Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

ASAP-UCT: Abstraction of State-Action Pairs in UCT

Ankit Anand, Aditya Grover, Mausam and Parag Singla
Indian Institute of Technology, Delhi
New Delhi, India
{ankit.anand, adityag.cs111, mausam, parags}@cse.iitd.ac.in

Abstract

Monte-Carlo Tree Search (MCTS) algorithms such as UCT are an attractive online framework for solving planning under uncertainty problems modeled as a Markov Decision Process. However, MCTS search trees are constructed in flat state and action spaces, which can lead to poor policies for large problems. In a separate research thread, domain abstraction techniques compute symmetries to reduce the original MDP. This can lead to significant savings in computation, but these have been predominantly implemented for offline planning. This paper makes two contributions. First, we define the ASAP (Abstraction of State-Action Pairs) framework, which extends and unifies past work on domain abstractions by holistically aggregating *both* states and state-action pairs – ASAP uncovers a much larger number of symmetries in a given domain. Second, we propose ASAP-UCT, which implements ASAP-style abstractions within a UCT framework combining strengths of online planning with domain abstractions. Experimental evaluation on several benchmark domains shows up to 26% improvement in the quality of policies obtained over existing algorithms.

1 Introduction

The problem of sequential decision making, often modeled as a Markov Decision Process (MDP), is a fundamental problem in the design of autonomous agents [Russell and Norvig, 2003]. Traditional MDP planning algorithms (value iteration and variants) perform offline dynamic programming in flat state spaces and scale poorly with the number of domain features due to the curse of dimensionality. A well-known approach to reduce computation is through domain abstractions. Existing offline abstraction techniques [Givan *et al.*, 2003; Ravindran and Barto, 2004] compute equivalence classes of states such that all states in an equivalence class have the same value. This projects the original MDP computation onto an abstract MDP, which is typically of a much smaller size.

Recently, Monte-Carlo Tree Search (MCTS) algorithms have become quite an attractive alternative to traditional approaches. MCTS algorithms, exemplified by the well-known UCT algorithm [Kocsis and Szepesvári, 2006], intelligently sample parts of the search tree in an online fashion. They can be stopped anytime and usually return a good next action. A UCT-based MDP solver [Keller and Eyerich, 2012] won the last two probabilistic planning competitions [Sanner and Yoon, 2011; Grzes *et al.*, 2014]. Unfortunately, UCT builds search trees in the original flat state space too, which is wasteful if there are useful symmetries and abstractions in the domain.

A notable exception is [Jiang *et al.*, 2014], which introduced the first algorithm to combine UCT with automatically computed approximate state abstractions, and showed its value through quality gains for a single deterministic domain. Our preliminary experiments with this method (which we name AS-UCT) on probabilistic planning domains indicate that it is not as effective in practice. This may be because AS-UCT tries to compute state abstractions on the explored part of the UCT tree and there likely isn't enough information in the sampled trees to compute meaningful state abstractions.

In this paper, we develop a different notion of abstractions, *state-action pair* (SAP) abstractions, where in addition to computing equivalence classes of states, we also compute equivalence classes of state-action pairs, such that Q-values of state-action pairs in the same equivalence class are the same. SAP abstractions generalize previous notions – abstractions of both Givan and Ravindran. These are special cases of SAP abstractions. Moreover, SAP abstractions find symmetries even when there aren't many available state abstractions, which is commonly true for abstraction computations over undersampled UCT trees.

We implement SAP abstractions inside a UCT framework and call the resulting algorithm ASAP-UCT – **A**bstraction of **S**tate-**A**ction **P**airs in **UCT**. Experiments on several probabilistic planning competition problems show that ASAP-UCT significantly outperforms both AS-UCT and vanilla UCT obtaining upto 26% performance improvements. Overall, our contributions are:

1. We develop the theory of state-action pair abstractions and prove that it subsumes existing notions of abstractions in MDPs.
2. We implement and release[1] ASAP-UCT, an algorithm that exploits SAP abstractions in a UCT framework.

[1] Available at *https://github.com/dair-iitd/asap-uct*

3. We experimentally demonstrate the effectiveness of ASAP-UCT over baseline UCT and [Jiang *et al.*, 2014]'s AS-UCT.

2 Background and Related Work

An infinite horizon, discounted cost Markov Decision Process(MDP) [Puterman, 1994] is modeled as a 5-tuple $(S, A, \mathcal{T}, C, \gamma)$. An agent in a state $s \in S$ executes an action $a \in A$ making a transition to $s' \in S$ with a probability $\mathcal{T}(s, a, s')$ incurring a cost $C(s, a)$ with a discount factor of γ ($\gamma < 1$). A policy $\pi : S \rightarrow A$ specifies an action to be executed in a state $s \in S$. Given a starting state $s_0 \in S$, the expected discounted cost $V^\pi(s)$ associated with a policy π is given by $V^\pi(s^0) = E[\sum_{t=0}^{\infty} C(s^t, a^t)\gamma^t | \pi(s^t) = a^t, t \geq 0]$ where expectation is taken over the transition probability $\mathcal{T}(s^t, a^t, s^{t+1})$ of going from state s^t to s^{t+1} under action a^t. The expected cost $Q^\pi(s, a)$ denotes the discounted cost of first taking action a in state s and then following π from then on. The optimal policy π^* minimizes the total expected cost for every state $s \in S$, i.e. $\pi^*(s) = argmin_\pi V^\pi(s)$. $Q^*(s, a)$ and $V^*(s)$ are shorthand notations for $Q^{\pi^*}(s, a)$ and $V^{\pi^*}(s)$ respectively, and $V^*(s) = \min_{a \in A} Q^*(s, a)$. Presence of goals can be dealt by having absorbing states for goals.

An MDP can be equivalently represented as an AND-OR graph [Mausam and Kolobov, 2012] in which OR nodes are MDP states and AND-nodes represent state-action pairs whose outgoing edges are multiple probabilistic outcomes of taking the action in that state. Value Iteration [Bellman, 1957] and other dynamic programming MDP algorithms can be seen as message passing in the AND-OR graph where AND and OR nodes iteratively update Q(s,a) and V(s) (respectively) until convergence.

A finite-horizon MDP executes for a fixed number of steps (horizon) and minimizes expected cost (or maximizes expected reward). States for this MDP are (s, t) pairs where s is a world state and t is number of actions taken so far. Finite horizon MDPs can be seen as a special case of infinite horizon MDPs by having all the states at the horizon be absorbing goal states and setting $\gamma = 1$.

2.1 Abstractions for Offline MDP Algorithms

In many MDP domains, several states behave identically, and hence, can be abstracted out. Existing literature defines abstractions via an equivalence relation $\mathcal{E} \subseteq S \times S$, such that if $(s, s') \in \mathcal{E}$, then their state transitions are equivalent (for all actions). All states in an equivalence class can be collapsed into a single aggregate state in an abstract MDP, leading to significant reductions in computation.

Various definitions for computing abstractions exist. Givan et al. [2003]'s conditions deduce two states to have an equivalence relation if they have the same applicable actions, local transitions lead to equivalent states and immediate costs are the same. Ravindran and Barto [2004] refine this by allowing the applicable actions to be different as long as they can be mapped to each other for this state pair. This can find more state abstractions than Givan's conditions. We call these settings AS (Abstractions of States) and ASAM (Abstractions of States with Action Mappings), respectively.

Our framework unifies and extends these previous notions of abstractions – we go beyond just an equivalence relation $\mathcal{E}$ over states, and compute equivalences of *state-action pairs*. This additional notion of abstractions leads to a discovery of many more symmetries and obtains significant computational savings when applied to online algorithms.

2.2 Monte-Carlo Tree Search (MCTS)

Traditional offline MDP algorithms store the whole state space in memory and scale poorly with number of domain features. Sampling-based MCTS algorithms offer an attractive alternative. They solve finite-horizon MDPs in an online manner by interleaving planning and execution steps. A popular variant is UCT [Kocsis and Szepesvári, 2006], in which during the planning phase, starting from the root state, an expectimin tree is constructed based on sampled trajectories. At each iteration, the tree is expanded by adding a leaf node. Since these MDPs are finite horizon a node is (state,depth) pair. UCT chooses an action a in a state s at depth d based on the UCB rule, $argmin_{a \in A} \left(Q(s, d, a) - K \times \sqrt{\frac{log(n(s,d))}{n(s,d,a)}} \right)$ where $K > 0$. Here, $n(s, d)$ denotes the number of trajectories that pass through the node (s, d) and $n(s, d, a)$ is the number of trajectories that take action a in (s, d).

Evaluation of a leaf node is done via a random *rollout*, in which actions are randomly chosen based on some default rollout policy until a goal or some planning horizon P is reached. This rollout results in an estimate of the Q-value at the leaf node. Finally, this Q-value is backed up from the leaf to the root. UCT operates in an anytime fashion – whenever it needs to execute an action it stops planning and picks the best action at the root node based on the current Q-values. The planning phase is then repeated again from the newly transitioned node. Due to the clever balancing of the exploration-exploitation trade off, MCTS algorithms can be quite effective and have been shown to have significantly better performance in many domains of practical interest[Gelly and Silver, 2011; Balla and Fern, 2009].

2.3 Abstractions for UCT

Hostetler *et al.* [2014] develop a theoretical framework for defining a series of state abstractions in sampling-based algorithms for MDP. But they do not provide any automated algorithm to compute the abstractions themselves. Closest to our work is [Jiang *et al.*, 2014], which applies Givan's definitions of state abstractions within UCT. The key insight is that instead of an offline abstraction algorithm, they test abstractions only for the states enumerated by UCT. Their approach first puts all partially explored nodes (not all of whose actions are present in UCT tree) in a single abstract state per depth. Since UCT solves finite-horizon MDPs, only the states at the same depth will be considered equivalent. Then, at any given depth, they test Givan's conditions (transition and cost equality) on pairs of states to identify ones that are in the same equivalence class. This algorithm proceeds bottom-up starting from last depth all the way to the root. Jiang et al. also consider two approximation parameters $\epsilon_\mathcal{T}$ and $\epsilon_\mathcal{C}$ which aggregate two states

together if the corresponding transition probabilities and costs are within $\epsilon_{\mathcal{T}}$ and ϵ_C, respectively. Their paper experimented on a single deterministic game playing domain and its general applicability to planning was not tested. We advance Jiang's ideas by applying our novel SAP abstractions in UCT, and show that they are more effective on a variety of domains.

3 ASAP framework

A preliminary treatment of our *Abstractions of State-Action Pairs* (ASAP) framework appears in our previous work [Anand *et al.*, 2015]. The ASAP framework unifies and extends Givan's and Ravindran's definitions for computing abstractions. To formally define the framework we introduce some notation. Consider an MDP $M = (S, A, \mathcal{T}, C, \gamma)$. We use P to denote the set of State-Action Pairs (SAPs) i.e. $P = S \times A$. We define an equivalence relation $\mathcal{E}$ over pairs of states i.e. $\mathcal{E} \subseteq S \times S$. Let $\mathcal{X}$ denote the set of equivalence classes under the relation $\mathcal{E}$ and let $\mu_{\mathcal{E}} : S \to \mathcal{X}$ denote the corresponding equivalence function mapping each state to the corresponding equivalence class. Similarly, we define an equivalence relation $\mathcal{H}$ over pairs of SAPs, i.e. $\mathcal{H} \subseteq P \times P$). Let $\mathcal{U}$ denote the set of equivalences classes under the relation $\mathcal{H}$, and let $\mu_{\mathcal{H}} : P \to \mathcal{U}$ denote the corresponding equivalence function mapping state-action pairs to the corresponding equivalence classes. Following previous work, we will recursively define state equivalences using state-pair equivalences and vice-versa.

Definition 1 (State Abstractions) Suppose we are given SAP abstractions, and $\mu_{\mathcal{H}}$. Intuitively, for state equivalence to hold, there should be a correspondence between applicable actions in the two states such that the respective state-action pair nodes are equivalent. Formally, let $a, a' \in A$ denote two actions applicable in s and s', respectively. We say that two states s and s' are equivalent to each other (i.e, $\mu_{\mathcal{E}}(s) = \mu_{\mathcal{E}}(s')$) if for every action a applicable in s, there is an action a' applicable in s' (and vice-versa) such that $\mu_{\mathcal{H}}(s, a) = \mu_{\mathcal{H}}(s', a')$.

Definition 2 (SAP Abstractions) As in the case of state abstractions, assume we are given state abstractions and the $\mu_{\mathcal{E}}$ function. Two state-action pairs (s, a), $(s', a') \in P$ are said to be equivalent i.e. $\mu_{\mathcal{H}}(s, a) = \mu_{\mathcal{H}}(s', a')$ iff:

- $\forall x \in \mathcal{X}$, $\sum_{s_t \in S} \mathcal{I}[\mu_{\mathcal{E}}(s_t) = x]\mathcal{T}(s, a, s_t) = \sum_{s'_t \in S} \mathcal{I}[\mu_{\mathcal{E}}(s'_t) = x]\mathcal{T}(s', a', s'_t)$ where $\mathcal{I}$ is indicator function (Condition 1)
- $C(s, a) = C(s', a')$ (Condition 2)

In other words, for state-action pair equivalence to hold, the sum of transition probabilities, to each abstract state that these state-action pair transition to, should match. Second condition requires the costs of applying corresponding actions to be identical to each other.

Base Case: For goal-directed infinite horizon MDPs, all goal states are in an equivalence class: $\forall s, s' \in G, \mu_{\mathcal{E}}(s) = \mu_{\mathcal{E}}(s')$. For finite-horizon case, all goal states at a given depth are equivalent. Moreover, all non-goal states at the end of the horizon are also equivalent. Repeated application of these definitions will compute larger sets of abstractions, until convergence.

It is important to note that ASAP framework does not reduce the original MDP M into an abstract MDP as done in earlier work. The reason is that the framework may abstract two state-action pairs whose parent states are not equivalent. It is not possible to represent this as an MDP, since MDP does not explicitly input a set of SAPs. However, ASAP framework can be seen as directly abstracting the equivalent AND-OR graph Gr, in which there is an OR node for each abstract state $x \in \mathcal{X}$, and an AND node for each abstract SAP $u \in \mathcal{U}$. There is an edge from x to u in Gr if there is a state-action pair (s, a) such that a is applicable in s, $\mu_{\mathcal{E}}(s) = x$ and $\mu_{\mathcal{H}}(s, a) = u$. The associated cost with this edge is $C(s, a)$ which is the same for every such (s, a) pair (follows from Condition 2). Similarly, there is an edge from u to x if there is a state-action-state triplet (s, a, s_t) such that application of a in s results in s_t (with some non-zero probability), $\mu_{\mathcal{H}}(s, a) = u$ and $\mu_{\mathcal{E}}(s_t) = x$. The associated transition probability is $\sum_{s'_t \in S} \mathcal{I}[\mu_{\mathcal{E}}(s'_t) = x]\mathcal{T}(s, a, s'_t)$ (follows from Condition 1).

Example: Figure 1 illustrates the AND-OR graph abstractions on a soccer domain. Here, four players wish to score a goal. The central player (S0) can pass the ball left, right or shoot at the goal straight. The top player (S1) can hit the ball right to shoot the goal. Two players at the bottom (S2, S3) can hit the ball left for a goal. It should be noted that right action (S0,R) of central player (S0) leads to (S2) and (S3) with uniform probabilities. The equivalent AND-OR graph for this domain is the leftmost graph in the figure. Givan's AS conditions check for exact action equivalence. They will observe that S2 and S3 are redundant players and merge the two states. Ravindran's ASAM conditions will additionally look for mappings of actions. They will deduce that S1's right is equivalent to S2's left and will merge these two states (and actions) too. They will also notice that S0's left and right are equivalent. Finally, our ASAP framework will additionally recognize that S0's straight is equivalent to S1's right and merge these two SAP nodes. Overall, ASAP will identify the maximum symmetries in the problem. □

ASAP framework is a strict generalization of past approaches. Here, it is important to state that our SAP abstraction definitions look similar to Ravindran's ASAM, but there is a key difference. They constrained SAPs to be equivalent only if the parent states were equivalent too. They did not directly use SAP abstractions in planning – they used those merely as a subroutine to output state abstractions. Our definitions differ by computing SAP equivalences irrespective of whether the parent states are equivalent or not, and thereby reducing the AND-OR graph of the domain further.

Theorem 1. *Both AS and ASAM are special cases of ASAP framework. ASAP will find all abstractions computed by AS and ASAM.*

ASAP reduces to ASAM with an additional constraint that $\mu_{\mathcal{E}}(s) = \mu_{\mathcal{E}}(s')$ when computing SAP abstractions for pair (s, a) and (s', a'). AS is simply ASAP with a further constraint $a = a'$ in Conditions 1 and 2.

Finally, we can also prove correctness of our framework, i.e., an optimal algorithm operating on our AND-OR graph converges to the optimal value function.

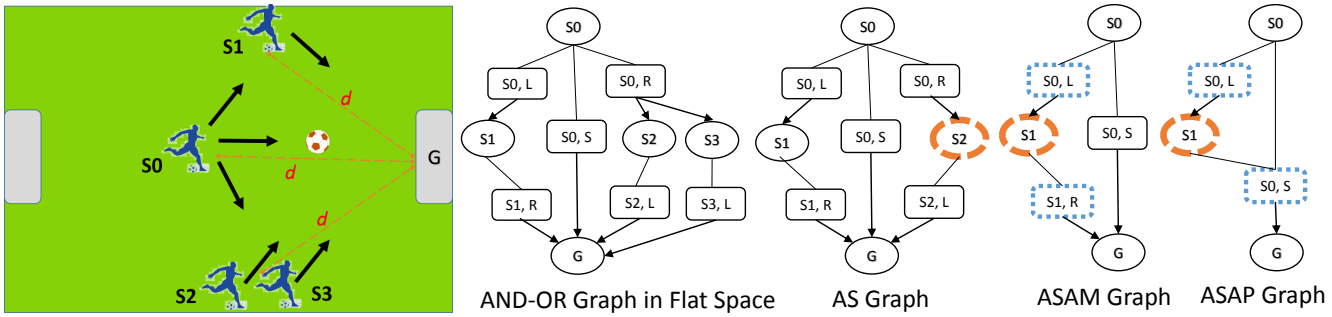

Figure 1: An example showing abstractions generated by various algorithms on a soccer domain. Givan's AS, Ravindran's ASAM and our ASAP frameworks successively discover more and more symmetries.

Theorem 2. *Optimal value functions $V^*_{Gr}(x)$, $Q^*_{Gr}(u)$, computed by Value Iteration on a reduced AND-OR graph Gr, return optimal value functions $V^*_M(s)$, $Q^*_M(s,a)$ for the original MDP M. Formally, $V^*_{Gr}(x) = V^*_M(s)$, and $Q^*_{Gr}(u) = Q^*_M(s,a)$, $\forall s \in S,\ a \in A$ s.t. $\mu_{\mathcal{E}}(s) = x, \mu_{\mathcal{H}}(s,a) = u$.*

4 ASAP-UCT

We now present ASAP-UCT, a UCT-based algorithm that uses the power of abstractions computed via the ASAP framework. Since UCT constructs a finite-horizon MDP tree (see Section 2.2), states at different depths have to be treated differently. Therefore, ASAP-UCT tests state equivalences only for the states at the same depth. To compute abstractions over UCT tree, we adapt and extend ideas presented by Jiang et al. [2014].

How to Compute Abstractions: Algorithm 1 gives the pseudocode for computing abstractions using ASAP-UCT. The algorithm takes as input a UCT Search Tree (ST) and outputs an Abstracted Search Tree (AST). Starting from the leaves of the UCT tree, it successively computes abstractions at each level (depth) all the way up to the root. At each level, it calls the functions for computing state and state-action pair abstractions alternately. It is helpful to understand each depth as consisting of a layer of state nodes and a layer of SAP nodes above it. We use $\mu^d_{\mathcal{E}}$ ($\mu^d_{\mathcal{H}}$) to denote the state (SAP) equivalence function at depth d. Similarly, we use S^d to denote the set of states at depth d and P^d to denote the set of SAP nodes at depth d. To keep the notation simple, we overload the equivalence function (map) $\mu^d_{\mathcal{E}}$ to also represent the actual equivalence relationship over state pairs (and similarly for $\mu^d_{\mathcal{H}}$).

Algorithm 2 gives the pseudocode for the computation of state abstractions at a depth using conditions from the previous section. *getPartiallyExplored*(S^d) returns the subset of states at depth d not all of whose applicable actions have been explored. *Apply*(s) returns the set of all the actions applicable in s. Following Jiang et al. [2014], we mark all the partially explored nodes at each depth to be in the same equivalence class. This is an approximation, but is necessary due to the limited information about states available in UCT tree at any given time. For the states all of whose applicable actions have been explored, we put states (s, s') in the same equivalence class if conditions for state equivalence (Definition 1) are satisfied.

Algorithm 3 gives the pseudocode for computing SAP abstractions. Out(s,a) returns the set of all states sampled in the UCT tree after application of a in s. $\mathcal{T}_X$ and $\mathcal{T}'_X$ are arrays storing the total transition probabilities to each abstract state at next level for the two SAPs being compared, respectively. Two SAP nodes (s, a) and (s', a') are put in same equivalence class if conditions for state-action pair equivalence (Definition 2) are satisfied.

Algorithm 1 Computing Abstract Search Tree

ComputeAbstractSearchTree(SearchTree ST)
  $d_{max} \leftarrow$ getMaxDepth(ST), $\mu^{d_{max}+1}_{\mathcal{H}} \leftarrow \{\}$
  for $d := d_{max} \rightarrow 1$ **do**
    $\mu^d_{\mathcal{E}} \leftarrow$ ComputeAS(S^d, $\mu^{d+1}_{\mathcal{H}}$);
    $\mu^d_{\mathcal{H}} \leftarrow$ ComputeASAP(P^d, $\mu^d_{\mathcal{E}}$);
  end for
  $AST \leftarrow$ SearchTree with Computed Abstractions
  Initialize Q-Values of abstract nodes
  return AST

Algorithm 2 Abstraction of States

ComputeAS(States S^d, Eq-Map $\mu^{d+1}_{\mathcal{H}}$)
  $S^d_L \leftarrow$ getPartiallyExplored(S^d)
  $\forall s, s' \in S^d_L, \mu^d_{\mathcal{E}}(s) = \mu^d_{\mathcal{E}}(s')$ (base case)
  $S^d_I \leftarrow S^d \setminus S^d_L$
  for all $s, s' \in S^d_I$ **do**
    $mapping = True$;
    for all $a \in Apply(s)$ **do**
      If($\nexists a' \in Apply(s')$: $\mu^{d+1}_{\mathcal{H}}(s,a) = \mu^{d+1}_{\mathcal{H}}(s',a')$)
        $mapping = False$;
    end for
    for all $a' \in Apply(s')$ **do**
      If($\nexists a \in Apply(s)$: $\mu^{d+1}_{\mathcal{H}}(s',a') = \mu^{d+1}_{\mathcal{H}}(s,a)$)
        $mapping = False$;
    end for
    if($mapping$) **then** $\mu^d_{\mathcal{E}}(s) = \mu^d_{\mathcal{E}}(s')$
  end for
  return $\mu^d_{\mathcal{E}}$

Updating Q-Values: For all the nodes belonging to the same equivalence class, we maintain a single estimate of the ex-

pected cost to reach the goal both in the state as well as state-action layers. In the beginning, Q-values for an abstract node are initialized with the average of the expected cost of its constituents as shown in Algorithm 1. During the backup phase, any Q-value update for a node in the original tree is shared with all the nodes belonging to the same equivalence class in the abstract tree, effectively replicating the rollout sample for every node. It should be noted that node expansion as well as the random roll out after expansion in the UCT tree are still done in the original flat (unabstracted) space. It is only during the upward update of the Q-value computation where abstractions are used.

When to Compute Abstractions: In order to compute abstractions, we need to construct the sampled UCT tree upto a certain level. But waiting until the full expansion of tree is not helpful, since the tree would already be constructed and we would not be able to use the abstractions. In our approach, we start by allocating a decision time d_t for the current step using a dynamic time allocation strategy (described below). Similar to Jiang et al. [2014], we then interleave the steps of tree expansion with abstraction computation. Abstractions are computed after every $d_t/(l+1)$ time units where l is a parameter which can be tuned (we used $l = 1$ in our experiments). Since future expansions might invalidate the currently computed abstractions, every phase of abstraction computation is done over the flat tree. Algorithm 4 provides the pseudocode for the above procedure. ST denotes the search tree in the flat space and AST denotes the search tree in the abstract space.

Algorithm 3 Abstraction of State-Action Pairs

ComputeASAP(States-Action Pairs P^d, Eq-Map $\mu^d_{\mathcal{E}}$)
  for all $p = (s, a)$, $p' = (s', a') \in P^d$ **do**
    $\forall x \in \mu^d_{\mathcal{E}}$, $\mathcal{T}_X[x] = 0$ and $\mathcal{T}'_X[x] = 0$;
    for all $s_i \in Out(s, a)$ **do**
      $\mathcal{T}_X[\mu^d_{\mathcal{E}}(s_i)] += \mathcal{T}(s, a, s_i)$
    end for
    for all $s'_i \in Out(s', a')$ **do**
      $\mathcal{T}'_X[\mu^d_{\mathcal{E}}(s'_i)] += \mathcal{T}(s', a', s'_i)$
    end for
    if($\forall x \in \mu^d_{\mathcal{E}}$: $\mathcal{T}_X(x) = \mathcal{T}'_X(x)$ & $C(s, a) = C(s', a')$)
      then $\mu^d_{\mathcal{H}}(s, a) = \mu^d_{\mathcal{H}}(s', a')$
  end for
  return $\mu^d_{\mathcal{H}}$

Algorithm 4 ASAP-UCT Algorithm

ASAP-UCT(StartNode S_0, NumAbstractions l)
  $ST, AST \leftarrow S_0$ //Single Node Search Tree
  $d_t \leftarrow$ getDecisionTime()
  while d_t is not exhausted **do**
    $AST \leftarrow$ ExpandTreeAndUpdateQ(AST)
    After every $d_t/(l+1)$ time units
      $ST \leftarrow$ getFlatTree(AST)
      $AST \leftarrow$ ComputeAbstractSearchTree(ST)
  end while
  return $argmin_{a \in A} Q^*(S_0, a)$ //best action at S_0

Adaptive Planning Time Allocation: We assume that the agent is given an execution horizon (maximum number of decisions to be taken) and a total process time. This setting has been previously used in the literature for online planning with UCT [Kolobov *et al.*, 2012]. A key meta-reasoning problem is the allocation of available time across various decision steps. Naïvely, we may decide to allocate equal time per decision. However, this can be wasteful in goal-directed settings since the goal can be reached sooner than the execution horizon. In such scenarios, the time allocated for remaining decisions will be wasted. To counter this, we adapt prior work on meta-reasoning algorithms for UCT [Baier and Winands, 2012; Baudiš and Gailly, 2012] to implement an adaptive time allocation strategy. Our approach uses random roll-outs performed during UCT to continually re-estimate the expected number of steps to reach the goal (effective execution horizon). This is then used to non uniformly divide the remaining execution time across future decision steps. Since we would like to allocate less time to later decisions when goal is closer, we use a decreasing arithmetic progression to divide the available total running time.

Efficiently Implementing Abstraction Computation: Computing SAP abstractions naïvely would require $O(n^2)$ comparisons, n being the total number of SAP nodes at each level (we would compare each SAP pair for equivalence). This can become a bottleneck for large problems with large number of states and applicable actions. We instead hash each SAP using its total transition probability to set of next abstract states as well the cost associated with the transition. Only the pairs falling in the same hash bucket now need to be compared bringing down the complexity to $O(rk^2)$ where r is the number of buckets and k is the number of elements in each bucket. This is crucial for scaling ASAP-UCT as observed in our experiments.

5 Experimental Evaluation

Our experiments aim to study the comparative performance of various abstraction approaches in UCT. In Section 5.2 we compare ASAP-UCT with vanilla UCT [Kocsis and Szepesvári, 2006], AS-UCT [Jiang *et al.*, 2014], and ASAM-UCT (our novel combination of UCT and Ravindran and Barto [2004]'s ASAM).

All our experiments are performed on a Quad-Core Intel i-5 processor. For each parameter configuration we take an average of 1000 trials. We use 100 as the execution horizon for Sailing wind and Navigation domains and 40 for Game of Life domain (as per IPPC instance), although reaching the goal earlier will stop the execution. The planning horizon for a decision is an input to each problem. All UCT roll-outs use random actions. The exploration constant K for the UCB equation is set as the negative of the magnitude of current Q value at the node (following [Bonet and Geffner, 2012]). Also, in cases of abstraction computation approaches, we need to set l, the number of times abstractions are computed for each decision. Since, computing abstractions can be expensive, l must be a small number. In our experiments we find that the setting of l as 1 works well for all systems. How to set this automatically is an important question for future

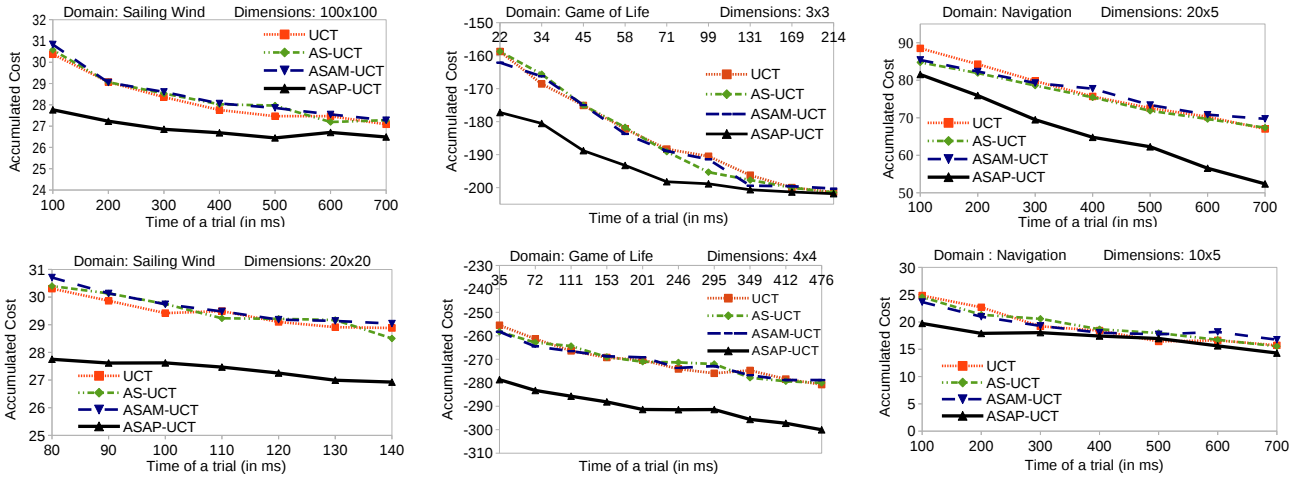

Figure 2: ASAP-UCT outperforms all other algorithms on problems from three domains.

work. All algorithms use the adaptive time-allocation strategy, which performs much better than equal time-allocation.

5.1 Domains

We experiment on three diverse domains, Sailing Wind [Kocsis and Szepesvári, 2006; Bonet and Geffner, 2012], Game of Life [Sanner and Yoon, 2011], and Navigation [Sanner and Yoon, 2011]. We briefly describe these below.

Sailing Wind: An agent in this domain is assigned the task of sailing from one point to another on a grid. The agent can move in any direction to an adjacent cell on the grid, as long as it is not against the stochastic wind direction. The cost at any time step is a function of the agent movement and the wind direction. We report on two instances with grid dimensions 20×20 and 100×100 (#states: $3200, 80000$).

Game of Life: An oracle agent is given the objective to maximize the number of alive cells in a cellular automata environment modeled as a grid. Live cells continue into the next generation as long as there is no overpopulation or underpopulation as measured by the number of adjacent cells. Similarly, a dead cell can become alive in the next generation due to a reproduction between adjacent cells. Additionally, an agent can make exactly one cell live on to the next generation. The dynamics of the game are stochastic in nature, set differently for different cells. The number of live cells determines the reward at every time step. Our empirical results are reported on two IPPC-2011 instances, of dimensions 3×3 and 4×4 (#states: $2^9, 2^{16}$).

Navigation: A robot has to navigate from a point on one side of the river approximated as a grid to a goal on the other side. The robot can move in one of the four possible directions. For each action, the robot arrives in a new cell with a non-zero drowning probability and unit cost. If a robot drowns, it will retry navigating from the start state. We report on two IPPC instances of sizes 20×5 and 10×5 (#states: $100, 50$).

5.2 ASAP-UCT vs. Other UCT Algorithms

We compare the four algorithms, vanilla UCT, AS-UCT, ASAM-UCT and ASAP-UCT, in all three domains. For each domain instance we vary the total time per trial and plot the average cost obtained over 1000 trials. As the trial time increases each algorithm should perform better since planning time per step increases. Also, we expect the edge of abstractions over vanilla UCT to reduce given sufficient trial time.

Figures 2 shows the comparisons across these six problems in three domains. Note that time taken for a trial also includes the time taken to compute the abstractions. In almost all settings ASAP-UCT vastly outperforms both UCT, AS-UCT and ASAM-UCT. ASAP-UCT obtains dramatically better solution qualities given very low trial times for Sailing Wind and Game of Life, incurring up to 26% less cost compared to UCT. Its overall benefit reduces as the total trial time increases, but almost always it continues to stay better or at par. We conducted one tailed student's t-test and found that ASAP-UCT is statistically significantly better than other algorithms with a significance level of 0.01 (99% confidence interval) in 41 out of 42 comparisons (six graphs, 7 points each). The corresponding error bars are very small and hence, not visible in the figures.

Discussion: We believe that the superior performance of ASAP-UCT is because of its effectiveness in spite of noise in sampled trees. AS and ASAM conditions are strict as they look for all pairwise action equivalences before calling two states equivalent. Such complete set of equivalences are hard to find in UCT trees where some outcomes may be missing due to sampling. ASAP-UCT, in contrast, can make good use of any partial (SAP) equivalences found. ASAM's performance does not improve over AS probably because its gain due to symmetries is undermined by increase in abstraction computation time.

We also experimented with the Canadian Traveler Problem (CTP) [Bonet and Geffner, 2012] and Sysadmin [Guestrin *et al.*, 2003], but found that no abstraction algorithm gives performance gains over vanilla UCT. For CTP, we believe this is due to lack of symmetries in the domain. This suggests that some domains may not be that amenable to abstractions which is not too surprising. For the Sysadmin domain, we are able to find symmetries, however, due to exponential stochas-

tic branching factor (in number of state variables), we are not able to exploit them efficiently in UCT. Our empirical findings validate this claim as time taken to compute abstractions in this domain is as high as $\approx 50\%$ of total time for a trial. Handling such domains is a part of future work.

Effect of Approximation Parameters: In our experiments, AS-UCT does not always perform better than UCT, rather, it often underperforms. This is surprising and contradicts previous observations [Jiang *et al.*, 2014], which were based upon a single, deterministic domain of Othello. This could be due to the fact that our experiments set the parameters $\epsilon_{\mathcal{T}}$ and $\epsilon_{\mathcal{C}}$ (see Section 2.3) zero for abstraction frameworks. However, we find that even after incorporating a range of approximation parameters in AS-UCT, ASAP-UCT without those parameters continues to perform significantly better.

6 Conclusion and Future Work

This paper develops a novel class of state-action pair (SAP) abstractions, which generalizes and extends past work on abstractions in MDPs. SAP abstractions find more symmetries in a domain compared to existing approaches and convert an input MDP into a reduced AND-OR graph. We present a new algorithm, ASAP-UCT, which computes these abstractions in an online UCT framework. ASAP-UCT obtains significantly higher quality policies (up to 26% reduction in policy costs) compared to previous approaches on three benchmark domains.

We have released our implementation for a wider use by the research community. In the future, we plan to extend our implementation to handle really large problems that are described in a factored or first-order representation such as RDDL [Sanner, 2010]. We will also investigate the potential boosts in solution quality by integrating abstractions with recent work in factored MCTS [Cui *et al.*, 2015].

Acknowledgements

We are grateful to Blai Bonet and Hector Geffner for sharing the base code of UCT. We thank Nan Jiang and Balaraman Ravindran for helpful and interesting discussions. Ankit Anand is supported by TCS Research Scholars Program. Mausam is supported by a Google research award.

References

[Anand *et al.*, 2015] Ankit Anand, Aditya Grover, Mausam, and Parag Singla. A Novel Abstraction Framework for Online Planning. In *AAMAS*, 2015.

[Baier and Winands, 2012] Hendrik Baier and Mark HM Winands. Time Management for Monte-Carlo Tree Search in Go. In *Advances in Computer Games*. Springer, 2012.

[Balla and Fern, 2009] Radha-Krishna Balla and Alan Fern. UCT for Tactical Assault Planning in Real-Time Strategy Games. In *IJCAI*, 2009.

[Baudiš and Gailly, 2012] Petr Baudiš and Jean-loup Gailly. Pachi: State of the art open source Go program. In *Advances in Computer Games*. Springer, 2012.

[Bellman, 1957] Richard Bellman. A Markovian Decision Process. *Indiana University Mathematics Journal*, 1957.

[Bonet and Geffner, 2012] Blai Bonet and Hector Geffner. Action Selection for MDPs: Anytime AO* Versus UCT. In *AAAI*, 2012.

[Cui *et al.*, 2015] Hao Cui, Roni Khardon, Alan Fern, and Prasad Tadepalli. Factored MCTS for Large Scale Stochastic Planning. In *AAAI*, 2015.

[Gelly and Silver, 2011] Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[Givan *et al.*, 2003] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1–2):163 – 223, 2003.

[Grzes *et al.*, 2014] Marek Grzes, Jesse Hoey, and Scott Sanner. International Probabilistic Planning Competition (IPPC) 2014. In *ICAPS*, 2014.

[Guestrin *et al.*, 2003] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored MDPs. *J. Artif. Intell. Res.(JAIR)*, 19:399–468, 2003.

[Hostetler *et al.*, 2014] Jesse Hostetler, Alan Fern, and Tom Dietterich. State Aggregation in Monte Carlo Tree Search. In *AAAI*, 2014.

[Jiang *et al.*, 2014] Nan Jiang, Satinder Singh, and Richard Lewis. Improving UCT Planning via Approximate Homomorphisms. In *AAMAS*, 2014.

[Keller and Eyerich, 2012] Thomas Keller and Patrick Eyerich. PROST: Probabilistic Planning Based on UCT. In *ICAPS*, 2012.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML*. Springer, 2006.

[Kolobov *et al.*, 2012] Andrey Kolobov, Mausam, and Daniel S. Weld. LRTDP versus UCT for online probabilistic planning. In *AAAI*, 2012.

[Mausam and Kolobov, 2012] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Morgan & Claypool Publishers, 2012.

[Puterman, 1994] M.L. Puterman. *Markov Decision Processes*. John Wiley & Sons, Inc., 1994.

[Ravindran and Barto, 2004] Balaraman Ravindran and Andrew Barto. Approximate homomorphisms: A framework for nonexact minimization in Markov decision processes. In *Int. Conf. Knowledge-Based Computer Systems*, 2004.

[Russell and Norvig, 2003] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[Sanner and Yoon, 2011] Scott Sanner and Sungwook Yoon. International Probabilistic Planning Competition (IPPC) 2011. In *ICAPS*, 2011.

[Sanner, 2010] Scott Sanner. Relational Dynamic Influence Diagram Language (RDDL): Language Description. 2010.