# Dissertation Abstract: Exploiting Symmetries in Sequential Decision Making under Uncertainty

**Ankit Anand**
Indian Institute of Technology, Delhi
New Delhi,India-110016
ankit.anand@cse.iitd.ac.in

## Synopsis

The problem of sequential decision making under uncertainty, often modeled as an MDP is an important problem in planning and reinforcement learning communities. Traditional MDP solvers operate in flat state space and don't scale well in large state and action spaces. A lot of real world domains have exponential number of states in terms of representation but many of these states and actions are symmetric to each other. In this work, we focus on exploiting symmetry in these domains to make contemporary algorithms more efficient and scalable. Our recent works ASAP-UCT and OGA-UCT which define new state-action pair symmetries and apply them in UCT show promising initial results of this approach. We study important research questions related to finding and using symmetry based abstractions and discuss interesting links with lifted inference in graphical models.

## Introduction

The problem of sequential decision making under uncertainty, often modeled as a Markov Decision Process (MDP), is a fundamental problem in the design of autonomous agents(Russell and Norvig 2003). Traditional MDP solving algorithms (value iteration and variants) perform offline dynamic programming or linear 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 in these scenarios is through domain abstractions. An interesting aspect which have been observed in many domains of interest is that even though flat state space is very large, many states are symmetric to one other. Existing offline abstraction techniques (Givan, Dean, and Greig 2003; Ravindran and Barto 2004) make use of these symmetries and 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. We intend to study these symmetry exploiting abstractions in traditional MDP setup as well as state of art algorithms which are mostly online, anytime and deal with very large state and action spaces.



Our recent works (Anand et al. 2015b)(Anand et al. 2015a), expands the aforesaid traditional notion of symmetries by giving a novel 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. Moreover, SAP abstractions find symmetries even when there aren't many available state abstractions, which is commonly true for many domains in practice.

During the last decade, 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, Hoey, and Sanner 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.

One of the recent works which correct this limitation is by (Jiang, Singh, and Lewis 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: Abstractions of state in 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 our recent works (Anand et al. 2015a)(Anand et al. 2016), we fill this gap by implementing SAP abstractions inside the UCT framework.

In our recent work (Anand et al. 2015a), we develop an algorithm- ASAP-UCT( **A**bstraction of **S**tate-**A**ction **P**airs in **UCT**) which is a first attempt to exploit SAP abstractions. ASAP-UCT is a batch algorithm like AS-UCT and alternates between two phases. One phase consists of an abstraction computation routine that uses the existing UCT tree to induce groups of symmetric nodes. These nodes are aggregated to construct an abstract search tree. The second phase

is the (modified) UCT algorithm, which is run as per original UCT in the beginning, but is modified to incorporate the abstractions after the abstraction routine has been run at least once. Experiments show that ASAP-UCT significantly outperforms both AS-UCT and vanilla UCT on a number of planning domains obtaining upto 26% performance improvements.

Our further research shows that these batch algorithms do not achieve the full potential of abstractions because of the two disjoint phases. Since abstractions are computed on a sampled tree, they are approximate. Erroneous abstractions computed as part of one batch of abstraction computation may get corrected only after a full phase of modified UCT – this wait could severely impact the solution quality.

In response, we propose *On the Go Abstractions* (OGA), a novel approach in which abstraction computation is tightly integrated into the MCTS algorithm in our recent work (Anand et al. 2016). We implement these on top of UCT and name the resulting algorithm OGA-UCT. It has several desirable properties, including (1) rapid use of new information in modifying existing abstractions, (2) elimination of the expensive batch abstraction computation phase, and (3) focusing abstraction computation on important part of the sampled search space. We experimentally compare OGA-UCT against ASAP-UCT, a recent state-of-the-art MDP algorithm as well as vanilla UCT algorithm. We find that OGA-UCT is robust across a suite of planning competition and other MDP domains, and obtains up to 18 % quality improvements. Based on these initial promising results, we intend to develop a domain independent state of art planner which can benefit from domain abstractions.

Lastly, in the past decade, there has also been revival of interest in abstractions and symmetries with the emergence of lifting techniques in probabilistic graphical models literature. Many of the problems which previously were intractable in probabilistic inference can now be solved by these advances in lifting techniques. Also, there is a strong co-relation between MDP solving methods and probabilistic inference as both of these algorithms depend upon local interactions between neighboring nodes. The ideas of Counting BP (Kersting, Ahmadi, and Natarajan 2009) is very similar to block-splitting algorithm proposed by Givan et. al. Also, homomorphisms (Bui, Huynh, and Riedel 2012) have also been well studied in probabilistic inference literature. Under this theme, we would like to explore the possibility of unifying these two different problems and other related problems under a common abstraction framework so that a generic abstraction approach for solving these can be developed.

## 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 $\gamma$ ($\gamma < 1$). 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 $\pi$ 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 $\pi$ from then on. The optimal policy $\pi^*$ 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 $\gamma = 1$.

### 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 ASAP 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.

### 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 tradeoff, 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).

### 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 works is (Jiang, Singh, and Lewis 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. 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. 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.

## ASAP: Abstraction of State-Action Pairs

In this section, we introduce a new type of State-Action Pair (SAP) abstractions (proposed by us) in addition to previously defined State Abstractions. SAP abstractions are general and can be used independently by any of MDP solving algorithms.

Our **Abstractions of State-Action Pairs** (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 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 $\mu_{\mathcal{E}} : S \rightarrow \mathcal{X}$ 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 $\mu_{\mathcal{H}} : P \rightarrow \mathcal{U}$ denote the corresponding equivalence function mapping state-action pairs to the corresponding equivalence classes. Next, we will recursively define state equivalences over state-pair equivalences and vice-versa.

**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 $\mu_{\mathcal{H}}(s,a) = \mu_{\mathcal{H}}(s',a')$.

**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. $\mu_{\mathcal{H}}(s,a) = \mu_{\mathcal{H}}(s',a')$ if:

- $\forall s_i \in S$ such that $\mathcal{T}(s,a,s_i) = p$, $\exists s'_i \in S, \mu_{\mathcal{E}}(s_i) = \mu_{\mathcal{E}}(s'_i)$ and $\mathcal{T}(s',a',s'_i) = p$. (Condition 1(a))
- $\forall s'_i \in S$ such that $\mathcal{T}(s',a',s'_i) = p$, $\exists s_i \in S, \mu_{\mathcal{E}}(s'_i) = \mu_{\mathcal{E}}(s_i)$ and $\mathcal{T}(s,a,s_i) = p$. (Condition 1(b))
- $C(s,a) = C(s',a')$ (Condition 2)

Intuitively, for state-action pair equivalence to hold, the corresponding states that they transition to should be equivalent and the respective transition probabilities should match. Second condition requires the costs of applying corresponding actions to be identical to each other. For Goal-directed 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 MDPs, all goal states at a given depth are equivalent.

**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. The equivalent AND-OR graph for this domain is the leftmost graph in the figure. Givan's Abstraction of States (AS) conditions check for exact action equivalence. They will observe that S2 and S3 are redundant players and merge the two states. Ravindran's Abstraction of States with Action Mapping (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.

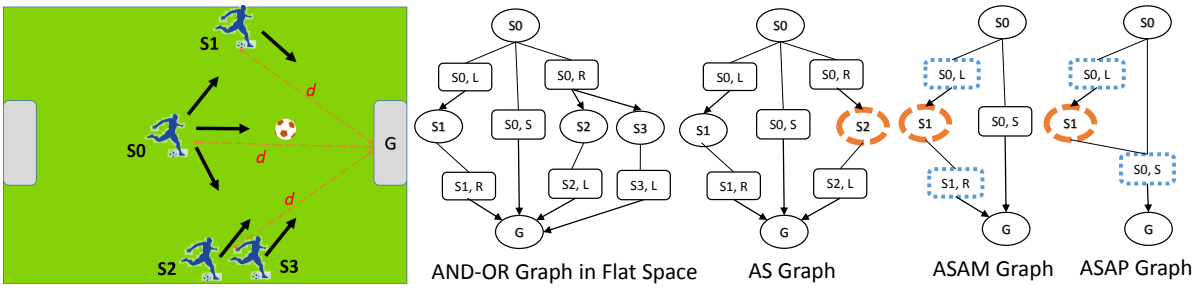


Figure 1: An example showing abstractions generated by various algorithms on a soccer domain. Givan's AS, Ravindran's ASAM and our ASAP frameworks successively discover more and more symmetries.

Overall, ASAP will identify the maximum symmetries in the problem. Next, we state theoretical results corresponding to ASAP.

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

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

## ASAP Symmetries in UCT

We next describe algorithms to incorporate ASAP framework in UCT. Firstly, we describe a batch algorithm ASAP-UCT which is followed by OGA-UCT. OGA-UCT builds on ASAP-UCT by computing abstractions on the go as we are building the tree. We show empirically that ASAP-UCT performs better than AS-UCT and ASAM-UCT and further illustrate that OGA-UCT outperforms ASAP-UCT on several domains of interest.

### ASAP-UCT

ASAP-UCT is a UCT-based algorithm that uses the power of abstractions computed via the ASAP framework. Recall that since UCT constructs a finite-horizon MDP tree, states at different depths have to be treated differently. Therefore, ASAP-UCT tests state equivalences for states at the same depth only. To compute abstractions over UCT tree, we adapt and extend ideas in Jiang et al. (2014)'s work.

**Computing UCT Abstractions:** ASAP-UCT computes abstractions in a bottom up fashion starting with the leaves and successively computing abstractions at each level (depth) all the way to the root (Algorithm 1). It takes as input a UCT Search Tree (ST) and outputs an Abstracted Search Tree (AST). At each level, it calls the functions for computing state and state-action pair abstractions, alternately.

For the pseudo-code 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 the superscript $d$ to denote the state (SAP) pair equivalence function $\mu^d_{\mathcal{E}}$ ($\mu^d_{\mathcal{H}}$) 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 (similarly for $\mu^d_{\mathcal{H}}$). $ComputeAS$ and $ComputeASAP$ at each level operate as per the definitions described in ASAP framework. As we are going bottom up, the abstractions at below level have already been computed.

**Algorithm 1** Computing Abstracted Search Tree

**ComputeAbstractedSearchTree**(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
  **return** $AST$

**Updating Q-Values:** Once the nodes at a level have been made a part of the same abstract state, we maintain an estimate of the expected cost (to reach the goal state) for the abstract node only (both in the state layer as well as in the state-action layer). The abstract node is initialized with the average of the expected cost of its constituent in the beginning. Any future Q-value updates are performed over the abstracted out representation.

**When to Compute Abstractions:** Since we need the current sampled tree for calculating the abstractions, abstraction can be computed only after the tree has been constructed to a certain level. But if we wait until the full expansion of the tree (i.e. end of the planning phase), the abstractions would not be useful. We compute abstractions for a fixed number of times $l$ in each decision. After every abstraction, the Q-values are computed on the abstract tree. Future expansions might invalidate the abstractions computed earlier. We correct for this by performing the next phase of abstractions from scratch on the flat (unabstracted) tree. In summary, the algorithm can be described as a batch algorithm which interleaves expansions, Q-value computations and abstraction steps.

**Experimental Results with ASAP-UCT** 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. Figures 2 shows the comparisons across two domains. Note that time taken for a trial also includes the time taken to compute the abstractions. In almost

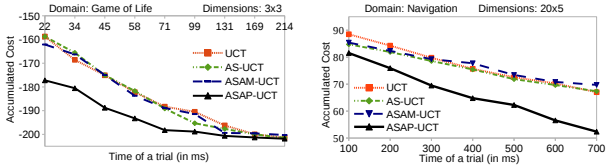


Figure 2: ASAP-UCT outperforms all other algorithms on problems from three domains.

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 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.

## OGA-UCT

Here, we describe OGA-UCT, an On the Go Abstraction algorithm which computes abstractions as we are building the tree. Our algorithm is best understood in terms of the construction of the original UCT tree. The UCT computation can be broadly divided in three phases 1) Sampling of a trajectory 2) Random rollout from a newly discovered leaf node 3) Back up of Q-values. In OGA-UCT, during the first phase, along with sampling of the trajectory, an abstraction for each state is also maintained *on the go*. Abstraction for any node is computed using the recursive updates similar to the ones used by ASAP-UCT. But the key difference is that instead of doing the batch computation uniformly for each node, we do it incrementally and in an adaptive manner. Each node has an associated *r*ecency count which stores the number of times the node was visited after its abstraction was last updated. If the recency count reaches a pre-decided threshold $K$, we re-compute the abstraction for this node and set the recency count back to 0. In the second phase when a rollout is performed, we initialize the abstraction of the newly created leaf and set its recency count to 0. Any $Q$-value updates in the UCT tree are now done over the abstract nodes rather than the original nodes. Since abstractions at a certain depth depend on the abstractions in the tree below, it may happen that when a node's abstraction changes, there could be a change in the abstraction of its ancestor nodes. Therefore, any change in the abstraction of a node at depth $d$, is propagated all the way up to the root of the tree, recomputing abstractions as necessary. We describe the Sampling Trajectory Procedure 2 in detail here.

**Sampling Trajectory (Algorithm 2)**: This is the main procedure of our algorithm. Lines 1-4 check the base condition for stopping a trajectory. Lines 7-11 add a newly discovered leaf node to the tree, initialize its abstraction and perform a rollout. If the procedure comes to line 12, we have not discovered a new leaf node yet. Line 12 selects an action based on the UCB rule. In lines 13-14, we add a newly discovered SAP node to the tree and initialize its abstraction. Lines 18-19 sample a new state node based on the chosen action and recursively call SAMPLETRAJECTORY. Lines 19-23 take care of maintaining the recency count and calling update abstractions if the count has reached the threshold $K$. Here, Update SAP abstraction updates the abstractions with respect to the state abstractions at the next level. Also, if the abstraction of SAP node changes, this change calls Update State Abstractions for the parent node and this update procedure is recursively repeated till the root if the abstractions changes. Finally, the UCT counts and Q values are updated in lines 24-26. It is insightful to note that if we remove the lines for computing abstractions and maintaining the recency count (lines 9,15-16,20-23), the procedure becomes identical to what standard UCT would do.

**Algorithm 2** Sample Trajectory in UCT

```
1:  procedure VAL = SAMPLETRAJECTORY(s, d)
2:      if terminal(s) then
3:          return −reward(s)
4:      else if d == Horizon then
5:          return 0
6:      end if
7:      if (s, d) is not in tree T then
8:          Add state node (s, d) to tree T
9:          INITIALIZESTATEABSTRACTION(s, d)
10:         return GETROLLOUT(s, d)
11:     end if
12:     a ← SELECT-UCB-ACTION(s, d)
13:     if (s, a, d) is not in tree T then
14:         Add SAP node (s, a, d) to tree T
15:         INITIALIZE-SAP-ABSTRACTION(s, a, d)
16:         RecencyCount[s, a, d] ← 0
17:     end if
18:     s' ← SAMPLE(s, a)
19:     RecencyCount[s, a, d] + +
20:     newVal ← SAMPLETRAJECTORY(s', d + 1)
21:     if RecencyCount[s, a, d] == K then
22:         UPDATE-SAP-ABSTRACTION(s, a, d)
23:     end if
24:     INCREMENTCOUNT(s, a, d)
25:     UPDATEQ((s, a, d), newVal)
26:     return newVal
27: end procedure
```

It is also important to note that OGA-UCT converges to correct Q-values as computed by UCT given sufficiently large amount of time.

**Theorem 3.** *Given an MDP $M = (S, A, \mathcal{T}, C, H)$, the value function computed by OGA-UCT for the abstract node containing a state $s$ at depth $d$ converges to the value function computed by UCT for state $s$, as number of trajectories $N \to \infty$ i.e $\forall s \in S \;\; \forall d \leq Horizon$*

$$\lim_{N\to\infty} V^N_{OGA}(\mu^d_{\mathcal{X}}(s), d) = \lim_{N\to\infty} V^N_{UCT}(s, d)$$

**Empirical Results of OGA-UCT** We compare OGA-UCT with ASAP-UCT and unabstracted UCT on these problems with different total planning times and draw cost vs. time curves. Representative runs on two domains are illustrated in Figure 3. Each curve is an average of 1,000 reruns and 95% confidence interval bars are also drawn.

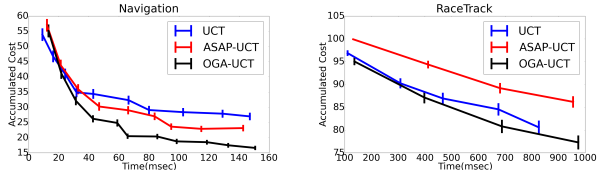


Figure 3: OGA-UCT performs better or at par with ASAP-UCT and UCT for most of the domains

We observe that OGA-UCT performs the best or on par with the best on four out of the five domains. These results demonstrate that difference in the performances of ASAP-UCT and UCT can depend heavily on the domain, but OGA-UCT admits least variance and is robust across these many domains.

## Proposed Scope and Future Focus

Abstraction and symmetry in MDPs in itself is a relatively old field with rich theoretical literature on it. This work has assumed great importance in today's world with the need for real time MDP solvers for large problem instances. The advent of space exploration missions like Mars rover is a perfect example of reinforcement learning problem where hard multiple objectives need to be achieved with in constraints of time, cost and safety. We plan to extend these abstraction frameworks and adapt these in complete end to end systems which can be used in real world. To achieve this, we will focus on some or all of these problems.

- **A Domain Independent Abstraction Based Planner:** Initial investigations of applying abstractions in UCT have shown promising results both in OGA-UCT and ASAP-UCT. Presently both OGA-UCT and ASAP-UCT operate in flat state space, how to modify and use the factored representations is an important step in the development of such a planner.
- **Learning Abstractions:** With advances in machine learning techniques, an interesting approach to compute abstractions is by learning the symmetries of state space. A recent work by Srinivasan et. al. (Srinivasan, Talvitie, and Bowling 2015) suggests the use of nearest neighbor approach to improve exploration in UCT. Learning abstractions is an important problem to be studied in context of online algorithms where abstraction computation overhead may become a bottleneck.
- **Relation between Lifted inference and Planning** As pointed out earlier, symmetries have played a significant role in advancing inference techniques in graphical models. Due to local nature of computation, there is significant overlap of techniques used to exploit symmetries in both planning and graphical models. We intend to study this co-relation in detail and wish to develop a generic abstraction framework for both these fields.

Finally, we believe that exploiting symmetry based abstractions could lead to significant improvements in many algorithms not limited to planning and reinforcement learning. Our initial investigations with it and development of ASAP-UCT and OGA-UCT clearly show the first step in this direction. Nevertheless, there are significant challenges like computing symmetries efficiently, operating in factored state space and adapting comtemporary algorithms to compute symmetries which are non-trivial and need to be investigated thoroughly.

## References


Anand, A.; Grover, A.; Mausam; and Singla, P. 2015a. ASAP-UCT: Abstraction of State-Action Pairs in UCT. In *IJCAI*, 1509–1515.

Anand, A.; Grover, A.; Mausam; and Singla, P. 2015b. A Novel Abstraction Framework for Online Planning. In *AAMAS*.

Anand, A.; Noothigattu, R.; Mausam; and Singla, P. 2016. OGA-UCT: On-the-Go Abstractions in UCT. In *ICAPS*.

Bellman, R. 1957. A Markovian Decision Process. *Indiana University Mathematics Journal*.

Bui, H. H.; Huynh, T. N.; and Riedel, S. 2012. Automorphism groups of graphical models and lifted variational inference. *CoRR* abs/1207.4814.

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

Givan, R.; Dean, T.; and Greig, M. 2003. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence* 147(1–2):163 – 223.

Grzes, M.; Hoey, J.; and Sanner, S. 2014. International Probabilistic Planning Competition (IPPC) 2014. In *ICAPS*.

Hostetler, J.; Fern, A.; and Dietterich, T. 2014. State Aggregation in Monte Carlo Tree Search. In *AAAI*.

Jiang, N.; Singh, S.; and Lewis, R. 2014. Improving UCT Planning via Approximate Homomorphisms. In *AAMAS*.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic Planning Based on UCT. In *ICAPS*.

Kersting, K.; Ahmadi, B.; and Natarajan, S. 2009. Counting Belief Propagation. In *UAI*, UAI '09, 277–284. Arlington, Virginia, United States: AUAI Press.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML*. Springer.

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

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

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

Russell, S. J., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

Sanner, S., and Yoon, S. 2011. International Probabilistic Planning Competition (IPPC) 2011. In *ICAPS*.

Srinivasan, S.; Talvitie, E.; and Bowling, M. 2015. Improving Exploration in UCT Using Local Manifolds. In *AAAI Conference on Artificial Intelligence*.