close to the true ASAP abstraction of the current search tree.

The only MCTS component that OGA-UCT affects is the tree policy which is enhanced by using the aggregate returns and visits of a Q-node's abstract node to enhance the UCB value.

$(\varepsilon_\mathrm{a}, \varepsilon_\mathrm{t})$-OGA: The ASAP framework groups only value equivalent states and state-action pairs. This condition can be relaxed like already done by a predecessor of OGA-UCT, called AS-UCT (Jiang et al., 2014), by allowing the rewards in Equation 1 to differ by some threshold $\varepsilon_\mathrm{a} > 0$ and the transition error F_t of Equation 1 to lie in the interval $[0, \varepsilon_\mathrm{t}]$ where $0 \leq \varepsilon_\mathrm{t} \leq 2$ is another parameter *which does not have any effect in deterministic environments*. Since in general, positive threshold values do not induce an equivalence relation over state-action pairs, OGA-UCT has to be slightly modified to accommodate these approximate abstractions. This has been done by Schmöcker and Dockhorn (Schmöcker et al., 2025) who introduced $(\varepsilon_\mathrm{a}, \varepsilon_\mathrm{t})$-OGA.

2.3 State-only abstractions

The ASAP framework is the direct successor of Abstraction of States (AS) by Jiang et al. (2014) that abstracts states if their actions are pairwise similar in the sense that Equation 2 only has to be approximately satisfied as described in the $(\varepsilon_\mathrm{a}, \varepsilon_\mathrm{t})$-OGA section above.

A different abstraction approach that is not based on approximate homomorphisms (Ravindran & Barto, 2004) like AS, is given by Sokota et al. (2021) who discard newly sampled successor states of state-action pairs when their distance, given some domain-specific distance function, exceeds a threshold that decays over time.

Hostetler et al. (2015) proposed Progressive Abstraction Refinement for Sparse Sampling (PARSS) that is based on Sparse Sampling. Contrary to ASAP and AS, PARSS does not group states by analysing the search tree, but rather by initially optimistically grouping all successor states of a state-action pair and then iteratively refining the abstraction.

Yet another approach is presented by Dockhorn et al. (2021) who introduced S-MCTS that assumes the existence of a factored representation of states. States are grouped when their representation is equal when discarding a hand picked set of components.

2.4 Abstractions of the transition function

Research effort has also been dedicated towards automatic abstractions of the transition function, which on an abstract level can be described as pruning certain successors from the transition function (Sokota et al., 2021; Yoon et al., 2008; 2007; Saisubramanian et al., 2017).

3 Method

This section introduces our novel KVDA-UCT algorithm. First, we will provide an example of a concrete search graph in which ASAP misses abstractions that the Known Value Differences Abstractions (KVDA) framework would find which will be introduced after the search graph example. At the end of this section, we will be describing how KVDA is integrated into UCT to yield KVDA-UCT.

3.1 WHICH ABSTRACTIONS ASAP MISSES

Consider the state-transition graph that is illustrated in Fig. 1 that consists of four states and four actions. The ASAP framework would not detect any equivalences. In fact, any framework that aims at finding value-equivalent states or state-action pairs would at most be able to detect that the two root actions are value-equivalent. However, by analysing the state graph, one could derive that the Q^* values of the action of node 2 differs only by 1 from the Q^* value of the action of node 3. This holds true even if the state-graph would extend past node 4. Consequently, the V^* values of nodes 2 and 3 differ only by 1. This in turn implies that the two actions of node 1 must have the same Q^* value. This is the core idea behind our **Known-Value-Difference-Abstractions (KVDA)** framework which is to abstract states and state-action pairs if one can derive their value differences. Later, when using these abstractions to enhance MCTS, the differences only have to be subtracted when aggregating values. When viewed from the lenses of the KVDA framework, ASAP only groups states or state-action pairs with a value difference of 0, i.e. detects only true equivalences. Next, we will formalize the KVDA framework which both in theory and in our empirical evaluations (see Tab. 1, more details are given in the experimental section) detects strictly more abstractions.

3.2 THE KNOWN-VALUE-DIFFERENCE-ABSTRACTIONS FRAMEWORK

Our method, the Known-Value-Difference-Abstractions (KVDA) extends the ASAP definition by additionally grouping states or state-action-pairs whose V^* or Q^* difference is known. While ASAP iteratively builds abstractions on abstractions, KVDA bootstraps of an abstraction, difference-function pair. More concretely, given a state abstraction $\mathcal{E}'$ (or a state-action pair abstraction $\mathcal{H}'$) and a difference function $d'_{\text{s}} \colon S \times S \mapsto \mathbb{R}$ (or $d'_{\text{a}} \colon P \times P \mapsto \mathbb{R}$), both a state-action-pair abstraction $\mathcal{H}$ (or a state abstraction $\mathcal{E}$) is produced as well as a state-action pair difference function $d_{\text{a}} \colon P \times P \mapsto \mathbb{R}$ (or a state difference function $d_{\text{s}} \colon S \times S \mapsto \mathbb{R}$). Both d_{a} and d_{s} will be constructed such that

$$\begin{aligned} d_{\text{a}}(p_1, p_2) = d^*_{\text{a}}(p_1, p_2) &\coloneqq Q^*(p_2) - Q^*(p_1) \\ d_{\text{s}}(s_1, s_2) = d^*_{\text{s}}(s_1, s_2) &\coloneqq V^*(s_2) - V^*(s_1) \end{aligned} \tag{3}$$

for any pair of states or state-action pairs in the same equivalence class (i.e., in the same abstract state). Even though in the experimental section, we will mostly consider deterministic environments, the now-to-be-described KVDA framework will be applicable to any MDP, including stochastic ones. Next, starting with the base case, we will formalize how KVDA abstractions are built.

The base case: Like ASAP, KVDA groups all terminal states into the same initial abstract node, the remaining states are singleton abstract nodes. The difference function between all nodes in the terminal abstract node is initialized with 0.

State-action-pair abstractions: Using the same notation as in Section 2, let $\mathcal{E}' \subseteq S \times S$ be a state abstraction and $d'_{\text{s}} \colon S \times S \mapsto \mathbb{R}$ be a difference function. The corresponding KVDA state-action pair abstraction $\mathcal{H} \subseteq P \times P$ is defined as follows: Any state-action pair $(s_1, a_1), (s_2, a_2)$ is equivalent i.e. $((s_1, a_1), (s_2, a_2)) \in \mathcal{H}$ if and only if

$$\sum_{x \in \mathcal{X}} \left| \sum_{s' \in x} \mathbb{P}(s' \,|\, s_1, a_1) - \mathbb{P}(s' \,|\, s_2, a_2) \right| = 0, \tag{4}$$

where $\mathcal{X}$ are the equivalence classes of $\mathcal{E}'$. Note that this is almost identical to the ASAP definition except that the immediate rewards do not have to coincide. The difference function $d_{\text{a}} \colon P \times P \mapsto \mathbb{R}$ for two $(p_1, p_2) \in \mathcal{H}$ is given by

$$\begin{aligned} d_{\text{a}}(p_1, p_2) = &R(p_2) - R(p_1) \\ &+ \sum_{x \in \mathcal{X}} \sum_{s' \in x} \left(\mathbb{P}(s' \,|\, p_1) - \mathbb{P}(s' \,|\, p_2) \right) \cdot d'_{\text{s}}(s', s_x) \end{aligned} \tag{5}$$

where for each $x \in \mathcal{X}$, s_x is an arbitrarily chosen but fixed representative of $\mathcal{X}$. We will later see that the value of d_{a} is independent of this choice.

State abstractions: Given a state-action pair abstraction $\mathcal{H}' \subseteq P \times P$ and a state-action

pair difference function $d_{\mathrm{a}}\colon P \times P \mapsto \mathbb{R}$ the corresponding KVDA state abstraction $\mathcal{E} \subseteq S \times S$ groups all states whose actions can be mapped to each other, and whose mappings all have the same value difference:

$$
\begin{aligned}
&(s_1, s_2) \in \mathcal{E} \iff \exists d \in \mathbb{R} : \\
&\forall a_1 \in \mathbb{A}(s_1)\, \exists a_2 \in \mathbb{A}(s_2) : ((s_1, a_1), (s_2, a_2)) \in \mathcal{H}' \\
&\wedge\, d_{\mathrm{a}}((s_1, a_1), (s_2, a_2)) = d \\
&\forall a_2 \in \mathbb{A}(s_2)\, \exists a_1 \in \mathbb{A}(s_1) : ((s_1, a_1), (s_2, a_2)) \in \mathcal{H}' \\
&\wedge\, d_{\mathrm{a}}((s_2, a_2), (s_1, a_1)) = d.
\end{aligned}
\tag{6}
$$

The difference function $d_{\mathrm{s}}(s_1, s_2)$ is defined as the value d in the equation above.

Theoretical guarantees:

Convergence: Given an MDP, the above-described construction steps can be repeated until convergence, which is guaranteed as in our MDP definition, there are finitely many states and state-action pairs and each construction step either leaves the abstraction unchanged or reduces the number of equivalence classes. The abstraction that one obtains at convergence is called the KVDA abstraction of an MDP.

Soundness of d_a and d_s: Both d_{a} and d_{s} at convergence are equal to the differences of their arguments Q^* or V^* values as formulated in Eq. 3. This is proven in the supplementary materials in Section A.1.

3.3 KVDA-UCT AND ε_{T}-KVDA

In this section, we will describe how the KVDA abstraction framework is integrated into MCTS, which is fully analogous to how $(\varepsilon_{\mathrm{a}}, \varepsilon_{\mathrm{t}})$-OGA (which itself is equivalent to OGA-UCT for $\varepsilon_{\mathrm{a}} = \varepsilon_{\mathrm{t}} = 0$) integrates the ASAP framework. The usage of the following to-be-described modifications to $(\varepsilon_{\mathrm{a}}, \varepsilon_{\mathrm{t}})$-OGA is called $\boldsymbol{\varepsilon_{\mathrm{t}}}$**-KVDA**. For the case $\varepsilon_{\mathrm{t}} = 0$, the algorithm is simply called **KVDA-UCT**. $\boldsymbol{\varepsilon_{\mathrm{t}}}$**-KVDA** does not depend on ε_{a} unlike $(\varepsilon_{\mathrm{a}}, \varepsilon_{\mathrm{t}})$-OGA.

1) Instead of (approximate) ASAP abstractions, (approximate) KVDA abstractions of the search tree are built. Both $(\varepsilon_{\mathrm{a}}, \varepsilon_{\mathrm{t}})$-OGA and ε_{t}-KVDA allow the transition error F_{t} of Equation 1 to be in the interval $[0, \varepsilon_{\mathrm{t}}]$.

2) In addition to abstract Q nodes, abstract state nodes now also keep track of a representative. In our case, this is the first original node added to the abstract node, and if that one is removed, a random new representative is chosen.

3) Each abstract Q node's value is tracked as the value of its representative $\mathcal{Q}_{\mathrm{R}}$ that encodes the state-action-pair $p_{\mathrm{R}} \in P$. When another original node $\mathcal{Q}$ of the same abstract node representing $p \in P$ is backed up in the MCTS backup phase with value $v \in \mathbb{R}$, the value $v + d_{\mathrm{a}}(p, p_{\mathrm{R}})$ is added to the abstract node's statistics. In turn, $\mathcal{Q}$ extracts its aggregated returns from the abstract node by subtracting $d_{\mathrm{a}}(p, p_{\mathrm{R}})$ from the abstract node's value. If an abstract node's representative changes to $\mathcal{Q}'_{\mathrm{R}}$ encoding $p'_{\mathrm{R}} \in P$, then $n \cdot d_{\mathrm{a}}(p_{\mathrm{R}}, p'_{\mathrm{R}})$ is added to the statistics where n is the abstract visit count.

4) To reduce the computational load of finding a perfect match as the one required for the state abstractions in Equation 6, we check for a stricter condition for states s_1, s_2. It is first checked that within s_1 (and analogously s_2) all actions within the same abstract node have a value difference of zero. Then, it is tested for all abstract Q nodes of s_1 if the value difference between an arbitrarily chosen ground action of s_1 and one of s_2 is constant for all abstract nodes.

5) Finally, whenever a Q-node's recency counter reaches the threshold, the value difference to its representative is recalculated. A change in this difference also results in a reevaluation (and subsequent recency counter reset) of the parent nodes' abstractions and difference functions.

4 EXPERIMENT SETUP

In this section, we describe the general experiment setup. Any deviations from this setup will be explicitly mentioned.

Parameters: Originally, OGA-UCT (Anand et al., 2016) used the absolute value of the abstract Q value under consideration as the exploration constant for the UCB formula. However, this technique has been improved by the dynamic, scale-independent exploration factor global-std (*Citation excluded for double anonymous review process*). The global-std exploration constant has the form $C \cdot \sigma$ where σ is the standard deviation of the Q values of all nodes in the search tree and $C \in \mathbb{R}^+$ is some fixed parameter. Furthermore, we always use $K = 3$ as the recency counter which was proposed by Anand et al. (2016).

Problem models: For this paper, we ran our experiments on a variety of MDPs, all of which are either from the International Probabilistic Planning Conference (Grzes et al., 2014), are well-known board games, or are commonly used in the abstraction algorithm literature. Since we will compare KVDA-UCT to OGA-UCT we chose only environments with a non-constant and non-sparse reward function, as constant reward environments would imply that KVDA-UCT and OGA-UCT are semantically equivalent.

All MDPs originally feature stochasticity, however, for some experiments we considered their deterministic versions which are obtained by sampling a single successor of each state-action pair. For each episode, new successors are sampled. All experiments were run on the finite horizon versions of the considered MDPs with a default horizon of 50 steps and 200 for the board games with a planning horizon of 50 and a discount factor $\gamma = 1$.

The board games are transformed into MDPs by inserting deterministic MCTS with 500 iterations as the opponent. Furthermore, since they are all sparse-reward (i.e. either win or lose) they were also transformed into dense-reward MDPs by using heuristics. For each board game we defined a heuristic V^{h} that assigns each state a heuristic value for its state value. The reward of the (deterministic) transition s, a, s' is then given by $V^{\mathrm{h}}(s') - V^{\mathrm{h}}(s)$. The concrete heuristics used along with a brief description of all MDPs is provided in the supplementary materials in Section A.8. The deterministic MDPs are denoted by adding the prefix *d-* and the stochastic ones are denoted by the prefix *s-*.

Evaluation: Each data point that we denote in the remaining sections of this paper (e.g. agent returns) is the average of at least 2000 runs. Whenever we denote a confidence interval for a data point, then this is always a confidence interval with a confidence level of 99% provided by ≈ 2.33 times the standard error.

Normalized pairings score: We will later construct the *normalized pairings score* to be able to compare numerous parameter combinations across a variety of tasks. The score is constructed as follows. Let $\{\pi_1, \ldots, \pi_n\}$ be n agents (e.g. each KVDA-UCT agent along with its parameter setting is an agent) where each agent was evaluated on m tasks (later, a task will be a given MCTS iteration budget and an environment). The corresponding normalized pairings score matrix is of size $n \times n$ and the entry (i, j) is equal to the number of tasks where π_i performed better than agent π_j subtracted by the number of times it performed worse, divided by m. The normalized pairings score $s_i \leq i \leq n$ is given by averaging over the i-th row when excluding the i-th column.

Reproducibility: For reproducibility, we released our implementation which is available at `https://anonymous.4open.science/r/KVDA_UCT-6E51/README.md`. Our code was compiled with g++ version 13.1.0 using the -O3 flag (i.e. aggressive optimization).

5 EXPERIMENTS

This section presents the experimental results of KVDA-UCT. We considered two settings. Firstly, in Section 5 we measured the number of additional abstractions KVDA-UCT finds in comparison to OGA-UCT. Then, we evaluated KVDA-UCT on deterministic settings in which the losslessness of the abstraction is guaranteed in Section 5 and Section 5. Lastly, we present the results for stochastic environments in Section 5.

Abstractions that KVDA-UCT finds but OGA-UCT does not: Firstly, we empirically measured the number of non-trivial abstractions (i.e. those that aren't of size one) that KVDA-UCT, OGA-UCT, and $(\infty, 0)$-UCT find. For all deterministic environments, Tab. 1 denotes the average ratio of non-trivial abstract state-action pairs (synonymously abstract Q nodes) to the number of total abstract Q nodes in their respective search trees. In most environments, KVDA-UCT detects more abstractions than OGA-UCT, including environments where the abstraction rate more than doubles, such as SysAdmin or Wildfire. Furthermore, KVDA-UCT detects roughly as many abstractions as $(\infty, 0)$-OGA in most environments, which fully ignores rewards when building abstractions.

Table 1: The average ratio of abstract Q nodes in KVDA, OGA-UCT's and $(\infty, 0)$-OGA's respective search trees after 1000 iterations with $\lambda = 2$ that encompass more than one original node divided by the total number of abstract Q nodes. The states in which these search trees are built are sampled from an OGA-UCT agent with $\lambda = 2$ and 500 iterations. This measurement excludes all size-one abstract Q nodes whose original Q node has not yet reached the recency counter (see Section 2.2) required for the first abstraction update. Hence, a ratio of 1 means that no abstractions were found while a ratio of 0 means that every state-action pair has been abstracted with another. Note that KVDA-UCT (our method) finds more abstractions than OGA-UCT in nearly every environment and in general as many as $(\infty, 0)$-OGA. While there are some environments where there is no gain, such as Constrictor, there are other environments such as Wildfire or SysAdmin, where the abstraction rate more than doubles.

Domain	KVDA-UCT (ours)	OGA-UCT	$(\infty, 0)$-OGA
d-Academic Advising	0.69	0.73	0.68
d-Connect4	0.80	0.91	0.80
d-Constrictor	0.98	0.97	0.97
d-Cooperative Recon	0.44	0.56	0.48
d-Earth Observation	0.65	0.99	0.68
d-Elevators	0.28	0.32	0.29
d-Game of Life	0.54	0.56	0.55
d-Manufacturer	0.64	0.95	0.79
d-Othello	0.96	0.99	0.96
d-Pusher	0.96	0.99	0.96
d-Push Your Luck	0.23	0.48	0.16
d-Red Finned Blue Eye	0.72	0.84	0.70
d-Sailing Wind	0.70	0.92	0.74
d-Saving	0.96	0.96	0.96
d-Skills Teaching	0.59	0.65	0.59
d-SysAdmin	0.15	0.48	0.20
d-Tamarisk	0.35	0.56	0.39
d-Traffic	0.53	0.54	0.53
d-Wildfire	0.19	0.37	0.30
d-Wildlife Preserve	0.06	0.08	0.08

Parameter-optimized KVDA-UCT: Firstly, we compared KVDA-UCT, $(0, 0)$-OGA (i.e. standard OGA-UCT), and $(\varepsilon_{\text{a}}, 0)$-OGA with $\varepsilon_{\text{a}} > 0$ in terms of their parameter-optimized performances on deterministic environments. For all methods, we optimized over $C \in \{0.5, 1, 2, 4, 8, 16\}$ and some domain-specific values for ε_{a} that are listed in the supplementary materials in Tab. 4. Each parameter combination was evaluated on all here-considered deterministic environments with a budget of 100, 200, 500, and 1000 MCTS iterations. Tab. 2 list the performance for 100 and 1000 iterations, the tables for 200 and 500 iterations are found in the supplementary materials in Tab. 7 and 6. The results clearly show that KVDA either outperforms all other competitor methods or is tightly within the confidence bounds. There are a number of environments such as Manufacturer, Tamarisk, or Push Your Luck where KVDA-UCT simultaneously outperforms all competitor methods at once. Figure 2 plots the performances in dependence of the iteration budget for these tasks. The figures for the all other environments are found in the supplementary materials in Section A.6.

Single-parameter KVDA-UCT: Next, the generalization capabilities of KVDA-UCT in comparison to OGA-UCT and $(\varepsilon_{\text{a}}, 0)$-OGA on deterministic environments were tested. Like last section, we ran all algorithms using 100, 200, 500, and 1000 MCTS iterations whilst varying $C \in \{0.5, 1, 2, 4, 8, 16\}$ and ε_{a} with domain-specific values (see supplementary materials Tab. 4). Instead of considering the best-performances, we calculated the normalized pairings score for each parameter combination (except for $(\varepsilon_{\text{a}}, 0)$-OGA where we considered the maximum performance across all $\varepsilon_{\text{a}} > 0$ values a single parameter combination), constructed over the tasks which are the pairs of all environments and iteration budgets. Bar chart 3 shows the 6 agents with the highest score

Table 2: Average returns (↑) using 100 (right) and 1000 (left) MCTS iterations for KVDA-UCT, $(\infty, 0)$-OGA, $(0, 0)$-OGA (i.e. OGA-UCT), and the maximal performance of $(\varepsilon_a, 0)$-OGA when varying $\varepsilon_a > 0$. In the supplementary materials in Tab.4 the domain-specific ε_a-values are listed. All performances are the maximal performances obtained by varying the exploration constants $C \in \{0.5, 1, 2, 4, 8, 16\}$. Note that KVDA-UCT (our method) outperforms OGA-UCT in most environments. Furthermore, KVDA-UCT is either equal or performs better than parameter-optimized $(\varepsilon_a, 0)$-OGA even though KVDA introduces no extra parameter.

	1000 Iterations				**100 Iterations**			
Domain	$(0,0)$-OGA	$(\infty,0)$-OGA	$(\varepsilon_a,0)$-OGA	**KVDA (ours)**	$(0,0)$-OGA	$(\infty,0)$-OGA	$(\varepsilon_a,0)$-OGA	**KVDA (ours)**
d-Connect4	42.7 ± 0.6	46.8 ± 1.0	$\mathbf{47.9 \pm 0.6}$	47.5 ± 0.9	28.3 ± 0.3	$\mathbf{28.8 \pm 0.5}$	$\mathbf{28.8 \pm 0.5}$	28.4 ± 0.4
d-Constrictor	96.1 ± 0.3	95.0 ± 0.2	96.0 ± 0.3	$\mathbf{96.1 \pm 0.3}$	85.0 ± 0.6	84.1 ± 0.4	84.9 ± 0.9	$\mathbf{85.1 \pm 0.6}$
d-Othello	$\mathbf{181.2 \pm 1.2}$	160.2 ± 0.8	179.6 ± 1.2	180.7 ± 1.5	154.7 ± 1.3	146.8 ± 0.8	$\mathbf{155.1 \pm 1.8}$	154.8 ± 1.3
d-Pusher	$\mathbf{104.5 \pm 0.0}$	$\mathbf{104.5 \pm 0.0}$	$\mathbf{104.5 \pm 0.0}$	$\mathbf{104.5 \pm 0.0}$	$\mathbf{104.5 \pm 0.0}$	$\mathbf{104.5 \pm 0.0}$	$\mathbf{104.5 \pm 0.0}$	$\mathbf{104.5 \pm 0.0}$
d-Academic Advising	$\mathbf{-39.9 \pm 0.2}$	-40.0 ± 0.2	-40.1 ± 0.2	-40.0 ± 0.2	-56.6 ± 0.7	-56.2 ± 0.6	-56.2 ± 0.6	$\mathbf{-56.1 \pm 0.6}$
d-Cooperative Recon	16.1 ± 0.1	14.8 ± 0.1	16.0 ± 0.1	$\mathbf{16.2 \pm 0.1}$	10.8 ± 0.3	10.9 ± 0.3	11.0 ± 0.3	$\mathbf{11.0 \pm 0.3}$
d-Earth Observation	-7.18 ± 0.11	-30.0 ± 0.4	$\mathbf{-7.02 \pm 0.10}$	-30.0 ± 0.4	-10.2 ± 0.2	-28.4 ± 0.2	-28.4 ± 0.2	$\mathbf{-7.45 \pm 0.12}$
d-Elevators	-14.9 ± 0.4	-14.2 ± 0.3	$\mathbf{-13.9 \pm 0.4}$	-14.0 ± 0.3	-18.0 ± 0.3	-18.1 ± 0.3	$\mathbf{-17.9 \pm 0.3}$	-18.0 ± 0.3
d-Game of Life	$\mathbf{666.7 \pm 2.5}$	664.6 ± 2.5	664.8 ± 2.6	665.8 ± 2.5	641.8 ± 2.1	$\mathbf{642.1 \pm 2.2}$	$\mathbf{642.1 \pm 2.2}$	641.9 ± 2.1
d-Manufacturer	-1255.6 ± 15.0	-1658.4 ± 22.1	$\mathbf{-1158.2 \pm 19.4}$	-1246.0 ± 16.2	-1423.1 ± 15.9	-1680.3 ± 22.4	-1392.6 ± 20.2	$\mathbf{-1233.5 \pm 23.2}$
d-Push Your Luck	125.1 ± 1.9	66.7 ± 1.1	$\mathbf{137.9 \pm 2.2}$	132.4 ± 2.5	103.5 ± 1.5	66.3 ± 0.9	107.0 ± 1.6	$\mathbf{107.6 \pm 1.5}$
d-RedFinnedBlueEye	8191.3 ± 44.7	7930.0 ± 44.7	$\mathbf{8229.8 \pm 46.5}$	7950.6 ± 33.5	7683.4 ± 33.8	7464.4 ± 34.7	7495.1 ± 35.1	$\mathbf{7698.2 \pm 34.0}$
d-Sailing Wind	-40.0 ± 0.6	-39.1 ± 0.6	$\mathbf{-37.7 \pm 0.6}$	-38.9 ± 0.6	-64.7 ± 0.8	-64.9 ± 0.9	-64.9 ± 0.9	$\mathbf{-63.6 \pm 0.8}$
d-Saving	$\mathbf{66.0 \pm 0.2}$	63.0 ± 0.3	65.4 ± 0.2	65.4 ± 0.2	57.1 ± 0.2	55.2 ± 0.3	56.8 ± 0.2	$\mathbf{57.2 \pm 0.2}$
d-Skills Teaching	207.9 ± 4.6	211.3 ± 5.1	$\mathbf{216.2 \pm 4.5}$	211.3 ± 5.1	159.0 ± 3.9	158.3 ± 3.9	160.7 ± 3.8	$\mathbf{162.3 \pm 3.8}$
d-SysAdmin	477.1 ± 1.5	448.4 ± 1.3	$\mathbf{477.2 \pm 1.5}$	450.7 ± 1.1	475.5 ± 1.7	449.5 ± 1.2	450.1 ± 1.2	$\mathbf{479.1 \pm 1.4}$
d-Tamarisk	-214.5 ± 3.9	-208.2 ± 2.3	$\mathbf{-185.6 \pm 2.0}$	-206.4 ± 2.7	-315.8 ± 6.5	-285.4 ± 4.8	-284.9 ± 4.7	$\mathbf{-263.0 \pm 4.8}$
d-Traffic	$\mathbf{-1.54 \pm 0.07}$	-1.61 ± 0.07	-1.58 ± 0.07	-1.61 ± 0.07	$\mathbf{-9.21 \pm 0.22}$	$\mathbf{-9.21 \pm 0.22}$	$\mathbf{-9.21 \pm 0.22}$	$\mathbf{-9.21 \pm 0.22}$
d-Wildfire	-195.6 ± 55.9	-503.5 ± 36.1	$\mathbf{-179.9 \pm 49.2}$	-415.0 ± 36.1	-194.1 ± 36.1	-498.7 ± 36.1	-408.9 ± 36.1	$\mathbf{-173.2 \pm 36.1}$
d-Wildlife Preserve	1388.0 ± 0.8	1387.9 ± 0.7	$\mathbf{1388.5 \pm 0.7}$	1387.9 ± 0.7	$\mathbf{1388.9 \pm 0.9}$	$\mathbf{1388.9 \pm 0.9}$	$\mathbf{1388.9 \pm 0.9}$	$\mathbf{1388.9 \pm 0.9}$

as well as the agent with the lowest score. Both the top spots are occupied by our KVDA-method, followed by OGA-UCT and $(\varepsilon_a, 0)$-OGA. The best performing algorithm overall is KVDA-UCT with $C = 4$.

ε_t-KVDA on stochastic environments: Lastly, the performance of ε_t-KVDA on stochastic environments were tested by running both ε_t-KVDA and $(\varepsilon_a, \varepsilon_t)$-OGA on the stochastic versions of the here-considered environments. We considered the iteration budgets of 100 and 1000 iterations and varied $C \in \{0.5, 1, 2, 4, 8, 16\}$, $\varepsilon_t \in \{0, 0.4, 0.8, 1.2, 1.6\}$ and the same domain-specific ε_a that were used in the previous sections whose values are listed in the supplementary materials in Tab. 4. For each environment, Tab. 5 of the supplementary materials lists the parameter-optimized performances of both algorithms. In contrast to the deterministic setting, KVDA does not outperform OGA. For most environments both perform equally well with some exceptions such as Manufacturer were ε_t-KVDA clearly performs best and Tamarisk were $(\varepsilon_a, \varepsilon_t)$-OGA performs best. We believe mediocre performance of ε_t-KVDA stems from the fact that since ε_t-KVDA essentially ignores immediate rewards when it builds abstractions, the number of faulty abstractions that occur in this approximate setting is increased. In the deterministic setting, no faulty abstractions are built.

6 Conclusion, Limitations and Future Work

This paper introduced KVDA-UCT, an extension to OGA-UCT, that additionally groups states and state-action pairs that do not have the same Q^* or V^* value as long as the difference is known. We evaluated and compared KVDA-UCT to OGA-UCT and $(\varepsilon_a, 0)$-OGA on a deterministic setting where KVDA-UCT outperforms all its competitors both in terms of generalization as well as the parameter-optimized performance. This does not hold true for the stochastic setting for which we generalized KVDA-UCT to ε_t-KVDA which is only clearly better than $(\varepsilon_a, \varepsilon_t)$-OGA in few environments.

A first avenue for future work will be to further investigate the reasons for the mediocre performance of the experiments in this paper of the stochastic setting and develop an extension of ε_t-KVDA suited for this setting. Another limitation of KVDA abstractions in its current form is its limitation to MDPs (i.e. single-player games). Even though MCTS is in principle applicable to multi-player games, KVDA is not because in any state, there is in general, no unique V^* value from which differences can be built, as there are potentially many equilibria with different payoff vectors. Future work will be to extend KVDA to this setting.

7 REPRODUCIBILITY STATEMENT

In our experiment setup, we have a subsection called *Reproducibility* in which we provide a download link to the full codebase used for this project as well as compilation details. The codebase contains an elaborate README detailing the steps to reproduce the experiments.

REFERENCES

Ankit Anand, Aditya Grover, Mausam, and Parag Singla. ASAP-UCT: Abstraction of State-Action Pairs in UCT. In Qiang Yang and Michael J. Wooldridge (eds.), *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 1509–1515. AAAI Press, 2015. URL http://ijcai.org/Abstract/15/216.

Ankit Anand, Ritesh Noothigattu, Mausam, and Parag Singla. OGA-UCT: on-the-go abstractions in UCT. In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'16, pp. 29–37. AAAI Press, 2016. ISBN 1577357574.

Anonymous Authors. Grouping nodes with known value differences: A lossless UCT-based abstraction algorithm, 2025. Anonymized (for double blind review purposes) Repository available at: https://anonymous.4open.science/r/KVDA_UCT-6E51/README.md.

Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intell. AI Games*, 4 (1):1–43, 2012. doi: 10.1109/TCIAIG.2012.2186810. URL https://doi.org/10.1109/TCIAIG.2012.2186810.

Alexander Dockhorn, Jorge Hurtado Grueso, Dominik Jeurissen, and Diego Perez Liebana. STRATEGA: A General Strategy Games Framework. In Joseph C. Osborn (ed.), *Joint Proceedings of the AIIDE 2020 Workshops co-located with 16th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2020), Worcester, MA, USA, October 19-23, 2020 (online)*, volume 2862 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020. URL https://ceur-ws.org/Vol-2862/paper30.pdf.

Alexander Dockhorn, Jorge Hurtado-Grueso, Dominik Jeurissen, Linjie Xu, and Diego Perez-Liebana. Game State and Action Abstracting Monte Carlo Tree Search for General Strategy Game-Playing. In *2021 IEEE Conference on Games (CoG)*, pp. 1–8, 2021. doi: 10.1109/CoG52621.2021.9619029.

Martin Gardner. The fantastic combinations of John Conway's new solitaire game 'life'. *Scientific American*, 223(4):120–123, October 1970. doi: 10.1038/scientificamerican1070-120. URL https://www.jstor.org/stable/24927642.

Marek Grzes, Jesse Hoey, and Scott Sanner. International Probabilistic Planning Competition (IPPC) 2014. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2014.

J. T. Guerin, J. P. Hanna, L. Ferland, N. Mattei, and J. Goldsmith. The Academic Advising Planning Domain. In *Proceedings of the Workshop on International Planning Competition (WS-IPC)*, 2012.

Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient Solution Algorithms for Factored MDPs. *J. Artif. Intell. Res.*, 19:399–468, 2003. doi: 10.1613/JAIR.1000. URL https://doi.org/10.1613/jair.1000.

Andreas Hertle, Christian Dornhege, Thomas Keller, Robert Mattmüller, Manuela Ortlieb, and Bernhard Nebel. An Experimental Comparison of Classical, FOND and Probabilistic Planning. In Carsten Lutz and Michael Thielscher (eds.), *KI 2014: Advances in Artificial Intelligence - 37th Annual German Conference on AI, Stuttgart, Germany, September 22-26, 2014. Proceedings*, volume 8736 of *Lecture Notes in Computer Science*, pp. 297–308. Springer, 2014. doi: 10.1007/978-3-319-11206-0_29. URL https://doi.org/10.1007/978-3-319-11206-0_29.

Jesse Hostetler, Alan Fern, and Thomas G. Dietterich. Progressive Abstraction Refinement for Sparse Sampling. In Marina Meila and Tom Heskes (eds.), *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands*, pp. 365–374. AUAI Press, 2015. URL http://auai.org/uai2015/proceedings/papers/81.pdf.

Nan Jiang, Satinder Singh, and Richard L. Lewis. Improving UCT planning via approximate homomorphisms. In Ana L. C. Bazzan, Michael N. Huhns, Alessio Lomuscio, and Paul Scerri (eds.), *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, pp. 1289–1296. IFAAMAS/ACM, 2014. URL http://dl.acm.org/citation.cfm?id=2617453.

Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou (eds.), *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, volume 4212 of *Lecture Notes in Computer Science*, pp. 282–293. Springer, 2006. doi: 10.1007/11871842_29. URL https://doi.org/10.1007/11871842_29.

B. Ravindran and A. G. Barto. Approximate Homomorphisms: A Framework for Non-Exact Minimization in Markov Decision Processes. In *Proc. Int. Conf. Knowl.-Based Comput. Syst.*, pp. 1–10, 2004.

S. Saisubramanian, S. Zilberstein, and P. Shenoy. Optimizing Electric Vehicle Charging Through Determinization. In *ICAPS Workshop on Scheduling and Planning Applications*, 2017.

Scott Sanner and Sungwook Yoon. International Probabilistic Planning Competition (IPPC) 2011. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2011.

Robin Schmöcker and Alexander Dockhorn. A survey of non-learning-based abstractions for sequential decision-making. *IEEE Access*, 13:100808–100830, 2025. doi: 10.1109/ACCESS.2025.3572830.

Robin Schmöcker, Lennart Kampmann, and Alexander Dockhorn. Time-critical and confidence-based abstraction dropping methods. In *2025 IEEE Conference on Games (CoG)*, pp. 1–8, 2025. doi: 10.1109/CoG64752.2025.11114261.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *CoRR*, abs/1712.01815, 2017. URL http://arxiv.org/abs/1712.01815.

Samuel Sokota, Caleb Ho, Zaheen Farraz Ahmad, and J. Zico Kolter. Monte Carlo Tree Search With Iteratively Refining State Abstractions. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 18698–18709, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/9b0ead00a217ea2c12e06a72eec4923f-Abstract.html.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2nd edition, 2018.

R. Vanderbei. Optimal Sailing Strategies. Technical report, University of Princeton, Statistics and Operations Research Program, 1996.

Sung Wook Yoon, Alan Fern, and Robert Givan. FF-Replan: A Baseline for Probabilistic Planning. In Mark S. Boddy, Maria Fox, and Sylvie Thiébaux (eds.), *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, pp. 352. AAAI, 2007. URL http://www.aaai.org/Library/ICAPS/2007/icaps07-045.php.

Sung Wook Yoon, Alan Fern, Robert Givan, and Subbarao Kambhampati. Probabilistic Planning via Determinization in Hindsight. In Dieter Fox and Carla P. Gomes (eds.), *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pp. 1010–1016. AAAI Press, 2008. URL http://www.aaai.org/Library/AAAI/2008/aaai08-160.php.

A Supplementary Materials

A.1 Proof that $d_a^* = d_a$ and $d_s^* = d_s$

In this section, it will be shown that the KVDA difference functions d_a and d_s at any step (including the step of convergence) coincide with d_a^* and d_s^* for state or state-action pairs within the same abstract state. This will be proven inductively. First, note that this statement is true for the base case as all terminal states have a value of $V^* = 0$; hence, their differences are also 0. For the following induction steps, assume that d_a is bootstrapped by d'_s and d_s is bootstrapped by d'_a. Next, let $(p_1, p_2) \in \mathcal{H}$. By definition, it holds that

$$d^*(p_1, p_2) = R(p_2) - R(p_1) + \underbrace{\sum_{s \in S} (\mathbb{P}(s|\, p_2) - \mathbb{P}(s|\, p_1)) V^*(s)}_{L :=}. \tag{7}$$

By rewriting $V^*(s)$ in terms of the difference to its abstract representative, one obtains

$$\begin{aligned}
L &= \sum_{x \in \mathcal{X}} \sum_{s' \in x} (\mathbb{P}(s'|\, p_2) - \mathbb{P}(s'|\, p_1)) \, (V^*(s_x) - \underbrace{d_\mathrm{s}^*(s', s_x))}_{= d'_\mathrm{s} \text{ by induction}} \\
&= \sum_{x \in \mathcal{X}} \sum_{s' \in x} (\mathbb{P}(s'|\, p_2) - \mathbb{P}(s'|\, p_1)) V^*(s_x) \\
&+ \sum_{x \in \mathcal{X}} \sum_{s' \in x} (\mathbb{P}(s'|\, p_1) - \mathbb{P}(s'|\, p_2)) \cdot d_\mathrm{s}^*(s', s_x) \\
&= \sum_{x \in \mathcal{X}} V^*(s_x) \underbrace{\sum_{s' \in x} (\mathbb{P}(s'|\, p_2) - \mathbb{P}(s'|\, p_1))}_{=0} \\
&+ \sum_{x \in \mathcal{X}} \sum_{s' \in x} (\mathbb{P}(s'|\, p_1) - \mathbb{P}(s'|\, p_2)) \cdot d'_\mathrm{s}(s', s_x) \\
&= d_\mathrm{a}(p_1, p_2) + R(p_1) - R(p_2).
\end{aligned} \tag{8}$$

Hence $d^*(p_1, p_2) = d_\mathrm{a}(p_1, p_2)$. Note that this proof holds for any choice of the abstract nodes' representatives. Lastly, let $(s_1, s_2) \in \mathcal{E}$ and $d_\mathrm{s}(s_1, s_2) = d$. First note that if $a \in \arg\max_{a' \in \mathbb{A}(s_1)} Q^*(s_1, a')$ and $((s_1, a), (s_2, \hat{a})) \in \mathcal{H}'$ with $d_\mathrm{a}^*((s_1, a), (s_2, \hat{a})) = d$ for some $\hat{a} \in \mathbb{A}(s_2)$ then $\hat{a} \in \arg\max_{a' \in \mathbb{A}(s_2)} Q^*(s_2, a')$ because for all $b \in \mathbb{A}(s_2)$ it holds that

$$\begin{aligned}
Q^*(s_2, \hat{a}) &= Q^*(s_1, a) - d_\mathrm{a}^*((s_2, \hat{a}), (s_1, a)) \\
&\geq Q^*(s_1, \hat{b}) - d_\mathrm{a}^*((s_2, \hat{a}), (s_1, a)) \\
&= Q^*(s_2, b) - d_\mathrm{a}^*((s_2, \hat{b}), (s_1, b)) - d_\mathrm{a}^*((s_2, \hat{a}), (s_1, a)). \\
&= Q^*(s_2, b) - (-d) + d = Q^*(s_2, b),
\end{aligned} \tag{9}$$

where $((s_1, \hat{b}), (s_2, b)) \in \mathcal{H}'$ with $d_\mathrm{a}^*((s_1, \hat{b}), (s_2, b)) = d$. And since $V^*(s) = \max_{a \in \mathbb{A}(s)} Q^*(s, a)$ it directly follows that $d = d_\mathrm{s}(s_1, s_2) = d_\mathrm{s}^*(s_1, s_2)$ □.

A.2 Runtime measurements

Tab. 3 lists the average decision-making times for each environment of KVDA-UCT compared to OGA-UCT for 100 and 2000 iterations on states sampled from a distribution induced by random

walks. The numbers show a median runtime overhead of $< 1\%$ for 100 iterations and $\approx 1\%$ for 2000 iterations.

Table 3: Average decision-making times in milliseconds of KVDA-UCT compared to OGA-UCT This data was obtained using an Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz.

Domain	KVDA-100	OGA-100	KVDA-2000	OGA-2000
Academic Advising	3.85	3.82	133.82	131.74
Cooperative Recon	5.26	5.23	221.98	220.19
Earth Observation	7.54	7.54	197.38	199.13
Elevators	6.85	6.79	231.93	227.69
Game of Life	4.99	4.96	139.38	134.06
Manufacturer	10.26	11.01	281.03	294.91
Red Finned Blue Eye	6.69	6.66	152.46	151.56
Sailing Wind	4.11	4.09	139.39	140.82
Saving	3.75	3.71	124.22	111.13
Skills Teaching	4.97	4.94	219.90	216.68
SysAdmin	6.27	6.20	150.19	152.72
Tamarisk	4.65	4.64	167.23	148.03
Traffic	5.44	5.42	161.33	159.35
Push Your Luck	5.84	5.78	170.25	175.19
Wildfire	7.46	7.43	368.66	476.95
Wildlife Preserve	7.06	7.05	2020.35	1685.53
Othello	18.63	18.67	434.88	428.34
Connect4	4.40	4.33	133.35	123.58
Constrictor	23.23	23.10	616.38	611.72

A.3 Domain-specific ε_a values

One problem with $(\varepsilon_\mathrm{a}, \varepsilon_\mathrm{t})$-OGA is that while ε_t is neatly bounded by 0 and 2, the ε_a value has to be chosen on a per-environment basis since for example the value $\varepsilon_\mathrm{a} = 0.5$ would be equivalent to $\varepsilon_\mathrm{a} = 0$ for any environment with rewards that are all greater than 0.5. Tab. 4 lists the hand picked values for each environment that we chose for the experiment section. The values were chosen to be at the boundary of rewards that actually occur in these environments.

Table 4: A list of the environment-specific ε_a values that were evaluated in the experiment section for $(\varepsilon_\mathrm{a}, 0)$-OGA. All domains that are not explicitly listed here, used the default values $\varepsilon_\mathrm{a} \in \{1, 2, \infty\}$.

Environment	ε_a values
Academic Advising	∞
Wildlife Preserve	5, 10, 20, ∞
Red Finned Blue Eye	50, 100, 200, ∞
Wildfire	5, 10, 100, ∞
Push Your Luck	2, 5, 10, ∞
Skill Teaching	2, 3, ∞
Tamarisk	0.5, 1.0, ∞
Cooperative Recon	0.5, 1.0, ∞
Manufacturer	10, 20, ∞
Connect 4	1, 5, 10, ∞
Othello	5, 10, 20, ∞
Constrictor	10, 20, 30, ∞
Default	1, 2, ∞

A.4 PERFORMANCES OF ε_t-KVDA

Table 5: The parameter-optimized performances of ε_{t}-KVDA (our method) and $(\varepsilon_{\text{a}}, \varepsilon_{\text{t}})$-OGA on the stochastic versions of the here-presented environments using 1000 MCTS iterations (left) and 100 MCTS iterations (right). The environments Push Your Luck and Wildlife Preserve from the deterministic experiments were excluded as computing their transition probabilities which is a requirement for both ε_{t}-KVDA and $(\varepsilon_{\text{a}}, \varepsilon_{\text{t}})$-OGA is infeasible. Furthermore, Red Finned Blue Eye, Wildfire and Elevators were also excluded as the performances' variances were so high in these settings such that obtaining low confidence bounds was infeasible. Contrary to the stochastic setting, ε_{t}-KVDA does not consistently perform better than OGA. For example, while ε_{t}-KVDA can gain a clear advantage in Manufacturer and Sailing Wind, $(\varepsilon_{\text{a}}, \varepsilon_{\text{t}})$-OGA is significantly better in Tamarisk.

Average returns (↑) for 1000 MCTS iterations

Domain	$(\varepsilon_{\text{a}}, \varepsilon_{\text{t}})$-OGA	ε_{t}-**KVDA** (ours)
s-Academic Advising	-63.9 ± 0.8	$\mathbf{-63.9 \pm 0.8}$
s-Cooperative Recon	$\mathbf{14.3 \pm 0.2}$	14.1 ± 0.2
s-Earth Observation	-7.97 ± 0.22	$\mathbf{-7.95 \pm 0.23}$
s-Game of Life	572.9 ± 2.3	$\mathbf{573.4 \pm 2.2}$
s-Manufacturer	-1141.0 ± 11.1	$\mathbf{-863.3 \pm 10.7}$
s-Sailing Wind	-62.1 ± 1.3	$\mathbf{-61.8 \pm 1.3}$
s-Saving	$\mathbf{50.7 \pm 0.1}$	50.6 ± 0.1
s-Skills Teaching	71.1 ± 7.8	$\mathbf{74.9 \pm 6.9}$
s-SysAdmin	$\mathbf{403.3 \pm 2.0}$	403.1 ± 2.1
s-Tamarisk	$\mathbf{-532.9 \pm 8.2}$	-563.6 ± 8.3
s-Traffic	$\mathbf{-13.4 \pm 0.3}$	-13.6 ± 0.3

Average returns (↑) for 100 MCTS iterations

Domain	$(\varepsilon_{\text{a}}, \varepsilon_{\text{t}})$-OGA	ε_{t}-**KVDA** (ours)
s-Academic Advising	-88.0 ± 1.2	$\mathbf{-87.8 \pm 1.2}$
s-Cooperative Recon	$\mathbf{7.08 \pm 0.37}$	7.07 ± 0.35
s-Earth Observation	$\mathbf{-13.7 \pm 0.3}$	-13.8 ± 0.3
s-Game of Life	$\mathbf{498.7 \pm 3.2}$	497.6 ± 3.0
s-Manufacturer	-1457.6 ± 21.5	$\mathbf{-1159.6 \pm 24.8}$
s-Sailing Wind	-78.3 ± 1.1	$\mathbf{-73.2 \pm 1.2}$
s-Saving	$\mathbf{44.6 \pm 0.2}$	44.5 ± 0.2
s-Skills Teaching	$\mathbf{-8.94 \pm 8.37}$	-9.60 ± 8.23
s-SysAdmin	$\mathbf{332.7 \pm 2.8}$	331.1 ± 2.7
s-Tamarisk	$\mathbf{-767.9 \pm 9.8}$	-836.1 ± 7.8
s-Traffic	$\mathbf{-22.0 \pm 0.4}$	$\mathbf{-22.0 \pm 0.4}$

A.5 PERFORMANCES OF PARAMETER-OPTIMIZED KVDA-UCT FOR 200 AND 500 ITERATIONS

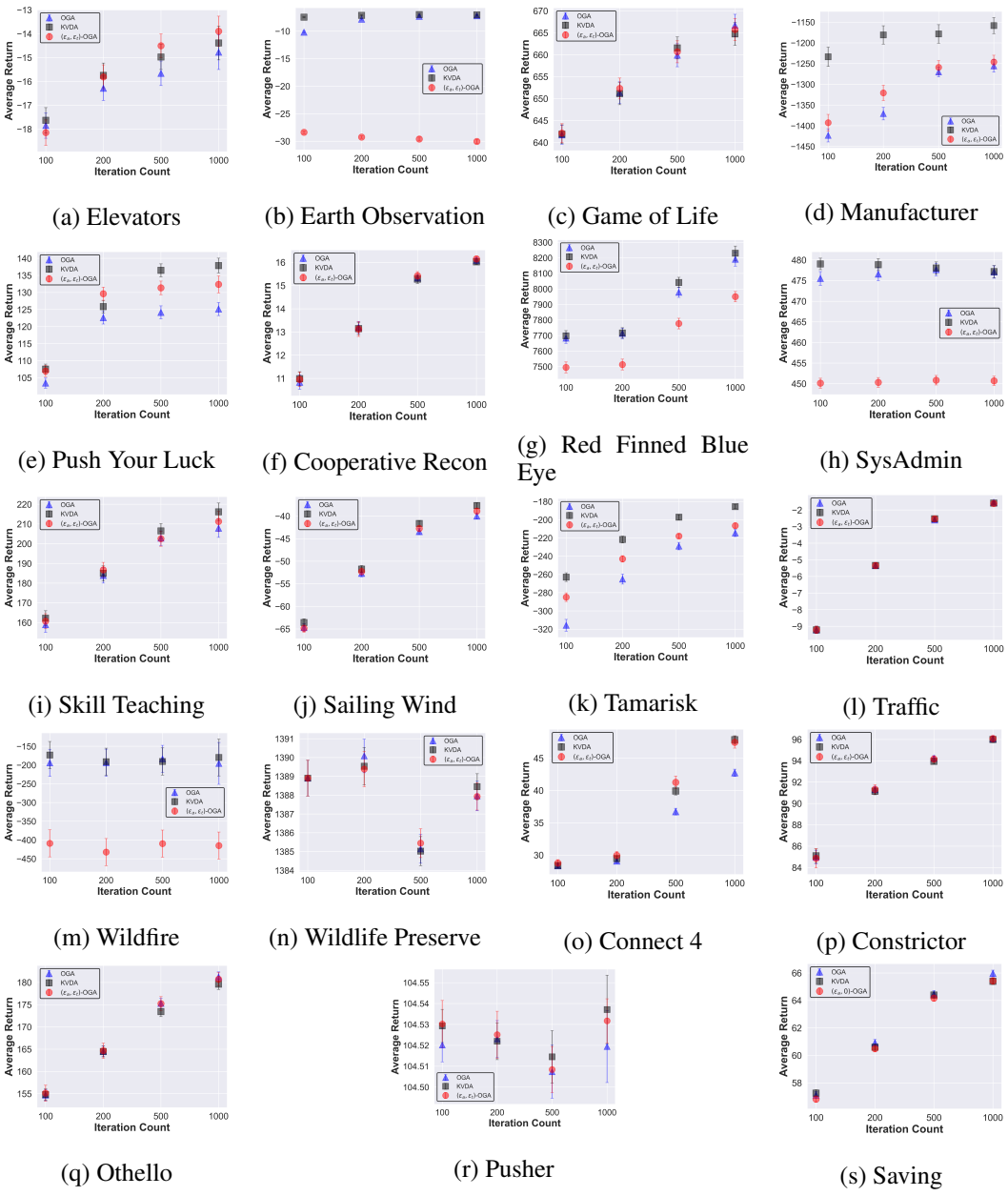
Table 6: Average returns (↑) using 500 MCTS iterations for KVDA-UCT, $(\infty, 0)$-OGA, $(0, 0)$-OGA (i.e. OGA-UCT), and the maximal performance of $(\varepsilon_{\text{a}}, 0)$-OGA when varying $\varepsilon_{\text{a}} > 0$. In the supplementary materials in Tab.4 the domain-specific ε_{a}-values are listed. All performances are the maximal performances obtained by varying the exploration constants $C \in \{0.5, 1, 2, 4, 8, 16\}$. Note that KVDA-UCT (our method) outperforms OGA-UCT in most environments. Furthermore, KVDA-UCT is either equal or performs better than parameter-optimized $(\varepsilon_{\text{a}}, 0)$-OGA even though KVDA-UCT introduces no extra parameter.

Domain	$(0, 0)$-OGA	$(\infty, 0)$-OGA	$(\varepsilon_{\text{a}}, 0)$-OGA	**KVDA-UCT** (ours)
Connect4	36.7 ± 0.5	$\mathbf{41.3 \pm 1.0}$	$\mathbf{41.3 \pm 1.0}$	39.9 ± 0.6
Constrictor	$\mathbf{94.2 \pm 0.2}$	92.8 ± 0.2	94.2 ± 0.3	94.0 ± 0.2
Othello	$\mathbf{175.3 \pm 1.1}$	156.8 ± 0.8	175.2 ± 1.6	173.4 ± 1.1
Pusher	104.5 ± 0.0	104.5 ± 0.0	104.5 ± 0.0	$\mathbf{104.5 \pm 0.0}$
d-Academic Advising	-42.1 ± 0.3	$\mathbf{-42.1 \pm 0.3}$	$\mathbf{-42.1 \pm 0.3}$	-42.3 ± 0.3
d-Cooperative Recon	15.3 ± 0.1	14.4 ± 0.1	$\mathbf{15.4 \pm 0.2}$	15.3 ± 0.2
d-Earth Observation	-7.28 ± 0.12	-29.6 ± 0.4	-29.6 ± 0.4	$\mathbf{-6.98 \pm 0.10}$
d-Elevators	-15.6 ± 0.3	$\mathbf{-14.6 \pm 0.3}$	$\mathbf{-14.6 \pm 0.3}$	-14.9 ± 0.3
d-Game of Life	659.9 ± 2.6	659.9 ± 2.6	660.7 ± 2.5	$\mathbf{661.6 \pm 2.5}$
d-Manufacturer	-1270.2 ± 11.7	-1646.1 ± 20.8	-1259.1 ± 16.5	$\mathbf{-1178.3 \pm 22.3}$
d-Push Your Luck	124.2 ± 1.9	67.4 ± 1.0	131.4 ± 2.0	$\mathbf{136.5 \pm 1.9}$
d-RedFinnedBlueEye	7978.4 ± 32.9	7738.1 ± 34.2	7777.4 ± 34.3	$\mathbf{8041.7 \pm 33.1}$
d-Sailing Wind	-43.4 ± 0.7	-42.8 ± 0.7	-42.8 ± 0.7	$\mathbf{-41.6 \pm 0.7}$
d-Saving	$\mathbf{64.5 \pm 0.2}$	61.7 ± 0.2	64.2 ± 0.2	64.4 ± 0.2
d-Skills Teaching	202.8 ± 3.7	201.0 ± 3.9	202.6 ± 3.9	$\mathbf{206.5 \pm 3.7}$
d-SysAdmin	477.7 ± 1.5	448.8 ± 1.2	450.8 ± 1.2	$\mathbf{478.1 \pm 1.5}$
d-Tamarisk	-228.9 ± 4.1	-217.9 ± 2.4	-217.9 ± 2.4	$\mathbf{-197.2 \pm 2.7}$
d-Traffic	-2.60 ± 0.10	$\mathbf{-2.53 \pm 0.09}$	$\mathbf{-2.53 \pm 0.09}$	$\mathbf{-2.53 \pm 0.09}$
d-Wildfire	$\mathbf{-183.6 \pm 36.1}$	-506.0 ± 36.2	-409.6 ± 36.1	-190.6 ± 36.3
d-Wildlife Preserve	1385.1 ± 0.8	1384.2 ± 0.8	$\mathbf{1385.5 \pm 0.8}$	1385.0 ± 0.8

Table 7: Average returns using 200 (↑) MCTS iterations for KVDA-UCT, $(\infty, 0)$-OGA, $(0, 0)$-OGA (i.e. OGA-UCT), and the maximal performance of $(\varepsilon_{\text{a}}, 0)$-OGA when varying $\varepsilon_{\text{a}} > 0$. In the supplementary materials in Tab.4 the domain-specific ε_{a}-values are listed. All performances are the maximal performances obtained by varying the exploration constants $C \in \{0.5, 1, 2, 4, 8, 16\}$. Note that KVDA-UCT (our method) outperforms OGA-UCT in most environments. Furthermore, KVDA-UCT is either equal or performs better than parameter-optimized $(\varepsilon_{\text{a}}, 0)$-OGA even though KVDA-UCT introduces no extra parameter.

Domain	$(0, 0)$-OGA	$(\infty, 0)$-OGA	$(\varepsilon_{\text{a}}, 0)$-OGA	**KVDA-UCT** (ours)
Connect4	29.1 ± 0.4	$\mathbf{29.9 \pm 0.6}$	$\mathbf{29.9 \pm 0.6}$	29.5 ± 0.4
Constrictor	91.3 ± 0.3	90.2 ± 0.2	$\mathbf{91.3 \pm 0.4}$	91.2 ± 0.3
Othello	164.5 ± 1.2	152.1 ± 0.8	$\mathbf{164.7 \pm 1.7}$	164.5 ± 1.2
Pusher	104.5 ± 0.0	$\mathbf{104.5 \pm 0.0}$	$\mathbf{104.5 \pm 0.0}$	104.5 ± 0.0
d-Academic Advising	-47.3 ± 0.4	$\mathbf{-47.2 \pm 0.4}$	$\mathbf{-47.2 \pm 0.4}$	-47.5 ± 0.4
d-Cooperative Recon	$\mathbf{13.2 \pm 0.3}$	12.7 ± 0.3	13.1 ± 0.3	13.1 ± 0.3
d-Earth Observation	-7.82 ± 0.14	-29.3 ± 0.2	-29.3 ± 0.2	$\mathbf{-7.11 \pm 0.11}$
d-Elevators	-16.2 ± 0.3	-16.1 ± 0.3	-16.1 ± 0.3	$\mathbf{-16.0 \pm 0.3}$
d-Game of Life	651.4 ± 2.5	$\mathbf{652.3 \pm 2.4}$	$\mathbf{652.3 \pm 2.4}$	651.1 ± 2.5
d-Manufacturer	-1370.7 ± 15.3	-1641.4 ± 21.2	-1320.7 ± 18.6	$\mathbf{-1180.8 \pm 22.2}$
d-Push Your Luck	122.6 ± 1.9	67.1 ± 0.9	$\mathbf{129.6 \pm 1.9}$	125.9 ± 1.8
d-RedFinnedBlueEye	7713.6 ± 33.9	7511.8 ± 34.7	7513.4 ± 34.9	$\mathbf{7716.9 \pm 33.7}$
d-Sailing Wind	-52.7 ± 0.8	-52.3 ± 0.8	-52.3 ± 0.8	$\mathbf{-51.7 \pm 0.8}$
d-Saving	$\mathbf{60.9 \pm 0.2}$	58.2 ± 0.2	60.5 ± 0.2	60.6 ± 0.2
d-Skills Teaching	183.9 ± 3.9	$\mathbf{186.7 \pm 3.8}$	$\mathbf{186.7 \pm 3.8}$	185.0 ± 3.8
d-SysAdmin	476.6 ± 1.6	450.2 ± 1.2	450.3 ± 1.2	$\mathbf{478.9 \pm 1.4}$
d-Tamarisk	-265.1 ± 5.5	-242.9 ± 3.5	-242.9 ± 3.5	$\mathbf{-221.7 \pm 3.8}$
d-Traffic	$\mathbf{-5.34 \pm 0.15}$	$\mathbf{-5.34 \pm 0.15}$	$\mathbf{-5.34 \pm 0.15}$	$\mathbf{-5.34 \pm 0.15}$
d-Wildfire	-193.8 ± 36.1	-476.5 ± 36.1	-432.2 ± 36.2	$\mathbf{-191.2 \pm 36.1}$
d-Wildlife Preserve	$\mathbf{1390.1 \pm 0.9}$	1389.4 ± 0.9	1389.4 ± 0.9	1389.5 ± 1.0

A.6 PERFORMANCE GRAPHS OF PARAMETER-OPTIMIZED KVDA-UCT



(a) Elevators (b) Earth Observation (c) Game of Life (d) Manufacturer

(e) Push Your Luck (f) Cooperative Recon (g) Red Finned Blue Eye (h) SysAdmin

(i) Skill Teaching (j) Sailing Wind (k) Tamarisk (l) Traffic

(m) Wildfire (n) Wildlife Preserve (o) Connect 4 (p) Constrictor

(q) Othello (r) Pusher (s) Saving

Figure 4: The performance plots in dependence of the iteration budget of parameter-optimized KVDA-UCT (our method), OGA-UCT, and $(\varepsilon_a, 0)$-OGA, $\varepsilon_a > 0$ on all considered environments. Our method KVDA-UCT is either always tightly within the confidence bounds of the top-competitor method or outperforms all competitors simultaneously.

A.7 MONTE CARLO TREE SEARCH

All here-presented abstraction algorithms rely on Monte Carlo Tree Search (MCTS) which we are going to describe now. Let M be a finite-horizon MDP. On a high level, MCTS repeatedly samples trajectories starting at some state $s_0 \in S$ where a decision has to be made until a stopping criterion is met. The final decision is then chosen as the action at s_0 with the highest average return. In contrast to a pure Monte Carlo search, MCTS improves subsequent trajectories by building a tree (or, in our

case, a directed acyclic graph) from a subset of the states encountered in the last iterations, which is then exploited. In contrast to pure Monte Carlo search, MCTS is guaranteed to converge to the optimal action.

An MCTS directed acyclic graph is made of two components. Firstly, the state nodes, that represent states and Q nodes that represent state action pairs. Each state node, saves only its children which are a set of Q nodes. Q nodes save both its children which are state nodes and the number of and the sum of the returns of all trajectories that were sampled starting at the Q node.

Initially, the MCTS search graph consists only of a single state node representing s_0. Until the iteration budget is exhausted, the following steps are repeated.

1. **Selection phase**: Starting at the root node, MCTS first selects a Q node according to the so-called *tree policy*, which may use the nodes' statistics, and then samples one of the Q node's successor states. If either a terminal state node, a state node with at least one non-visited action (partially expanded), or a new Q node successor state is sampled that is not represented by another node of the same layer, the selection phase ends.

   A commonly used tree policy (**and the one we used**) that is synonymously used with MCTS is Upper Confidence Trees (UCT) (Kocsis & Szepesvári, 2006), which selects an action that maximizes the Upper Confidence Bound (UCB) value. Let $s \in S$ and V_a, N_a with $a \in \mathbb{N}$ be the return sum and visits and of the Q nodes of the node representing s. The UCB value of any action a is then given by

$$\text{UCB}(a) = \underbrace{\frac{V_a}{N_a}}_{\text{Q term}} + \underbrace{\lambda \sqrt{\frac{\log\left(\sum_{a' \in \mathbb{A}(s)} N_{a'}\right)}{N_a}}}_{\text{Exploration term}}. \tag{10}$$

   The exploration term quantifies how much the Q term could be improved if this Q node was fully exploited and is controlled by the exploration constant $\lambda \in \mathbb{R} \cup \{\infty\}$. If one chose $\lambda = 0$, the UCT selection policy becomes the greedy policy and for $\lambda = \infty$, the selection policy becomes a uniform policy over the visits. In case of equality, some tiebreak rule has to be selected, which is typically a random tiebreak. From here, will use MCTS and UCT (MCTS with UCB selection formula) synonymously.

2. **Expansion**: Unless the selection phases ended in a terminal state node, the search directed acyclic graph is expanded by a single node. In case the selection phase ended in a partially expanded state node, then one unexpanded action is selected (e.g. randomly, or according to some rule), the corresponding Q node is created and added as a child and one successor state of that Q node is sampled and added as a child to the new Q node. If the selection phase ended because a new successor of a Q node was sampled, then a state node representing this new state is added as a child to that Q node.

3. **Rollout/Simulation phase**: Starting at the state $s_{rollout}$ of the newly added state node of the expansion phase (or at a terminal state node reached by the selection phase), actions according to the *rollout policy* are repeatedly selected and applied to $s_{rollout}$ until a terminal state is reached. All states encountered during this phase are not added to the search graph.

4. **Backpropagation**: In this phase, the statistics of all Q nodes that were part of the last sampled trajectory that corresponds to a path in the search graph are updated by incrementing their visit count and adding the trajectory's return (of the trajectory starting at the respective Q node) to their return sum statistic.

A.8 Problem descriptions

In the following, we provide a brief description of each domain/environment that was used in this paper. Some of these environments can be parametrized (e.g., choosing a concrete map for Pusher). The concrete parameter settings can be found in the *ExperimentConfigs* folder in our publicly available GitHub repository (Authors, 2025). In the following, for the reader's convenience, we re-introduced the relevant environment descriptions from the survey paper (Schmöcker

& Dockhorn, 2025) as well as added new ones for those not contained in the survey. For a detailed description of these environments, we refer to our implementation available at `https://anonymous.4open.science/r/KVDA_UCT-6E51/README.md`.

- **Academic Advising**: The Academic Advising domain, introduced by Guerin et al. (2012) and a modified version featured in the IPPC 2014 competition (Grzes et al., 2014), models a student navigating their academic progress. The agent represents a student aiming to successfully complete a subset of academic courses. Formally, the state space is defined as $\{P, \text{NP}, \text{NT}\}^n$ where n is the total number of courses, P means the course is passed, NP means the course was taken at least once but not yet passed, and NT means not taken. At each decision step, the agent selects a course to attempt. The outcome of the course depends probabilistically on the completion status and grades of its prerequisites. Rewards are assigned based on whether the student takes the target courses and the level of success (grade) achieved in them.

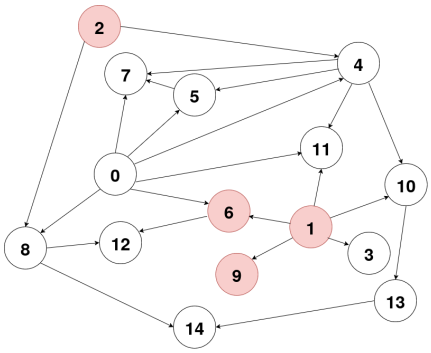


Figure 5: The requirements graph of the Academic Advising instance used for the experiments. The courses marked in red are the target classes.

- **Constrictor**: Constrictor is played on an n times n grid. Players take turns moving to any of the neighbouring (4-neighbourhood) grid cells that neither moves the player out of bounds nor hits any cell that has already been visited by any of the two players. The game ends when one player has nowhere left to move. The heuristics function used for player i is the number of grid cells that i could reach before player $i-1$. If player i wins, the value 100 is added to the heuristic.

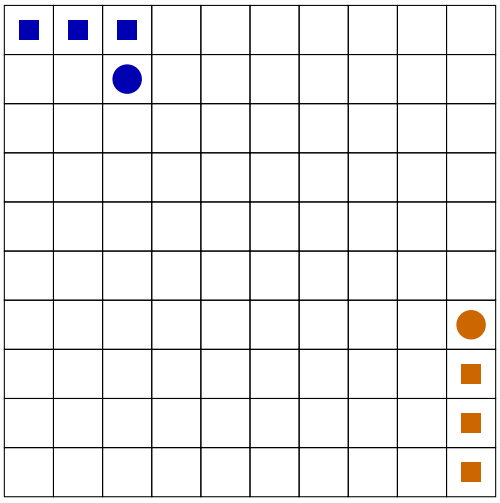


Figure 6: A visualization of a game state that could have occurred after 6 steps of Constrictor. The circles indicate the players' heads and the squares their bodies.

- **Connect4**: Connect 4 is played on grid with 7 columns and 6 rows. Each turn, one player places a stone of its colour in one of the columns that is not yet filled with stones. The stone occupies the first cell in the chosen column that is not yet occupied. As the heuristic V^{h} from the perspective of player one is given by

$$n_2 + 5n_3 + 25n_4 \tag{11}$$

where n_i is the total number of i stones that are in one row/column/diagonal divided by i but which are not part of a row/column/diagonal of size $i + 1$. Fig. 7 visualizes a concrete Connect 4 game state.

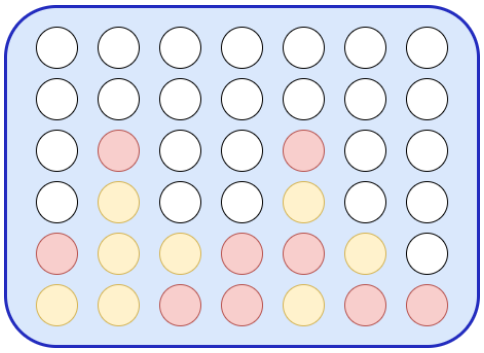


Figure 7: A visualization of a Connect 4 board state that could have occurred after 17 moves of playing.

- **Cooperative Recon**: This domain models a robot tasked with discovering signs of life on a foreign planet. The robot operates on a two-dimensional grid populated with various objects of interest and a central base. When the robot reaches an object of interest, it can perform surveys to detect the presence of water and, subsequently, life. The probability of detecting life increases if water is first identified. If life is successfully detected, the robot can then photograph the object - this is the only action that yields a reward. Each use of a detector carries a risk of failure, which may render the detector unusable or reduce its reliability. Detectors can be repaired, but only at the base location.

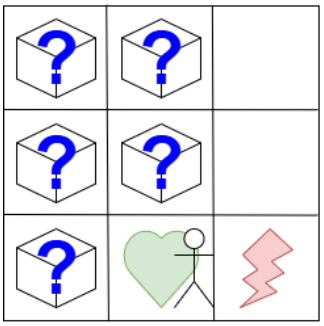


Figure 8: A visualization of the Cooperative Recon instance used for the experiments. Question mark cells indicate objects of interest, the lightning represents a hazard, the green heart the base for repairs and the stick figure is the initial position of the agent.

- **Crossing Traffic**: This domain, also featured in IPPC 2014 (Grzes et al., 2014), presents a grid-based navigation challenge where the agent must traverse multiple lanes of moving traffic. Obstacles travel along the x-axis from right to left and are randomly generated at the right edge of the grid. A collision with an obstacle immobilizes the agent, making the episode unsolvable. The objective is to cross all traffic lanes while avoiding collisions, ideally following the shortest possible path.

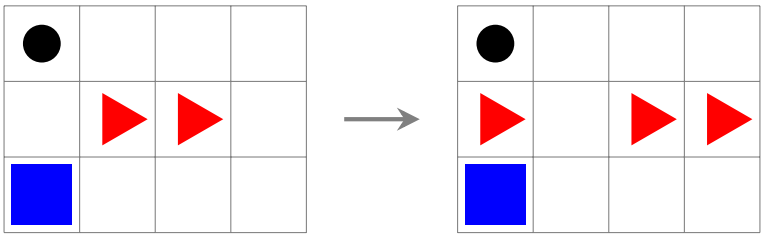
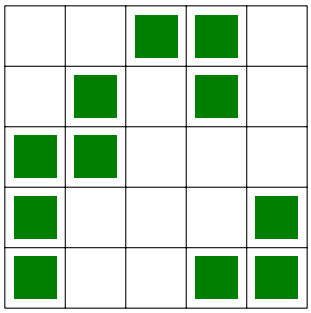


Figure 9: A visualization of a sample state transition for the Crossing Traffic instance used in the experiments assuming that the player (black dot) chose to idle. Red triangles indicate obstacles and the blue square is the goal position.

- **Earth Observation**: This problem, proposed by Hertle et al. (2014), models a satellite orbiting Earth while performing photographic observations. Each state corresponds to a position on a two-dimensional grid, where the satellite's longitudinal location and the latitude at which its camera is aimed are represented. Additionally, certain designated cells have associated weather levels that influence observation quality. Weather conditions change stochastically at each time step, independent of the agent's actions. The agent can choose to idle, take a photograph of the current target cell, or adjust the camera's focus by incrementing or decrementing the y-position (latitude). A reward is granted when a designated cell is photographed, with the reward magnitude depending on the prevailing weather in that cell.
- **Elevators**: This domain, featured in IPPC 2014 (Grzes et al., 2014), involves managing a set of elevators tasked with transporting passengers to their requested destinations—either the top or bottom floor. Passengers arrive randomly on different floors, each specifying their desired direction of travel. At each time step, the agent controls the elevators by issuing commands to open doors and set travel direction, close doors, or move the elevator in the indicated direction.
- **Game of Life**: John Conway's original Game of Life (Gardner, 1970), a deterministic cellular automaton, was adapted into a stochastic Markov Decision Process (MDP) as a benchmark domain for the International Probabilistic Planning Competition (Sanner & Yoon, 2011). This adaptation introduces stochasticity into the state transitions, defines the reward as the number of alive cells at each step, and grants the agent control over one cell per round, ensuring its survival into the next generation. States are represented as elements of $\{0, 1\}^{n \times n}$, where each grid cell indicates whether it is alive (1) or dead (0). Like other authors (Schmöcker et al., 2025), we reduced the action space complexity by allowing only to select living cells in the current step that will be protected from dying.



(a) A visualization of the initial state used for the 5x5 Game of Life instance used for the experiments. Green cells indicate living cells.

- **Manufacturer**: In this domain, the agent is responsible for managing a manufacturing company with the objective of selling goods to customers. To do so, the agent must first produce the goods, which may involve constructing factories and procuring the necessary input materials. A key challenge lies in the stochastic fluctuations of goods' market prices.

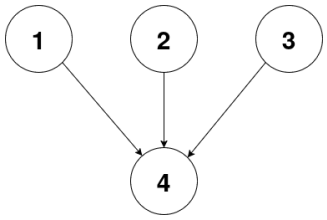


Figure 11: Good production dependency graph of the Manufacturer instance considered for the experiments. An arrow from node i to node j indicates that product j requires product i for production.

- **Numbers Race**: In Numbers Race, players take turns choosing an integer between 1 and $n \in \mathbb{N}$. The goal is to choose a number $m \leq n$ such that the sum of all previous numbers is equal to some goal number $g \in \mathbb{N}$. If this sum exceeds g, then the player that overshot, loses.

- **Othello**: Othello is played on an 8x8 board (in our case a 6x6 board) where players take turns placing a stone of their colour on an empty cell. Once placed, all opponent's stones are flipped that are contained in a vertical, horizontal or diagonal line that starts at the placed stone and that ends at the first stone of the same colour as the placed stone going in the line's direction. When neither player has an available move, the player with the most stones wins. The heuristic function for any state for player i is given by a weighted sum of all the occupied grid cells. The weight w of cell (x, y) is given by $10/(1 + d)$ where d is the distance to the closest corner cell. The weight w is positive for any cells occupied by stones of i and negative for one occupied by $1 - i$. In the round player i wins, 100 is added to the heuristics value and -100 if $1 - i$ won.

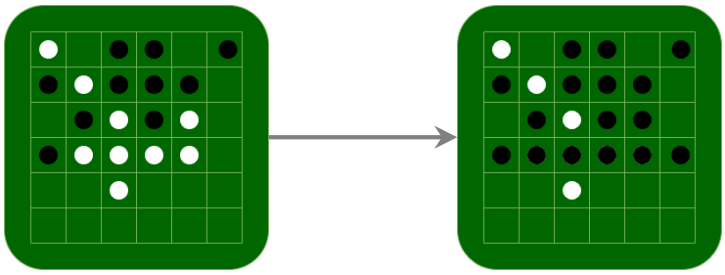


Figure 12: A of a state transition for the 6x6 version of Othello. In this example, black placed a stone, which caused five white stones to switch their color to black.

- **Push Your Luck**: In Push Your Luck the agent has to decide which of n, m-sided, not-necessarily fair dice or cash-out. If cashed-out, the agent receives a reward dependent on all dice faces that are marked. Faces are marked if they have been rolled (each face is shared by all n dice). However, if the agent rolls an already marked face, or rolls two unmarked faces at the same time, all markings are removed.

- **Pusher**: Pusher is a game mode from the Stratega framework (Dockhorn et al., 2020). Pusher is played on a 2-dimensional grid. Each player controls an initially equal number of units. The goal is to eliminate all enemy units, which can be achieved by pushing them into holes which are spread around the map. Per step, a player may move one unit to an empty neighbouring tile (4-neighbourhood) and optionally push an enemy unit in the movement direction by 1 tile if the enemy unit is located at a neighbouring tile. We used the heuristic for player i that is equal to the difference of alive units of i and $i - 1$ in addition to the Manhatten distance between player i and $i - 1$'s units' center of mass. In the turn that player i wins, 100 is added to the heuristic value.

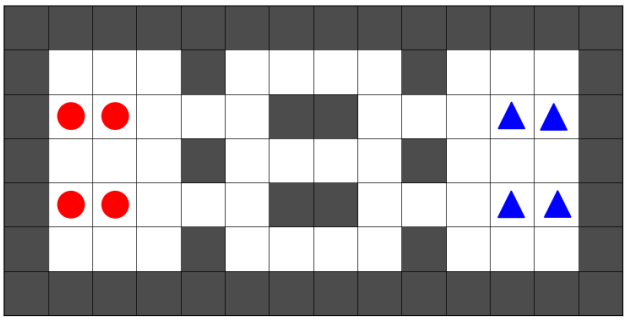
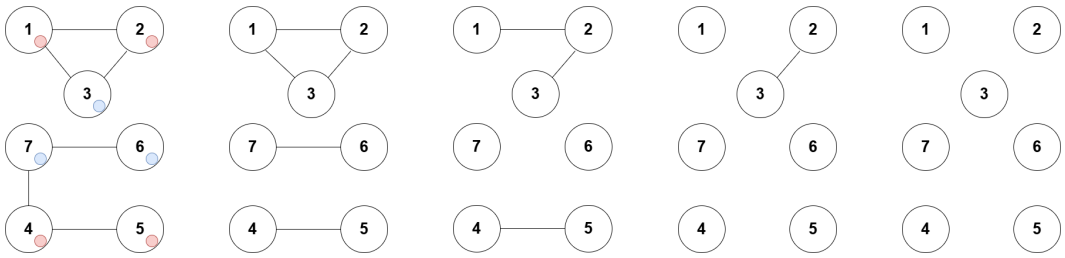


Figure 13: A visualization of the initial state of the Pusher instance used for the experiments. The red circles are units of player 1 and the blue triangles are those of player 2.

- **Red Finned Blue Eye**: In this environment, the agent is tasked with preserving and restoring the Red Finned Blue Eye (RFBE) fish population which is being threatened by an invasive species of Gambusian fish. The ecosystem is being modelled as springs that are connected in a directed graph, however, the connections' accessibility is dependent on the current global water level, which changes stochastically. Gambusian spreads aggressively between connected springs.



(a) The spring connections at the highest water level 7.

(b) Spring connections at water levels 5 and 6.

(c) Spring connections at water level 4.

(d) Spring connections at water level 3.

(e) Spring connections at water levels 1 and 2.

Figure 14: A visualization of the Red Finned Blue instance used for the experiments, showing how the springs are connected as water level varies from 1 (least) to 7 (most). Furthermore, the leftmost subfigure (a) also shows the initial populations of the Gambusian (red) and the Red Finned Blue Eye (blue).

- **Sailing Wind**: Originally proposed by Robert Vanderbei (Vanderbei, 1996), the Sailing Wind domain challenges the agent to navigate a ship from the starting position $(1, 1)$ to the goal at (n, n) on an $n \times n$ grid while minimizing total cost. The formulation of transition and reward functions varies across the literature. Some versions restrict the agent to only two actions - down and right (Jiang et al., 2014)—while others allow up to seven possible moves, excluding the cell directly against the current wind direction (Anand et al., 2015). Action costs depend on the wind direction, which changes stochastically at each time step, independent of the agent's actions.

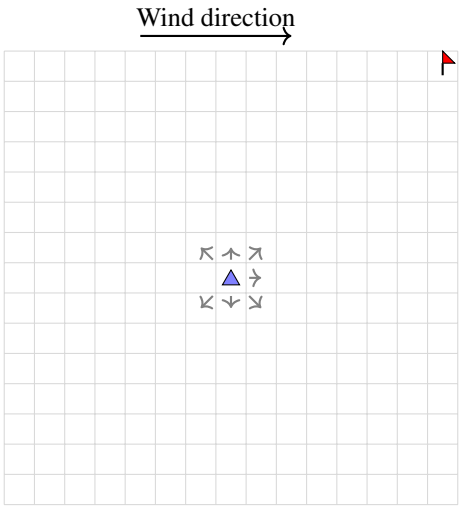


Figure 15: A visualization of the Sailing Wind instance used for the experiments. The red flag indicates the goal position, the blue triangle is the agent's position, and the grey arrows indicate the direction it can move in in the current step.

- **Skill Teaching** In Skill Teaching, the agent takes the role of a tutor that is tasked with increasing the proficiency level of a student at various skills. The student can have one of three proficiency levels at each skill: Low, medium, and high. The skills from a prerequisite graph, giving the student higher chances of learning a new skill the higher the prerequisites' levels of proficiency. Difficulty arises from the proficiency levels decaying if the corresponding skill wasn't practised. This decay is deterministic for skills at medium proficiency and stochastic for those at high proficiency.

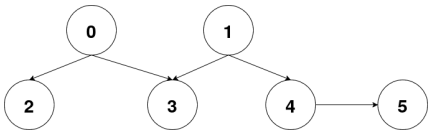


Figure 16: The prerequisites graph for the Skill Teaching instance used for the experiments.

- **Saving**: Saving is introduced by Hostetler et al. (2015), where the agent aims to maximize accumulated wealth over time. At each step, the agent can choose one of three actions: Invest, Borrow, or Save. Borrow provides an immediate reward of 2 but imposes a penalty of -3 after n time steps. Once this action is taken, it cannot be repeated until the delayed penalty is applied. Save yields an immediate reward of 1 with no further consequences. Invest offers no immediate reward but enables the agent to take the Sell action within the next m time steps. The agent cannot invest again until either the Sell action is executed or m steps have elapsed. If Sell is chosen, then the agent receives a reward equal to the current price level that changes stochastically and independently of the agent's actions.

- **SysAdmin**: Proposed by Guestrin et al. (2003), the SysAdmin domain models the management of a computer network, represented as a graph with $n \in \mathbb{N}$ nodes, where each node corresponds to a machine. The state space is $\{0, 1\}^n$, indicating the operational status of each machine (1 for working, 0 for failed). The action space is $\{1, \ldots, n, \text{IDLE}\}$, where each action corresponds to rebooting a specific machine or idling. At each time step, the reward depends on the number of machines that are currently operational. Rebooting a machine increases the likelihood that it will be functioning in the next step. However, machine failures occur stochastically, with an elevated probability when neighboring machines are also down.

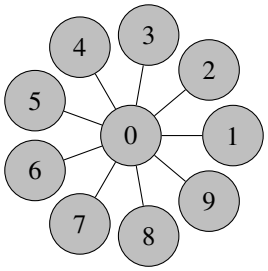


Figure 17: Visualization of the SysAdmin topology used for the experiments.

- **Tamarisk**: Tamarisk, another domain from IPPC 2014 (Grzes et al., 2014), models the spread of an invasive plant species within a river system. The environment is represented as a linear chain of river segments, or reaches, each containing several slots. Each slot can be unoccupied, occupied by a native plant, or occupied by the invasive Tamarisk plant. Both native and Tamarisk plants spread stochastically to adjacent reaches, with a higher probability of spreading downstream. At each time step, the agent selects one action for one reach: do nothing, eradicate Tamarisk, or restore native vegetation. The chosen action is applied uniformly to all slots within that reach. All actions - except doing nothing - have a chance of failure, occurring randomly.
- **Tic Tac Toe**: Tic Tac Toe is played on a grid of width and height 3. Each player is assigned a colour and places one stone of its colour in one of the empty grid cells. The first player to create a vertical, horizontal, or diagonal row of three same-coloured stones wins.

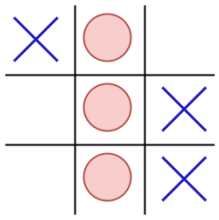


Figure 18: Visualization of a Tic Tac Toe gamestate in which the player with the red circles won because a vertical row was completed.

- **Traffic**: In this environment, the agent is tasked with simultaneously controlling a number of traffic lights with the goal of minimizing traffic jams. This traffic is modelled as a directed graph, however, some edges are only available depending on the state of a traffic light. Each vertex may either contain a car or not.

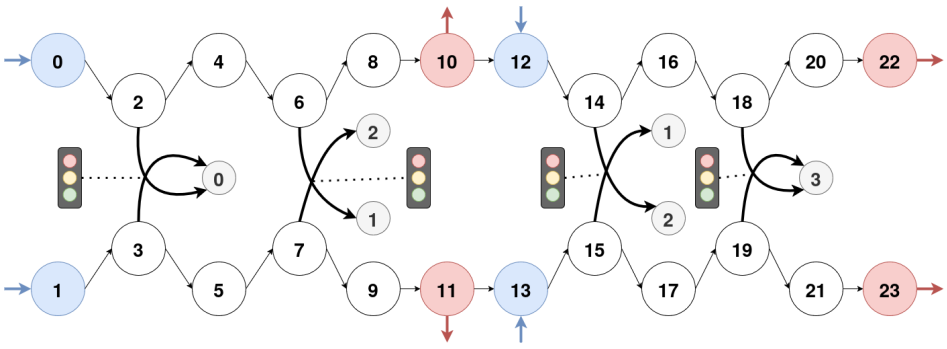


Figure 19: The traffic graph used in the experiments. Blue cells are cells that receive a random car inflow, redly marked cells are cells where the traffic flow exits and the bold edges are intersection edges. Only one of crossing bold edges can ever be active at the same time.

- **Wildfire**: Wildfire models the spread of a fire on a grid. Each grid cell is either untouched, burning, or out-of-fuel meaning that no new fire can ignite at this cell. If a cell is untouched it can at each time step randomly ignite with the probability increasing exponentially in the number of neighbouring burning cells. The neighbourhood is defined on an instance level with most instances choosing the 8-neighbourhood and manually cutting a handful of neighbourhood connections between individual cells. The agent is tasked with controlling the spread of the fire.

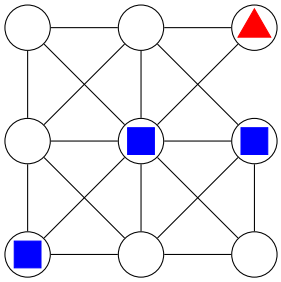
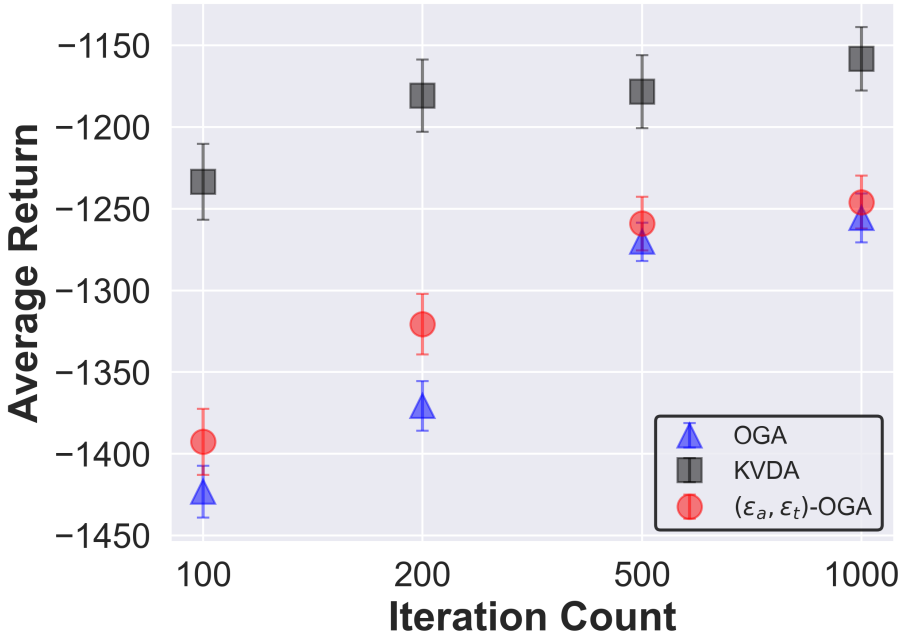
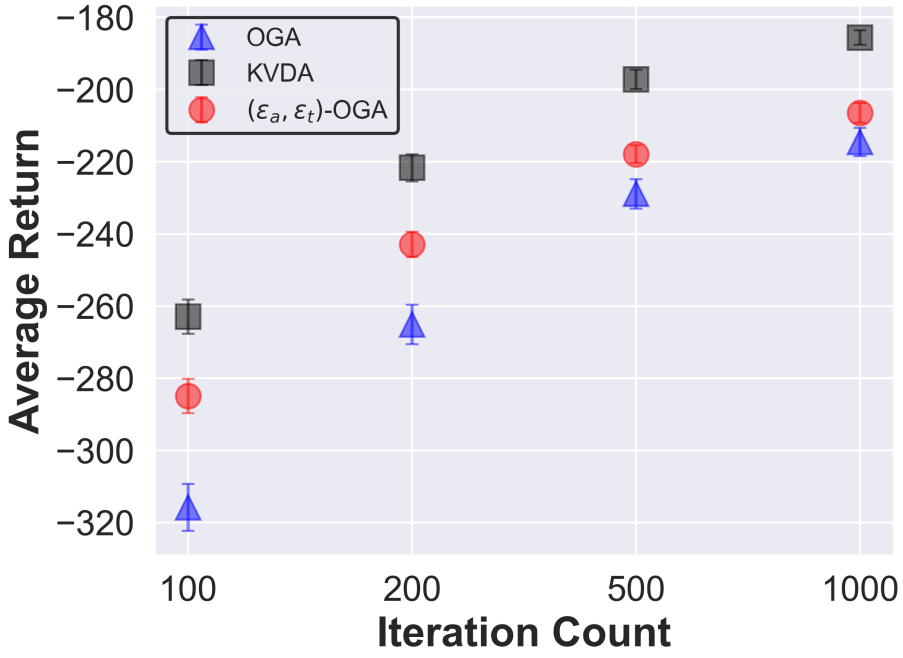


Figure 20: A visualization of the initial state of the Wildfire instance used for the experiments. The red triangle indicates fire and the blue squares indicate so-called targets which have to be protected.

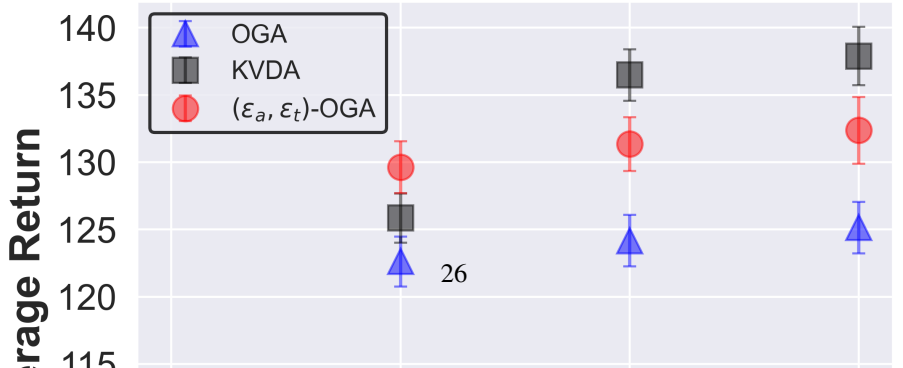
- **Wildlife Preserve**: In Wildlife preserve the agent manages rangers to defend areas from poachers. If a ranger was sent to defend an area a poacher decided to attack, the poacher is caught and can not attack an area in the next step. Each poacher has different area preferences and remembers whether how often which area was defended in the last couple of steps.



(a) d-Manufacturer



(b) d-Tamarisk



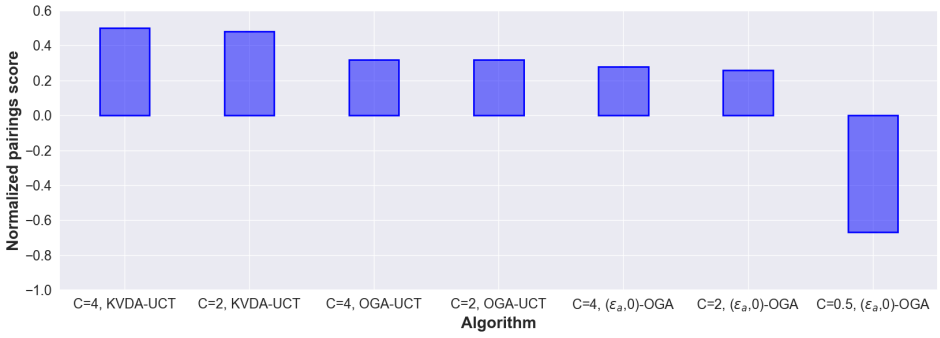


Figure 3: The normalized pairings score (↑) for the top 6 and the worst agent on deterministic environments. The agents considered were KVDA-UCT (our method) which performs best overall, OGA-UCT (Anand et al., 2016), and $(\varepsilon_{\mathrm{a}}, 0)$-OGA (Schmöcker et al., 2025), $\varepsilon_{\mathrm{a}} > 0$ with the exploration constants $C \in \{0.5, 1, 2, 4, 8, 16\}$ and budgets of $\{100, 200, 500, 1000\}$ iterations. The top two spots are occupied by our method KVDA-UCT, with the best overall performing algorithm being KVDA-UCT with $C = 4$.