AUPO - Abstracted Until Proven Otherwise: A Reward Distribution Based Abstraction Algorithm

Robin Schmöcker[1], **Alexander Dockhorn**[2], **Bodo Rosenhahn**[1]
[1]Institute for Information Processing, Leibniz University Hannover, Hannover, Germany
[2]SDU Metaverse Lab, University of Southern Denmark, Odense, Denmark
schmoecker@tnt.uni-hannover.de, rosenhahn@tnt.uni-hannover.de, adoc@mmmi.sdu.dk

Abstract

We introduce a novel, drop-in modification to Monte Carlo Tree Search's (MCTS) decision policy that we call AUPO. Comparisons based on a range of IPPC benchmark problems show that AUPO outperforms vanilla MCTS in domains with dense rewards and value-equivalent sibling actions under finite iteration budgets. AUPO is an automatic action abstraction algorithm that solely relies on reward distribution statistics acquired during the MCTS. Thus, unlike other automatic abstraction algorithms, AUPO requires neither access to transition probabilities nor does AUPO require a directed acyclic search graph to build its abstraction, allowing AUPO to detect symmetric actions that state-of-the-art frameworks like ASAP struggle with when the resulting symmetric states are far apart in state space. Furthermore, as AUPO only affects the decision policy, it is not mutually exclusive with other abstraction techniques that only affect the tree search.

Introduction

A plethora of important problems can be viewed as sequential decision-making tasks such as autonomous driving (Liu et al. 2021), energy grid optimization (Sogabe et al. 2018), financial portfolio management (Birge 2007), or playing video games (Silver et al. 2016). Though arguably state-of-the-art on such decision-making tasks is achieved using machine learning (ML) as demonstrated by DeepMind with their AlphaGo agent for Go (Silver et al. 2016) or OpenAI Five for Dota 2 (Berner et al. 2019), there is still a demand for general domain-knowledge independent, on-the-go-applicable planning methods, properties which ML-based approaches usually lack but which are satisfied by Monte Carlo Tree Search (Browne et al. 2012) (MCTS), the method of interest for this paper. For example, Game Studios rarely implement ML agents as they have to be expensively retrained whenever the game and its rules and updated. Though not within the scope of this paper, improvements to MCTS might also potentially translate to ML-based methods that use MCTS as their underlying search.

One family of approaches to improve the performance of MCTS is using abstractions that usually group similarly behaving nodes or actions of the search tree. State-of-the-art abstraction tree searches such as OGA-UCT (Anand et al. 2016) all rely on the reward function being deterministic, having full access to the transition probability of any sampled state-action pair, and on the search graph being a directed acyclic graph (DAG). While these methods could in principle still be applied if the first two conditions aren't met (i.e. by, approximating the reward and transition probabilities), they fundamentally rely on doing search on a DAG, which requires being able to check state equalities which is not always guaranteed (e.g., in memory-constrained settings where states may only be represented as action, stochastic-outcome sequences, in partially observable domains, in continuous-state settings, or in blackbox simulation settings). Until now, no domain-independent, non-learning-based MCTS abstraction algorithms for discrete, fully-observable settings exist that have no additional constraints than MCTS. This is a gap that this paper closes. Concretely, we introduce **A**bstracted **U**ntil **P**roven **O**therwise (AUPO), the first MCTS-based abstraction algorithm that can outperform MCTS in a discrete, fully-observable, non-learning-based setting whilst requiring neither access to transition probabilities nor a DAG, nor a deterministic reward setting. Necessary conditions for AUPO to yield performance gains are that the environment contains dense rewards and states with multiple actions that have similar values.

AUPO is a heuristic that only affects the decision policy and can thus even be combined and enhanced with other abstraction algorithms during the tree policy. Furthermore, in practice, AUPO can detect symmetric actions that the ASAP (Anand et al. 2015) framework cannot when the resulting symmetric states are far apart in state space, as ASAP needs the search graph to converge. As only the decision policy is affected, AUPO's runtime overhead vanishes with an increase in the iteration count (see appendix Tab.3).

The key idea of AUPO is to consider all actions at the root node initially as grouped, only splitting groups if the layerwise reward distributions, which were tracked during the MCTS search phase, differ significantly. To our knowledge, AUPO is the first abstraction algorithm to build abstractions based solely on reward distribution statistics.

The paper is structured as follows. First, in **Related**

Work, we give an overview of domain knowledge independent abstraction tree searches. Next, in **Foundations** we formalize our problem setting, and lay the theoretical groundwork for understanding AUPO. This is followed by the section **Method** where we formalize AUPO and **Experiments** where we experimentally verify AUPO and discuss the experimental results. Lastly, in **Limitations and Future Work** we summarise our findings and show avenues for future work.

Related Work

The literature on abstraction-using planners is vast and ranges from abstractions for strategy games (Moraes and Lelis 2018; Xu, Dockhorn, and Perez-Liebana 2023), card games such as Poker (Billings et al. 2003) to board games such as Go (Childs, Brodeur, and Kocsis 2008) or even hospital scheduling planners (Friha, Berry, and Choueiry 1997). Aside from such domain-specific abstractions, general abstraction methods are developed for continuous and/or partially observable domains (Hoerger et al. 2024) or learning-based abstractions such as learning and planning on abstract models (Ozair et al. 2021; Kwak et al. 2024; Chitnis et al. 2020), or building abstractions that rely on learned functions (e.g. a value function) (Fu et al. 2023). A comprehensive survey that focuses on non-learning-based abstraction techniques is provided by Schmöcker et al. (Schmöcker and Dockhorn 2025).

The literature on non-learning-based, fully domain-independent abstraction (the scope of this paper) techniques is small. For MCTS-based planners research has heavily focused on grouping states or state-action pairs of the current MCTS search graph such that their aggregate statistics can be used the enhance the UCB formula. This is achieved by bootstrapping of state-action pair with common successors and inferring which other states or state-action pairs must consequently be value-equivalent (Jiang, Singh, and Lewis 2014; Anand et al. 2015, 2016). In contrast to AUPO, however, these methods can only be applied if one has access to an equality check operator and if the state space graph is not a tree.

All of the above-mentioned techniques can be thought of as pessimistic in that they only abstract actions or states when precise conditions are met. However, in environments where equivalences are the norm and not the exception, optimistic approaches, such as AUPO, can thrive. For example, PARSS (Hostetler, Fern, and Dietterich 2015) modifies Sparse Sampling by initially grouping all successors of each state-action pair. These groups are refined by repeatedly splitting them in half as the search progresses. Like PARSS, AUPO can also be viewed as a refining and optimistic abstraction algorithm, but whereas PARSS randomly refines its abstractions when it does not have access to additional state information, AUPO does so using statistical evidence. Like PARSS and AUPO, the method of fully abandoning an abstraction mid-search (Xu, Dockhorn, and Perez-Liebana 2023) or dynamically dropping abstractions on a per-node level (Schmöcker, Kampmann, and Dockhorn 2025) can also be seen as an abstraction refinement technique. Another refining approach that is not fully domain-independent is to group states based on a domain-specific distance function, and the maximal grouping distance shrinks as the search progresses (Sokota et al. 2021).

Foundations

We use finite Markov Decision Processes (MDP) (Sutton and Barto 2018) to formalize the sequential decision-making tasks AUPO attempts to solve.

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

- $S \neq \emptyset$ is the finite set of states, $T \subseteq S$ is the (possibly empty) set of terminal states, and $\mu_0 \in \Delta(S)$ is the probability distribution for the initial state where $\Delta(X)$ denotes the probability simplex of a finite, non-empty set X.
- $\mathbb{A}\colon S \mapsto A$ maps each state s to the available actions $\emptyset \neq \mathbb{A}(s) \subseteq A$ at state s where $|A| < \infty$. Let $P \coloneqq \{(s,a) \mid s \in S, a \in \mathbb{A}(s)\}$ be the set of admissible state-action pairs.
- $\mathbb{P}\colon P \mapsto \Delta(S)$ is the stochastic transition function where we use $\mathbb{P}(s' \mid s, a)$ to denote the probability of transitioning from $s \in S$ to $s' \in S$ after taking action $a \in \mathbb{A}(s)$ in s.
- R is the reward function that maps from P to a real-valued random variable.

Starting in $s_0 \sim \mu_0$, an MDP progresses from state s_t to s_{t+1} by first selecting an action $a_t \sim \pi(s_t)$ and then sampling $s_{t+1} \sim \mathbb{P}(\cdot|s_t, a_t)$ where π is any agent for M. An agent $\pi\colon S \mapsto \Delta(A)$ for an MDP M is a mapping from states to action distributions with $\pi(s)(a) = 0$ for any $a \notin \mathbb{A}(s)$. Crucially, an agent's output depends only on a single state. At each transition M samples the reward $r_t \sim R(s_t, a_t)$.

In this paper, we consider only the finite horizon setting where the game ends after at most $h \in \mathbb{N}$ steps or earlier when a terminal state is reached. We call h the horizon and any state-action-reward sequence $(s_0, a_0, r_0), \ldots, (s_n, a_n, r_n), s_{n+1}$ that can be reached a trajectory. If additionally $n + 1 = h$ or $s_{n+1} \in T$ we call this trajectory an episode.

The goal of any agent is to maximize its expected return. The return of an episode $\tau = (s_0, a_0, r_0), \ldots, (s_n, a_n, r_n), s_{n+1}$ is defined as the (possibly discounted) sum of rewards, i.e. $R(\tau) \coloneqq \sum_{i=0}^{n} r_i \cdot \gamma^i$ where $0 < \gamma \leq 1$ is the discount factor. For any given state s, action $a \in \mathbb{A}(s)$, and maximum remaining steps $k \leq h$ we call $Q^*(s,a,k)$ the Q-value of (s,a,k) and $V^*(s,k)$ the state value of s (given k remaining steps) which are defined as

$$Q^*(s,a,k) \coloneqq \max_{\pi} \mathbb{E}_{\tau \sim \tau(\pi,s,a,k)}[R(\tau)], \tag{1}$$

$$V^*(s,k) \coloneqq \max_{a \in \mathbb{A}(s)} Q^*(s,a,k) \tag{2}$$

where $\tau(\pi, s, a, k)$ denotes the trajectory distribution of an agent π induced by starting at state s, directly applying a and then playing according to π for at most $k-1$ steps or until a terminal state is reached. We write $Q^*(s,a) \coloneqq Q^*(s,a,h)$ and $V^*(s) \coloneqq V^*(s,h)$ and $V^* \coloneqq \mathbb{E}_{s_0 \sim \mu_0}[V^*(s_0)]$.

Our AUPO method will heavily rely on and be compared to MCTS. For a detailed MCTS description, see appendix section Monte Carlo Tree Search. The appendix together with the code can be found in this paper's GitHub repository (Schmöcker, Dockhorn, and Rosenhahn 2025b). The MCTS version we employ here, chooses the root action with the highest Q value as the decision policy and uses the UCB tree policy. In the appendix in Fig. 8, we compared this decision policy against choosing the most visited action, the latter of which was inferior in most environments which is why we decided to use the greedy Q decision policy.

Method

In this section, AUPO will be introduced. The pseudocode is given in Alg. 1. The main idea of AUPO is to first find action abstractions at the root node during the decision policy by comparing the reward distributions at depths $1, \dots, D$ of the game tree. Initially, AUPO assumes all actions to be grouped, however, if the reward distributions of two actions differ significantly at any depth, the two actions are separated. Then, these found groupings are used to obtain lower-variance Q-value estimates for each root action by using the aggregate statistics of their corresponding group. A more detailed statistical analysis of this is found in the appendix in The benefit of abstractions to the decision policy.

By construction the abstraction built by AUPO will be sound i.e. only actions with the same Q^* will be grouped in the iteration limit. This as well as a characterization of the convergence speed of AUPO is formalized and proven in the appendix in the sections AUPO soundness and AUPO convergence speed. This section is structured as follows. Next, it will be described how AUPO groups the root actions after MCTS has been run based on the gathered MCTS statistics. Then, it will be described how these groupings/abstractions are used to enhance the decision policy.

Building the AUPO abstraction: Let us assume we are in a state $s \in S$ with actions $a_1, \dots, a_n$. After running standard MCTS for m iterations, we have sampled m trajectories where we denote the trajectories that started with action a_j by $\tau_{i,j} = (a_{w_1}, r_1, s_1), (a_{w_2}, r_2, s_2), \dots, (a_{w_{D_{i,j}}}, r_{D_{i,j}}, (s_{D_{i,j}}))$, $a_{w_1} = a_j,\ 1 \leq i \leq m_j, m_1 + \ldots + m_n = m$. Consider the reward sequence R_{d,a_j} obtained at depth d after playing action a_j at the root node i.e.

$$((R_{d,a_j})_i)_{1 \leq i \leq m_j} \coloneqq r_d \text{ with } (a_{w_d}, r_d, s_d) = (\tau_{i,j})_d \quad (3)$$

where we define $r_d \coloneqq 0$ in case $D_{i,j} < d < D$. Though this is a heuristic assumption, we assume that all R_{d,a_j} are samples from a stationary distribution $\mathcal{R}_{d,a_j}$ (this assumption would only hold if we performed a pure Monte Carlo search).

Next, we compute the empirical mean and standard deviation for all $\mathcal{R}_{d,j}$, $d \leq D$, along with their confidence intervals for a fixed confidence level $q \in [0, 1]$. Any pair of actions a_j, a_k has $2 \cdot D$ reward distributions associated with them which are $\mathcal{R}_{1,a_j}, \dots, \mathcal{R}_{D,a_j}$ for a_j and $\mathcal{R}_{1,a_k}, \dots, \mathcal{R}_{D,a_k}$ for a_k.

AUPO then groups a_j, a_k if and only if all confidence intervals (both the mean and std intervals) up to depth $d \leq D$ of the pairs $(\mathcal{R}_{d,a_j}, \mathcal{R}_{d,a_k})$ overlap. AUPO builds the standard Gaussian confidence intervals for $q \in (0, 1)$ even though the reward distributions themselves are not necessarily Gaussian. In principle though one could employ other confidence intervals, however, we achieved the best empirical results with this technique. The comparisons with other confidence intervals and even with distribution distance metrics like the Kolmogorov-Smirnov-distance are shown in the appendix in Other distribution discrimination techniques. For $q = 1$, the interval is defined as $(-\infty, \infty)$ and for $q = 0$ the interval is equal to the singleton set containing only the empirical mean of the quantity of interest.

If any confidence interval pair does not overlap, then a_j, a_k are separated. Note that this induces a symmetric and reflexive but not necessarily transitive relation over the root actions.

Optionally, to ensure that in the limit, AUPO does not group non-value-equivalent actions, we may additionally separate two actions, if the distribution of their returns differs significantly (in the sense that their mean and standard deviation confidence intervals do not overlap). The return of a trajectory is the (possibly discounted) sum of all its rewards. We call this option the return filter RF $\in \{0, 1\}$. Analogously, whether AUPO uses the standard deviation confidence intervals for distribution separation is denoted by the std filter SF $\in \{0, 1\}$.

Using the AUPO abstraction: We use the abstractions during the decision policy only. AUPO transforms the decision policy into a two-step process. In the first step, we assign each action a_j its abstract Q-value which is the sum of the returns divided by the sum of the visits of all actions a_j is grouped with. We select the action a^* that maximizes the abstract Q-value. Ties are broken randomly. In the second step, we select the action inside the abstraction of a^* with the highest unabstracted/ground Q-value. This decision policy makes AUPO a generalization of the greedy decision policy as for both $q \in \{0, 1\}$ AUPO's decision policy degenerates to the greedy policy. While for $q = 0$ step two becomes redundant, for $q = 1$ step one becomes redundant.

AUPO example: Next, we illustrate on an instance of the IPPC problem SysAdmin how AUPO detects abstractions. A detailed explanation is provided by Schmöcker et al. (Schmöcker, Dockhorn, and Rosenhahn 2025a). Assume we are in a state where all computers, except one outer computer, are online. This is visualized in Fig. 1a. This state features exactly four value-equivalent action types. Idling, rebooting the offline computer (machine 3), rebooting (even though it is still online) the hub computer (machine 0), or rebooting any outer running computer (machines 1-2,4-9). Given enough trajectory samples, AUPO separates and

Algorithm 1: AUPO

Parameters: q , D, *filter_std*, *filter_return*, *mcts_args*
Input: *state*

// Run MCTS and collect reward distribution data
$n =$ num_actions($state$) , $R[d, j] = [] \; \forall d, j$
for $i = 1 \ldots$ *mcts_iterations* **do**
  sample MCTS trajectory with rewards $r_1, \ldots, r_{D^*}$ and first action a_j
  for $d = 1 \ldots D$ **do**
    $R[d, j]$.append(r_d **if** $d \leq D^*$ **else** 0)
  end
  $R^*[j]$.append$\big(r_1 + \cdots + r_{\min(D, D*)}\big)$
end

// Compute confidence intervals
for $j = 1 \ldots n$ **do**
  for $d = 1 \ldots D$ **do**
    $mean\_interval[d, j] =$ mean_conf_interval($R[d, j], q$)
    $std_interval[d, j] =$std_conf_interval($R[d, j], q$)
  end
  $return_mean_interval[j] =$mean_conf_interval($R^*[j], q$)
  $return_std_interval[j] =$std_conf_interval($R^*[j], q$)
end

// Compute abstractions
for $i = 1 \ldots n$ **do**
  $abstract\_visits = 0, abstract\_value = 0$
  $abstraction[i] = \{\}$
  for $j = 1 \ldots n$ **do**
    $abstracted =$ **true**
    for $d = 1 \ldots D$ **do**
      if $mean\_interval[d, j] \cap mean\_interval[d, i] == \emptyset$ *or* $filter\_std$ *and* $std\_interval[d, j] \cap std\_interval[d, i] == \emptyset$ **then**
        $abstracted =$ **false**
      end
    end
    if $filter\_return$ *and (* $return\_mean\_interval[d, j] \cap return\_mean\_interval[d, i] == \emptyset$ *or* $filter\_std$ *and* $return\_std\_interval[d, j] \cap return\_std\_interval[d, i] == \emptyset$*)* **then**
      $abstracted =$ **false**
    end
    if $abstracted$ **then**
      $abstract\_visits+ = action\_visits(j)$
      $abstract\_value+ = action\_returns(j)$
      $abstraction[i]$.insert(j)
    end
    $abstract\_Q[i] = \frac{abstract\_value}{abstract\_visits}$
  end
end

// Action selection
$abs\_action =$ arg $\max_{i=1\ldots n}$ $abstract\_Q[i]$
$ground\_action =$ arg $\max_{i \in abstraction[abs\_action]}$ $Q[i]$
return $ground\_action$;

subsequently detects these equivalences as follows.

Idle action: All actions except idling have the same immediate reward, the reboot cost. Therefore, the idle action is easily separated by considering only the mean of the 1-step reward distribution.

Rebooting the offline computer: This action can be separated from the others by the 2-step reward distribution, as it takes one step for the computer to be rebooted and then another step to receive the reward from the additional running computer. Though a little noisy, the 2-step reward will be on average 1 higher than that of the other actions. We quantitatively verified this in the appendix in Tab. 2.

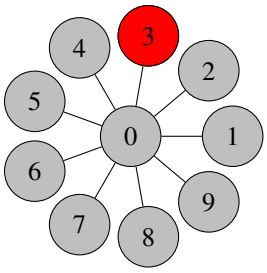
Rebooting the hub computer: This action can be separated from rebooting any of the outer running computers by the standard deviation of the 3-step reward. If we reboot the hub computer, we safeguard it from randomly crashing in the next step, which prevents the catastrophe where numerous other computers fail in the next step as they are connected to the then-broken hub computer. This scenario happens only rarely but when it does happen it is catastrophic, thus causing the 3-step reward of not rebooting the hub to have a relatively high variance compared to rebooting and thus protecting it. We quantitatively verified this in the appendix in Tab. 1.

Rebooting an outer running computer: Since they are symmetric and thus have identical reward distributions at all downstream steps, AUPO optimistically assumes that they are value-equivalent and thus abstracts them into a single action.

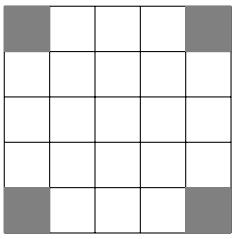
Relation to other abstraction frameworks: The abstractions AUPO detects include symmetric actions or more formally, actions whose sub game-trees (the state-transition graph rooted at the corresponding state-action pair) are isomorphic. However, AUPO is only able to detect a proper subset of value-equivalent actions and would miss, for example, that an action with an immediate reward of 0, which is always followed by an immediate reward of 1 could be equivalent to an action with an immediate reward of 1 followed 0.

In practice, AUPO is able to detect abstractions that ASAP could not because the latter requires the state graph to converge on states from which the abstraction building can be bootstrapped. Hence it is practically impossible for ASAP to detect equivalences that arise due to symmetry. For example, while it would be no problem for AUPO to detect that saving any of the four corner cells is value-equivalent in the Game of Life state visualized in Fig. 1b, ASAP would not be able to detect this with feasible computational resources. Game of Life is defined in the appendix section Problem models.

Furthermore, ASAP struggles with a high stochastic branching factor. While AUPO is able to detect that rebooting any of the outer machines from the SysAdmin example in Fig. 1a is value-equivalent, ASAP is not able to detect these equivalences if two value-equivalent actions have not sampled the exact same set of successors from which there are $33554432 = 2^{25}$.



(a) Visualization of a SysAdmin state where machine 3 has gone offline but all other machines are running.



(b) A 5×5 Game of Life configuration with only the four corner cells alive.

Figure 1: Visualization of two environments considered in this paper.

Experiments

In this section, we will present the setup and results for the comparison of AUPO with MCTS showing that AUPO is the first and currently only tree-search abstraction algorithm that does neither require access to the transition probabilities, nor the model having determinstic rewards, nor requires a directed acyclic search graph but can outperform MCTS.

Problem models: The problem models that we tested AUPO on are either problems from the International Conference on Probabilistic Planning (IPPC) (Grzes, Hoey, and Sanner 2014) or appear throughout the literature. For the readers not familiar with these problem models, these are explained in the appendix section Problem models. For details and the concrete instances, i.e. model parameter choices, we refer to our publicly available implementation (Schmöcker, Dockhorn, and Rosenhahn 2025b), which is the translation into C++ of the Relational Dynamic Influence Diagram (RDDL) (Sanner 2011) descriptions of these environments found at the RDDL repository (Taitler et al. 2022). These environments were deliberately chosen as they appear throughout the abstraction literature (Anand et al. 2015, 2016; Hostetler, Fern, and Dietterich 2015; Yoon et al. 2008; Jiang, Singh, and Lewis 2014) or have been used for planning competitions (Grzes, Hoey, and Sanner 2014), feature value-equivalent sibling actions, dense rewards, two theoretically necessary requirements for AUPO to yield any performance increase. We include the data for sparse-reward environments in Fig. 7 in the appendix to show that while AUPO cannot gain advantages in these environments, it at least causes no harm.

Experiment setup and reproducibility: For every experiment, we used a horizon of 50 episode steps. Since we are in the finite-horizon setting, we used a discount of $\gamma = 1$. We ran every experiment for at least 2000 episodes, and whenever we denote the mean return of this experiment we additionally provide a 99% confidence interval. We denote the confidence interval of any quantity by its mean

and the half of the interval size, e.g. we would denote a return confidence interval $(1, 3)$ by 2 ± 1. For both MCTS and AUPO, we performed random playouts until the episode terminates. Additionally, as the problem models vary in their reward scale, we used the dynamic exploration factor Global Std (Schmöcker, Schnell, and Dockhorn 2025) that is given by $C \cdot \sigma$ where σ is the empirical standard deviation of all Q values of the current search tree and $C \in \mathbb{R}^+$ is a parameter. For reproducibility, we released our implementation (Schmöcker, Dockhorn, and Rosenhahn 2025b). Our code was compiled with g++ version 13.1.0 using the -O3 flag (i.e. aggressive optimization). An overview of all experimental setting parameters is provided in Tab. 6 in the appendix.

Parameter-optimized performances: First, we tested whether and in which environments AUPO can increase the parameter-optimized performance over MCTS. To do this, we considered the best AUPO performance when varying the parameters exploration constant $C \in \{0.5, 1, 2, 4, 8, 16\}$, distribution tracking depth $D \in \{1, 2, 3, 4\}$, using the return filter RF $\in \{0, 1\}$, and varying the confidence level $q \in \{0.8, 0.9, 0.95, 0.99\}$. Furthermore, since the standard UCB tree policy results in non-uniformly distributed visits, we also considered AUPO's performance when using a uniform root policy (denoted as U-AUPO) which has two main effects. Firstly, each action, even those that UCB would not exploit, receive visits, thus shrinking their confidence intervals, making them easier to separate from other actions. And secondly, we reduce the risk of separating actions with layerwise identical reward distribution because in MCTS the distributions shift with an increasing visit count as MCTS starts to exploit.

We compared AUPO and U-AUPO to the performance of MCTS and MCTS with a uniform root policy U-MCTS, as well as RANDOM-ABS that is the same as AUPO except that for each action pair they are randomly abstracted at the decision policy with the probability $p_{\text{random}} \in \{0.1, 0.2, \ldots, 0.9\}$. Hence, RANDOM-ABS is equivalent to MCTS in the cases $p_{\text{random}} \in \{0, 1\}$. RANDOM-ABS verifies that the abstractions found by AUPO outperform randomly formed abstractions. To reduce the amount of visuals; any RANDOM-ABS data points are simply the maximum of both RANDOM-ABS with a uniform root policy and standard root policy. The parameter-optimized performances in dependence of the iteration number are visualized in Fig. 2. The following key observations can be made:

1) AUPO can gain a performance **advantage** over MCTS (and RANDOM-ABS) in **11 out of the 14 environments considered here**, in at least one iteration budget. In the environments, Academic Advising, Game of Life, Multi-armed bandit, Push Your Luck, Cooperative Recon, SysAdmin, and Traffic, AUPO maintained a performance edge for the majority of iteration budgets.

2) Expectedly, U-MCTS mostly performed worse than MCTS, however, the performance improvements between U-MCTS and U-AUPO is mostly significantly greater than the gap between MCTS and AUPO, showing that AUPO as suggested benefits from uniformly distributed visits. Notably, there is an environment, namely Cooperative Recon, in which MCTS and U-MCTS performed evenly, where however, U-AUPO clearly outperformed AUPO. Also, in Saving both U-MCTS and U-AUPO outperformed their non-uniform counterparts. Hence, using a uniform root policy can be a tool to improve the peak performance.

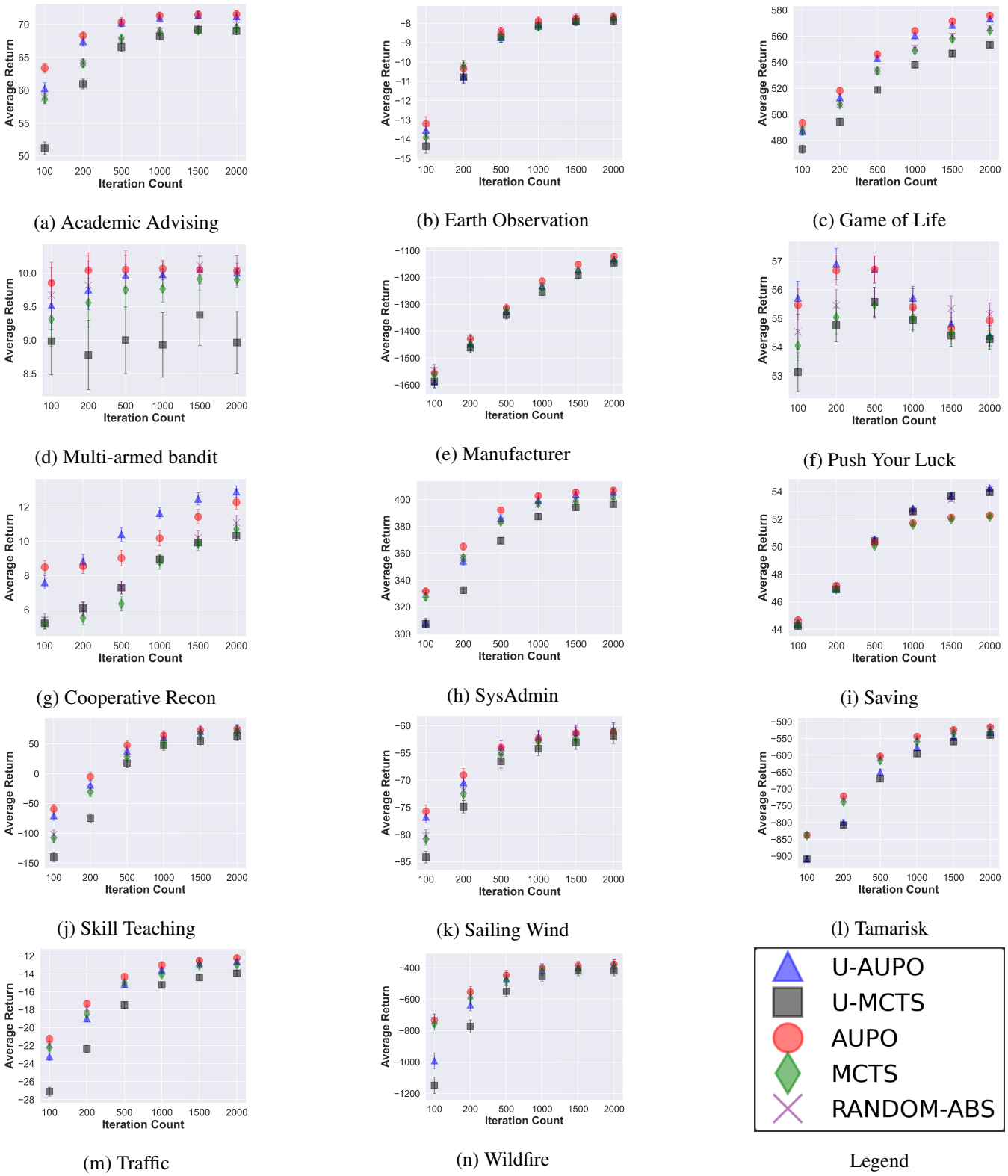
Generalization capabilities: Next, we tested AUPO's generalization capabilities. For this, we computed the pairings and relative improvement scores for all AUPO, U-AUPO, MCTS, U-MCTS, and RANDOM-ABS parameter combinations. These scores are Borda-like rankings of individual parameter-combinations and both lie in the interval $[-1, 1]$ (1 is the best value and -1 the worst) and they are formalized in the appendix in equations 19 and 21. The relative improvement score of an agent is the average (normalized) performance improvement when selecting a random environment and opponent agent. The pairings score of an agent is the difference of the ratio of opponent-environment pairs in which the agent obtained a higher performance and the ratio of pairs in which the agent performed worse. The results for all iteration budgets and environments combined are visualized in Fig. 3.

The results show that the best performances with respect to both scores with large margins, are reached with AUPO. These results are qualitatively identical for each iteration budget which is presented in the appendix in Tab. 4 and Tab. 5.

Ablations: Lastly, we studied the impact of the individual parameters. Instead of displaying the best-performing parameter set, we fixed the parameter in question and max over the remaining parameters. With only a few exception such as Multi-armed bandit, the confidence level did only have a significant impact for low iteration counts if it has any impact all. In the low iteration count regime, lower confidences generally outperform higher confidence levels. Depending on the environment, the impact of the distribution tracking depth can be significant to non-existent. In cases where it does matter, high depths are always preferred with the only exception being Push Your Luck. Filters can be extremely beneficial to some environments, such as Multi-armed Bandit or Push Your Luck Whilst not causing any harm to the environment where it has little impact. We visualize the concrete performance values for this ablation in the appendix Fig. 5 which shows the results when varying the confidence level, in Fig. 4, which shows the results when varying the distribution tracking depth, and in Fig. 6 that shows the results when varying either the std filter or the return filter.

Limitations and Future Work

In this paper, we introduced a novel action abstraction algorithm that we call AUPO which only affects the decision policy of MCTS. We could experimentally show that AUPO outperforms MCTS in a wide range of environments that contain states with value-equivalent sibling actions as well as dense rewards. Though AUPO introduces four new



(a) Academic Advising (b) Earth Observation (c) Game of Life

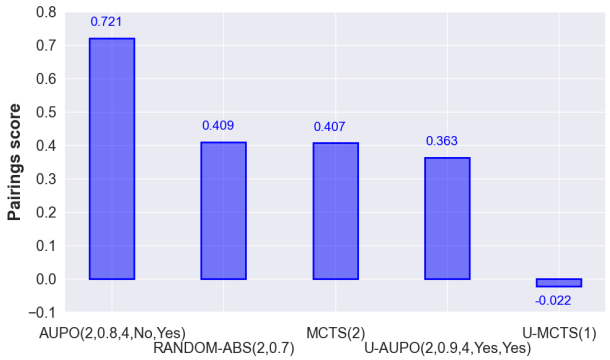
(d) Multi-armed bandit (e) Manufacturer (f) Push Your Luck

(g) Cooperative Recon (h) SysAdmin (i) Saving

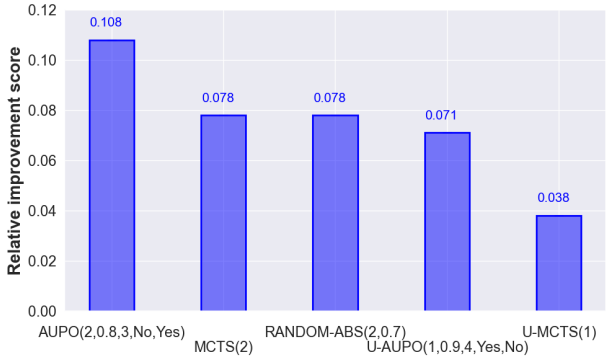
(j) Skill Teaching (k) Sailing Wind (l) Tamarisk

(m) Traffic (n) Wildfire Legend

Figure 2: The performance graphs of in dependence of the MCTS iteration count of the parameter optimized versions of AUPO, MCTS, and RANDOM-ABS. The prefix U- denotes AUPO and MCTS using a uniform root policy. To avoid p-hacking, the results of a differently seeded rerun of the best performing parameter combinations are plotted.



(a) Pairings score



(b) Relative improvement score

Figure 3: The pairings and relative improvement scores across all environments and iteration budgets for different AUPO, U-AUPO (parameter format $(C, q, D$,RF,SF)), MCTS, U-MCTS (parameter format (C)), and RANDOM-ABS (parameter format (C, p_{random})) agents. The bar charts show the top score reached by each agent type as well as the parameter combination to reach that score. In the case of RANDOM-ABS the score was reached with the standard root policy.

parameters, their choice mostly has only a minor impact on performance.

First and foremost, for AUPO to achieve any performance gain, the environment must contain state-action pairs with the same parent that have similar Q^* values, i.e. there need to be abstractions to be detected in the first place.

Another key limitation of AUPO is that it is reliant on dense-rewards. For example, in binary-outcome zero-sum two-player games AUPO would have a hard time distinguishing actions, as only the return distribution can be used for differentiation. How this limitation can be overcome, is left as future work.

Yet another weakness of AUPO is that it requires many visits for the distributions to be distinguishable; hence it cannot be used in low iteration settings and therefore not during the tree policy. Therefore, another area for future work is how to make AUPO much more sensitive to be able to deal with low iterations.

Furthermore, for future work, as mentioned in the introduction, it could also be of interest to combine AUPO with other abstraction algorithms. For example, one may use state-of-the-art such as OGA-UCT (Anand et al. 2016) during the search phase, replacing only the decision policy with AUPO. In its current form, AUPO uses the same confidence level for each layer. However, it might be worth investigating if additional performance can be achieved by making this parameter layer-dependent.

Lastly, it might be worth extending the set of evaluation environments by artificial dense reward environments which are created by setting the value of handcrafted heuristics or learned value functions as the reward.

References

Anand, A.; Grover, A.; Mausam; and Singla, P. 2015. ASAP-UCT: Abstraction of State-Action Pairs in UCT. In Yang, Q.; and Wooldridge, M. J., eds., *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 1509–1515. AAAI Press.

Anand, A.; Noothigattu, R.; Mausam; and Singla, P. 2016. 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, 29–37. AAAI Press. ISBN 1577357574.

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; Józefowicz, R.; Gray, S.; Olsson, C.; Pachocki, J.; Petrov, M.; de Oliveira Pinto, H. P.; Raiman, J.; Salimans, T.; Schlatter, J.; Schneider, J.; Sidor, S.; Sutskever, I.; Tang, J.; Wolski, F.; and Zhang, S. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. *CoRR*, abs/1912.06680.

Billings, D.; Burch, N.; Davidson, A.; Holte, R. C.; Schaeffer, J.; Schauenberg, T.; and Szafron, D. 2003. Approximating Game-Theoretic Optimal Strategies for Full-scale Poker. In Gottlob, G.; and Walsh, T., eds., *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, 661–668. Morgan Kaufmann.

Birge, J. R. 2007. Chapter 20 Optimization Methods in Dynamic Portfolio Management. In Birge, J. R.; and Linetsky, V., eds., *Financial Engineering*, volume 15 of *Handbooks in Operations Research and Management Science*, 845–865. Elsevier.

Browne, C.; Powley, E. J.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Liebana, D. P.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intell. AI Games*, 4(1): 1–43.

Childs, B. E.; Brodeur, J. H.; and Kocsis, L. 2008. Transpositions and move groups in Monte Carlo tree search. In Hingston, P.; and Barone, L., eds., *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games, CIG 2009, Perth, Australia, 15-18 December, 2008*, 389–395. IEEE.

Chitnis, R.; Silver, T.; Kim, B.; Kaelbling, L. P.; and Lozano-Pérez, T. 2020. CAMPs: Learning Context-Specific Abstractions for Efficient Planning in Factored MDPs. In Kober, J.; Ramos, F.; and Tomlin, C. J., eds., *4th Conference on Robot Learning, CoRL 2020, 16-18 November 2020, Virtual Event / Cambridge, MA, USA*, volume 155 of *Proceedings of Machine Learning Research*, 64–79. PMLR.

Friha, L.; Berry, P.; and Choueiry, B. Y. 1997. DISA: A Distributed scheduler using abstractions. *Revue d'Intelligence Artificielle*, 11.

Fu, Y.; Sun, M.; Nie, B.; and Gao, Y. 2023. Accelerating Monte Carlo Tree Search with Probability Tree State Abstraction. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Grzes, M.; Hoey, J.; and Sanner, S. 2014. International Probabilistic Planning Competition (IPPC) 2014. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Hoerger, M.; Kurniawati, H.; Kroese, D. P.; and Ye, N. 2024. Adaptive Discretization using Voronoi Trees for Continuous POMDPs. *Int. J. Robotics Res.*, 43(9): 1283–1298.

Hostetler, J.; Fern, A.; and Dietterich, T. G. 2015. Progressive Abstraction Refinement for Sparse Sampling. In Meila, M.; and Heskes, T., eds., *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands*, 365–374. AUAI Press.

Jiang, N.; Singh, S.; and Lewis, R. L. 2014. Improving UCT planning via approximate homomorphisms. In Bazzan, A. L. C.; Huhns, M. N.; Lomuscio, A.; and Scerri, P., eds., *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, 1289–1296. IFAAMAS/ACM.

Kwak, Y.; Hwang, I.; Kim, D.; Lee, S.; and Zhang, B. 2024. Efficient Monte Carlo Tree Search via On-the-Fly State-Conditioned Action Abstraction. In Kiyavash, N.; and Mooij, J. M., eds., *Uncertainty in Artificial Intelligence, 15-19 July 2024, Universitat Pompeu Fabra, Barcelona, Spain*, volume 244 of *Proceedings of Machine Learning Research*, 2076–2093. PMLR.

Liu, Q.; Li, X.; Yuan, S.; and Li, Z. 2021. Decision-Making Technology for Autonomous Vehicles: Learning-Based Methods, Applications and Future Outlook. In *24th IEEE International Intelligent Transportation Systems Conference, ITSC 2021, Indianapolis, IN, USA, September 19-22, 2021*, 30–37. IEEE.

Moraes, R. O.; and Lelis, L. H. S. 2018. Asymmetric Action Abstractions for Multi-Unit Control in Adversarial Real-Time Games. In McIlraith, S. A.; and Weinberger, K. Q., eds., *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 876–883. AAAI Press.

Ozair, S.; Li, Y.; Razavi, A.; Antonoglou, I.; van den Oord, A.; and Vinyals, O. 2021. Vector Quantized Models for Planning. In Meila, M.; and Zhang, T., eds., *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, 8302–8313. PMLR.

Sanner, S. 2011. Relational Dynamic Influence Diagram Language (RDDL): Language Description. Technical report, Australian National University. https://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf Accessed: 22-01-2025.

Schmöcker, R.; and Dockhorn, A. 2025. A Survey of Non-Learning-Based Abstractions for Sequential Decision-Making. *IEEE Access*, 13: 100808–100830.

Schmöcker, R.; Dockhorn, A.; and Rosenhahn, B. 2025a. Accelerating Probabilistic Planning Research: High-Speed Implementations of Stochastic Markov Decision Processes and Their Catalogization. *IEEE Access*, 13: 193430–193461.

Schmöcker, R.; Dockhorn, A.; and Rosenhahn, B. 2025b. Aupo. Repository available at: https://github.com/codebro634/Aupo.

Schmöcker, R.; Kampmann, L.; and Dockhorn, A. 2025. Time-Critical and Confidence-Based Abstraction Dropping Methods. In *2025 IEEE Conference on Games (CoG)*.

Schmöcker, R.; Schnell, C.; and Dockhorn, A. 2025. Investigating Scale Independent UCT Exploration Factor Strategies. arXiv:2510.21275.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T. P.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nat.*, 529(7587): 484–489.

Sogabe, T.; Malla, D. B.; Takayama, S.; Shin, S.; Sakamoto, K.; Yamaguchi, K.; Singh, T. P.; Sogabe, M.; Hirata, T.; and Okada, Y. 2018. Smart Grid Optimization by Deep Reinforcement Learning over Discrete and Continuous Action Space. In *2018 IEEE 7th World Conference on Photovoltaic Energy Conversion (WCPEC) (A Joint Conference of 45th IEEE PVSC, 28th PVSEC and 34th EU PVSEC)*, 3794–3796.

Sokota, S.; Ho, C.; Ahmad, Z. F.; and Kolter, J. Z. 2021. Monte Carlo Tree Search With Iteratively Refining State Abstractions. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 18698–18709.

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

Taitler, A.; Gimelfarb, M.; Gopalakrishnan, S.; Liu, X.; and Sanner, S. 2022. pyRDDLGym: From RDDL to Gym Environments. *CoRR*, abs/2211.05939.

Xu, L.; Dockhorn, A.; and Perez-Liebana, D. 2023. Elastic Monte Carlo Tree Search. *IEEE Transactions on Games*, 15(4): 527–537.

Yoon, S. W.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic Planning via Determinization in Hindsight. In Fox, D.; and Gomes, C. P., eds., *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 1010–1016. AAAI Press.