Efficient Exploration in Monte Carlo Tree Search using Human Action Abstractions

Kaushik Subramanian
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332
ksubrama@cc.gatech.edu

Jonathan Scholz
Google DeepMind
London, United Kingdom
jonathan.scholz@gmail.com

Charles L. Isbell
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332
isbell@cc.gatech.edu

Andrea L. Thomaz
Electrical and Computer Engineering
University of Texas at Austin
Austin, Texas 78705
athomaz@ece.utexas.edu

Abstract

Monte Carlo Tree Search (MCTS) is a family of methods for planning in large domains. It focuses on finding a good action for a particular state, making its complexity independent of the size of the state space. However such methods are exponential with respect to the branching factor. Effective application of MCTS requires good heuristics to arbitrate action selection during learning. In this paper we present a policy-guided approach that utilizes action abstractions, derived from human input, with MCTS to facilitate efficient exploration. We draw from existing work in hierarchical reinforcement learning, interactive machine learning and show how multi-step actions, represented as stochastic policies, can serve as good action selection heuristics. We demonstrate the efficacy of our approach in the PacMan domain and highlight its advantages over traditional MCTS.

1 Introduction

Monte Carlo Tree Search (MCTS) [5] algorithms have been used to address problems with large state spaces. They focus on solving the policy for a single state—the state the agent is in—making the planning time independent of the total number of the states. MCTS covers a family of algorithms including Sparse Sampling [16] and its successors, UCT [17] and FSSS [31]. It has grown rapidly in visibility due to its early successes in the boardgame Go and by winning AAAI's General Game Playing competitions [11, 10]. More recently, with the growth of deep learning research [14], MCTS methods have been combined with function approximation using deep learning to achieve state-of-the-art performance in Atari games [13] and Go [26]. These results have highlighted the use of MCTS for planning in large domains.

The successes however have their share of costs. Tree search methods are, in general, exponential in their depth, with a branching factor that depends on the number of possible actions and subsequent states at each node. Thus to make MCTS effective requires the use of heuristics that help action selection during tree search and roll-out execution. Existing methods (UCT, FSSS) utilize confidence bounds on the value function [3], by tracking state-action pair visitations, to decide which actions to explore and exploit. These methods are sample intensive and pay a substantial computational cost for every step of action selection.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

In parallel, researchers have focused on how to leverage human help to improve learning and planning, including work in learning by demonstration [2], imitation learning [1], and interactive machine learning [7]. The motivation for these works stems from the observation that (1) human help is often available, and (2) humans excel at some important tasks that automated methods have difficulty with. Application of human input has yielded promising results such as helicopter flying [21], teaching a AIBO robot basic soccer skills [12] and played an important role in the success of AlphaGo [26]. Of particular importance in this work is the ability of humans to help autonomous agents explore promising parts of the state space [9] and the use of human input to construct action abstractions that can decompose complex problems in simpler subparts [32].

In this paper, we show how we can leverage recent work in utilizing action abstractions for reinforcement learning [29] to help satisfy the requirements of MCTS without incurring the expensive computational costs. Action abstractions like Options [28] and Constraints [15] represent multi-step policies that can significantly speed up planning in reinforcement learning. This form of knowledge allows the agent to lookahead over multiple timesteps, obtain better estimates of the utility of an action and propagate this information to multiple states. We note the use of constraints as actions While options and constraints are similar at a high-level, these abstractions differ in their respective definitions and tackle different aspects of the problem. Options represent abstractions that capture *goals to achieve* in a task while constraints capture *situations to avoid*. These ideas have been successfully combined and utilized in Q-learning [25] to solve gridworld domains. To our knowledge, this is the first method that utilizes constraints as actions abstractions for MCTS. In this work we characterize specific properties of these action abstractions and show how they can used as action pruning heuristics and high quality roll-out policies for MCTS to solve large problems. In particular, we show that:

- Options offer coherent, near-optimal action sequences for solving sub-tasks. They allow us to increase the effective search depth of MCTS methods.
- Constraints complement options by identifying actions not to follow. They can act as both a form of pruning and a way to encapsulate an intelligent roll-out policy.

We leverage these properties to develop a novel approach, Policy-Guided Sparse Sampling (PGSS), that can effectively use such abstractions to overcome some of its limitations and plan efficiently. Using the PacMan domain, we show how PGSS satisfies the requirements for efficient exploration in MCTS.

2 Related Work

The state of the art techniques in MCTS [5] include SS [16], UCT [17], its variants and FSSS [31]. They provide different ways of performing action selection in MCTS. The respective exploration strategies depend on good Q-value estimates which are often time-consuming and hard to obtain. The results also depend heavily on the choice of parameters used for learning (number of sampled trajectories and branching depth). We seek to circumvent this problem by directly incorporating domain knowledge in the form of action abstractions to bias action selection. Recently MCTS has been combined with deep learning methods [14] to facilitate function approximation in large game domains [13, 26]. While these methods are able to leverage the generalization capabilities of deep networks to generate state-of-the-art performance, they require large amounts of training data to learn the parameters of deep neural network models. We argue for the incorporation of domain knowledge to help exploration in MCTS without significant computational costs. We note that there has been prior work on using MCTS with expert knowledge [8]. This approach focuses on using human knowledge of the boardgame Go to define a comprehensive set of rules that help in directing exploration of the tree. While the idea behind this approach is similar to ours, their implementation encodes expert knowledge as part of the computations that measure value confidence bounds and requires careful tuning of several coefficients which sometimes result in conflicting learning objectives.

Action abstractions like Options [28] were introduced in the hierarchical reinforcement learning literature as a principled approach to learning from temporally extended actions. They instantiate policies which represent different sub-tasks for a problem and use them to accelerate planning. Constraints introduced more recently [15] instantiate policies that capture negative outcomes in a domain, by looking over multiple timesteps, and use that information to guide action selection for

the agent. Guliz and Feigh [30] were able to show that humans solving problems (specifically game domains) by using these action abstractions. These approaches have been used to solve problems independently [22] and together [25].

There have been other methods introduced in the literature that have approached the problem of combining different forms of action abstraction. One approach is the Concurrent Actions Model [24, 23] which formally describes a framework where an agent plans over concurrent temporally extended actions. These actions have different kinds of termination schemes which are similar to abstractions used in our approach. In their work, they highlight that the bottleneck for their approach is an efficient way of searching through the space of multi-actions that can be run in parallel. Our PGSS algorithm aims to solve exactly that problem. [19] constructs different types of skills and uses Q-learning to learn domain specific skill combinations. We note that in their work it is not clear how non-terminating skills can be utilized in the Q-learning framework. Overall while our approach has the same motivation as theirs, the intelligent use of the action abstractions in MCTS helps to overcome the exploration complexities of Q-learning. Recent work closely related to ideas presented in this paper [4] use options as action abstractions in MCTS to solve partially observable MDPs. While their method does not utilize constraints as action abstractions, they show advantages of temporal actions for MCTS planning.

Taking advantage of the action abstractions as domain knowledge includes the cost of defining them for the problems we would like to solve. There are several methods that aim to solve the problem of instantiating options [6, 27, 18, 32, 20] and constraints [15] either automatically or using human input. We add that devising automated ways of instantiating these abstractions is not the main focus of our work. We will show in our experiments that our incorporation of domain knowledge in into MCTS achieves compelling gains over complex problems that potentially offset the initial computations spent in instantiation.

3 Background and Terminology

We formulate our approach using the Reinforcement Learning (RL) [29] framework. An RL agent interacts with an environment described by a Markov Decision Process (MDP), a tuple $M = \langle S, A, \mathcal{P}, R, \gamma \rangle$ with states S, actions A, transition function $\mathcal{P} : S \times A \mapsto \Pr[S]$, reward function $R : S \times A \mapsto [R_{min}, R_{max}]$, and discount factor $\gamma \mapsto [0, 1]$. A policy $\pi : S \mapsto A$ is a relation that defines which action should be taken in a particular state. For a given MDP, a Q-function $Q(s, a)$ represents the *expected long-term reward* of taking action a in state s, and following the optimal policy thereafter. This function is all an agent needs to know to act optimally in an MDP, and is the quantity that MCTS algorithms attempt to estimate. We describe MCTS and specific properties of options and constraints as action abstractions in more detail in the context of our proposed approach.

4 Approach

Here we discuss properties of Monte Carlo Tree Search (MCTS) for action-value estimation, and our method of improving it with auxiliary information in the form of action abstractions.

4.1 Monte-Carlo Tree Search

Monte Carlo Tree Search is a general approach to MDP planning which uses online Monte-Carlo simulation to estimate action (Q) values. The basic observation behind MCTS algorithms is that for MDPs with $\gamma < 1$, there is an effective horizon H beyond which rewards do not significantly affect the optimal policy for the agent's *current state*. This places a theoretical (though perhaps still intractable) bound on the number of steps that must be considered to accurately estimate the Q-values of the current state.

MCTS algorithms perform a forward search from the current state, selecting and branching on actions and possible transitions from $P(s'|s, a)$, out to some depth d. From this search, we can estimate the d-horizon Q-values:

$$Q^d(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q^{d-1}(s', a') \tag{1}$$

where $Q^1(s,a) = R(s,a)$.

Note that Eq. 1 requires iterating over both the set of actions and possible transitions in the MDP. The number of possible transitions defined by $P(s'|s,a)$ is $|S|$, the total number of states; however, Kearns et al. 2002 showed that it is possible to obtain ϵ-optimal Q-value estimates for the current state from a set of sampled transitions, and that the number of samples C per state was independent of $|S|$.

Unfortunately, MCTS remains exponential in the *depth* of the tree. The sample complexity of uninformed MCTS is then $O(|A| * C)^H$ [16], corresponding to a depth-H tree. To address the exponential blow-up in H, practical MCTS implementations must typically truncate the tree expansion at some depth or time threshold, and approximate the values of the leaf nodes by evaluating a fixed (possibly random) "roll-out" policy. There are therefore two opportunities to optimize generic MCTS: introduce biases during the tree search, and during the roll-out policy. The bulk of the work in MCTS, including ours, focuses on what kind of information would be most useful to generate this bias.

Algorithms such as Upper-Confidence Trees (UCT) [17] and Forward Search Sparse Sampling (FSSS) [31] attempt to generate the bias by using relative Q-values. The intuition is that if we had high confidence in $Q(s,a_1) > Q(s,a_2)$ for two actions a_1, a_2, obtaining further samples from s, a_2 would be wasteful: they cannot change the Q-value of s. From this we see that the best case search policy is in fact the optimal policy π^*, as it wastes no samples on sub-optimal trajectories. The search policy can have a significant role in the complexity of MCTS: with an optimal policy, the required number of samples for an accurate Q-estimate is closer to $C * H$ than $(|A| * C)^H$.

4.2 Policy-Guided Sparse Sampling

As discussed in Section 4.1, the key property in determining the efficiency of MCTS is the implicit tree-search policy of the algorithm. Non-interactive approaches to designing this search bias are value-based, and require the agent to visit states multiple times in order to compute the relevant statistics for directing future search. This requirement can be intractable for a number of reasons, including a high branching factor, strict realtime deadlines, a γ close to 1, or a transition model that is expensive to query (*e.g.,* requires running a physics simulation). This motivates the main idea behind Policy-Guided Sparse Sampling (PGSS): to construct the search policy explicitly by using a combination of different action abstractions with desirable properties.

4.2.1 Action Abstractions

We noted in Section 4.1 that MCTS presents two points for biasing action selection: 1) during tree search and 2) during roll-out. This suggests the use of two different and complementary policy classes: options and constraints. The use of these policies in MCTS is highlighted in Figure 1.

Options. The first policy class will serve to augment the set of primitive actions, allowing deeper look-ahead in the tree. Following [28], an option is a sub-policy with clearly defined initiation and termination conditions, and is generally used to encapsulate sub-tasks in a planning problem. Options allow the planner to make large jumps in the state space: assuming the options' policies are locally optimal for their subtask, searching at the level of options increases the effective branching depth of the planner by a factor of d_o, where d_o is the expected length of the option.

Constraints. The second type of abstraction [15] encodes a bias to *disallow* certain actions, and has two modes of operation: (1) as a action-pruning heuristic during tree expansion, and (2) as a roll-out policy for obtaining value estimates for the leaf nodes. In an uninformed implementation of MCTS, the roll-out policy is a random policy, significantly underestimating the actual value of leaf nodes. By comparison, a policy that avoids terminal states where one cannot escape negative reward will generally provide a better estimate of the value of the leaf nodes. We refer to a policy designed to achieve this survivor effect as a *constraint*, to indicate that it restricts the agent from executing actions that result in terminal states. A constraint is represented by a policy that satisfies its conditions, along with an initiation set that indicates when the policy of the constraint should be taken into account. We find the constraint policy to be useful not only for biasing action selection during the tree search, but also as a self-contained roll-out policy. As we discuss in the next section, this provides a soft form of tree pruning to remove branches unlikely to lead to high value states.

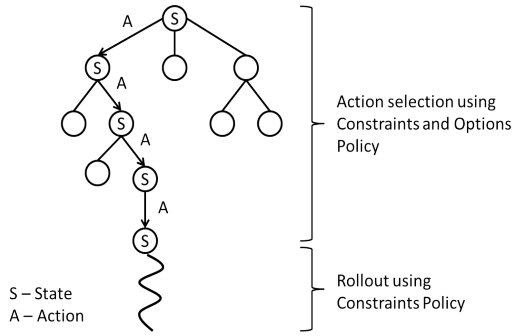


Figure 1: Monte Carlo Tree Search highlighting where we use both constraints and options for effective exploration.

Algorithm 1: Policy-Guided Sparse Sampling

PGSS(s, d, O)
if $d = H$ **then** return 0
end if
if $O = \emptyset$ **OR** $T_O(s) > rand$ **then**
    % Sample an available option
    $O \sim \{O : I_O(s) = true\}$
end if
if $O \neq \emptyset$ **AND** $d < d_{max}$ **then**
    % Sample from constrained option
    $a \sim P(a|s) \propto P_{\pi_O}(a|s) \times \frac{P_{\pi_c}(a|s)^\beta}{\sum_{a \in A} P_{\pi_c}(a|s)^\beta}$
else
    % Sample directly from constraint
    $a \sim \frac{P_{\pi_c}(a|s)^\beta}{\sum_{a \in A} P_{\pi_c}(a|s)^\beta}$
end if
$s' \sim P(s'|s, a)$
$Q_{ss}(s, a) = R(s, a) + \gamma PGSS(s', d + 1, O)$
return $\max_{a \in A} Q_{ss}(s, a)$

4.2.2 Policies as Heuristics

When considered within MCTS, options and constraints provide ideal heuristics to help bias tree search, allowing for deeper or more accurate value estimation. We first show how to incorporate constraint policies. Because a constraint explicitly represents the permissible actions for all states, it can be used for pruning at each node. In order to prevent constraints from filtering out optimal actions, and thereby removing the theoretical guarantees of MCTS, we apply the softmax operation using an auxiliary β parameter to define a probability distribution over actions for each state:

$$P(a|s) = \frac{P_{\pi_c}(a|s)^\beta}{\sum_{a \in A} P_{\pi_c}(a|s)^\beta} \tag{2}$$

π_c represents the constraint policy and β controls how peaked the distribution is over the preferred action, controlling how much to "trust" the constraint. Note that constraints are represented as typical policies, but encode a preference for "safe" actions, with entropy proportional to β. By incorporating the soft-maxed constraint, we can achieve an arbitrarily safe union of policies.

For options, we first review the basic theory of offline planning with options (*e.g.*, value iteration) [28]. An option is defined as a tuple $< I, T, \pi >$ representing the set of states I_o where the option can be initiated, a distribution T_o over states for terminating the option, and the option policy π_o itself. Traditional approaches are based on Bellman-updates over primitive actions, so planning with options requires an expected reward and terminal state for each option.

$$E[R|\pi_o(s)], E[s'|\pi_o(s)] \tag{3}$$

We can extend MCTS to incorporate options by adding them as additional actions to all states in their respective initiation sets I_O, and terminate them during each step according to their respective termination probabilities $T_O(s)$. In this way, MCTS performs the option evaluation. With the expected length of the option counting towards the total depth reached by the agent, options are serving to bias search towards specific trajectories that we have *a priori* reason to believe are useful.[1]

Here we emphasize the need for options and constrains to be handled differently. In our formulation, an option always suggests an action to take while a constraint rarely prefers an action to take unless the agent is about to enter a dire circumstance. More specifically, the use of constraints at the leaves of the tree keeps the agent "alive" by avoiding low expected utility and out-performs options at that task. Similarly options drive one towards goals and out-perform constraints at those tasks. This characteristic makes them qualitatively different and therefore should be managed differently in order to exploit their unique properties.

[1]The availability of the constraint puts a minor modification on the option's roll-out: since the constraint can preempt the option, we're actually taking samples of a hybrid option+constraint policy for each option

Algorithm 1 is our approach to Policy-Guided Sparse Sampling. The algorithm recursively constructs a search tree to branching depth d_{max}, and performs constraint policy roll-outs to the horizon H. π_c is the constraint policy, π_O is an option policy, $I_O(s)$ returns true if option O can be initiated from state s, and $T_O(s)$ is the probability of terminating option O in state s. Our implementation branches over primitive actions only when there are no valid options for the current state. This was a reasonable restriction for our experiments, since the options fully covered the set of appropriate actions for all time-steps. However, in general we would typically branch over primitives as well.

4.2.3 Combining Multiple Constraints

In domains where multiple constraints are required to be satisfied, they can be combined in a straightforward manner. For any given state s we create a list of the constraints that are activated there and then generate a set by taking a union of all the actions the constraints suggest to take. We then reweigh the probabilities of this action set according to the outcome of disobeying each individual actions suggested by the respective constraints. We now have a stochastic distribution over actions that takes into account information of multiple constraints. We draw from it and proceed down the tree to the next node. More details are available in [15].

5 Experiments

In this section we present empirical evaluation of our approach by instantiating it on the PacMan[2] domain. PacMan naturally lends itself to be abstracted by hierarchical decompositions and is a domain which poses difficulties for tree search methods due to its long horizon. For example, in our experiments, a 25x25 grid with four ghosts and four power pellets has a total of over 10^{15} states with an effective depth of 340 steps. We implemented the necessary abstractions using human interaction.

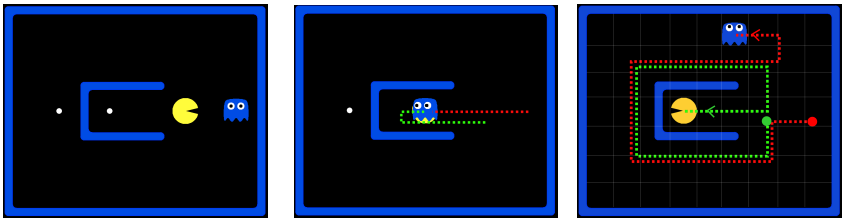
5.1 Information From Humans

In Tokadli and Feigh [2015], the authors describe useful action abstractions for the PacMan domain and motivate how humans naturally provide this information when interacting with the domain. Using this work as motivation, we leverage existing interactive learning methods to learn options [27] and constraints [15]. These approaches use human input in the form of demonstrations to efficiently learn probabilistic policies that define the necessary heuristics.

In our tests, we learn the heuristics from human interaction and refer to existing work [30] to confirm their utility for learning to solve PacMan. As a result of this, we learned the options *eatFood* and *eatCapsule*, and the constraint *avoidGhost* (avoids the nearest one).

We first describe a simple experiment that illustrates the advantages of using a policy biased approach in MCTS and then show how our approach scales to problems of increased horizon depths.

5.2 The Dead-End Experiment



(a) Start (R = 0) (b) Without Constraint (R = -495) (c) With Constraint (R = 497)

Figure 2: Starting map configurations for the dead-end problem (left), terminal state for flat MCTS agent (middle), and optimal solution discovered by PGSS (right). Total reward shown in parentheses

[2]The version of PacMan we used is an open-source implementation available online at http://www-inst.eecs.berkeley.edu/ cs188/pacman/pacman.html

The dead-end experiment is a simple problem designed to provide intuition about the utility of constraint policies in the context of Monte Carlo search. By explicitly asking the question "is this leaf node a state that I can survive in?", the constraint gives the agent a significant advantage in look-ahead. In particular, a constraint policy provides a *more optimistic lower bound* than a random policy for the values of leaf nodes in the search tree. We used a small PacMan grid shown in Figure 2(a) with an effective horizon depth of 18 steps. The ghosts move directionally towards the agent.

As Figures 2(a)-2(c) show, a flat MCTS agent sees the nearest food and goes for it, not realizing that it's a dead-end. By doing an *avoidGhost* roll-out from this state, the constraint agent discovers that it is eventually terminal, backs up that reward to the start state, and chooses to go around instead. When using a random rollout policy, the agent is unlikely to escape the ghost regardless of whether Pacman is trapped. Therefore this agent is less capable of discriminating between the trap and the open space, and is more likely to make the wrong choice.

We note that the inclusion of options as actions that the agent can branch over is a significant advantage as it enables deeper lookahead during rollouts. Overall the PGSS agent can rollout the *eatFood* option policy to obtain reward from the food pellet and at the same time use the constraint to avoid the ghost. This combination allows PGSS to perform optimally using very small search depths.

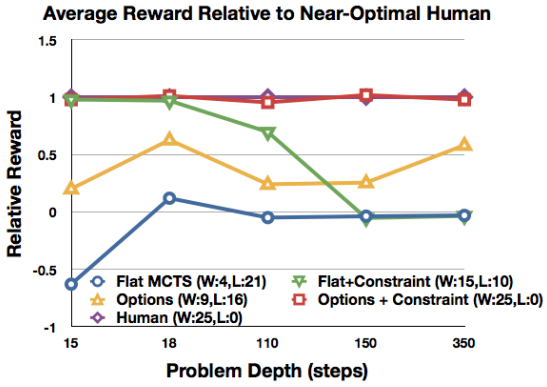
5.3 Scalability

In this experiment we investigate 1) how action abstractions compare to each other and 2) their performance on problems of increasing horizons. We achieve this by implementing several policy-based variants of the PGSS agent in PacMan domains of different sizes. We note that by increasing the size, the effective horizon increases making it significantly harder for MCTS algorithms. We use four variants of PGSS agents. The original sparse-sampling algorithm which branches only over primitive actions, as well as three policy-guided variants: using only options, using only constraints, and using both. We also compare the performance of these agents with that of an average human player. We show the results of this experiment in Figure 3(a). The average rewards were computed over 5 trials. We limited search depth to 34 steps, after which we evaluated the constraint as a rollout policy 3 times. Inside the constraint the Boltzmann temperature value was 10. In these experiments the ghost directions were random. The agents' decisions were made in real-time.

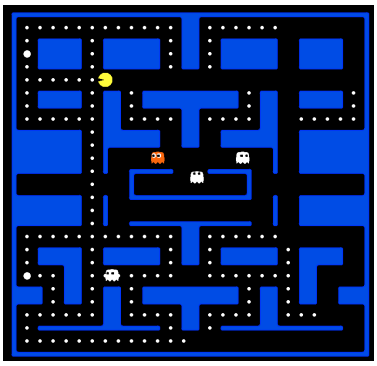
Unsurprisingly, the flat agent proved to be the worst in terms of reward, outcome (win/lose), and runtime. While a small look-ahead is sufficient to win for tiny domains, we found that larger maps required a tree depth that was prohibitively expensive to compute (due to the physics engine). Adding the options extended the effective look-ahead, and significantly increased the average reward per episode on larger maps; however, options also frequently led to bad terminal states, and this agent eventually died in 4 out of 5 trials on the largest map.

Replacing the options with the constraint meant the agent was less likely to die prematurely, but sacrificed the look-ahead depth of options. While this agent performed well in smaller maps, it frequently became disconnected from regions of reward (food) in larger maps, and wandered randomly until trapped or chased away by a ghost. As shown before, by reflecting whether the agent can stay alive from the leaf nodes of tree, the constraint is essentially a dead-end detection mechanism. Using only the constraint, we observed that the agent ate all the food in a neighborhood and then couldn't "see" outside the sample horizon of the constraint and so wandered randomly. Eventually a ghost would either chase him towards a good region or, especially in the big maps, a dead-end.

Fortunately, the strengths and weaknesses of our constraint and options agents are complementary: the options roll-outs find deep action trajectories that are likely to be good, and the constraints help ensure that they do not lead to undesirable states. We found the options+constraints agent to be the superior policy across all problem sizes in terms of speed, total reward, and final outcome. Taking a closer look at these episodes, it seemed that the primary motif this agent excelled at, as compared to the others, was eating ghosts. Ghosts can only be eaten for narrow windows after eating a power pellet, and it typically requires a long and specific sequence of actions to achieve this result. The probability of an uninformed search discovering this full trajectory by chance was too low to observe for ghosts more than a couple steps away from PacMan. In addition to achieving the best reward, the options+constraint agent produced the only policy that could reliably beat the largest map with a effective horizon depth of 350 steps. (Figure 3(a)).



(a) Relative Reward



(b) Episode on Largest Map

Figure 3: Average reward obtained per trial versus map size for different configurations (left), and a sample run on the largest map (right). The numbers in brackets indicate the win/loss ratio.

We have also tested the PGSS algorithm on other related domains (for example Cat and Mouse) that lend themselves to action abstractions and were able to achieve similar results.

6 Discussion and Conclusion

Our experiments yield insights about the use of human-derived action abstractions in MCTS and we highlight them here. An interesting observation is that when sampling the constraint policy, it is possible for us to reach the goal state. In these cases, the computed value for constraint evaluation will be more informative as it includes information about the reward at the goal state. The effect of using such constraints is that it allows us to learn a good policy with a smaller tree depth. We note that this might not be true in all scenarios; however when constructing constraints for a domain, we believe that knowledge of constraints potentially reaching the goal can be utilized to perform more efficient planning.

We view the applicability of PGSS as a way of addressing the class of MDPs in which not only is it intractable to compute a policy for the entire state space, but even for a single state. In Section 4.2 we explained that modern MCTS algorithms like UCT and FSSS assume the agent can afford to explore certain parts of the space quite extensively. In fact, FSSS only terminates after closing all nodes in its search tree, which requires visiting every possible state-action transition out to the problem horizon H. This implies that the time required by FSSS to return an action for the current state is *exponential* in the problem depth. Clearly there are many MDPs in which this is infeasible, such as in our PacMan results from Figure 3(a). These were obtained in *real-time*, which was only possible by shifting to policy-based heuristic that relaxed the need to explore the search-tree exhaustively.

In our tests on instantiating action abstractions, we find that interactive learning approaches provide abstractions more suited for PGSS than autonomous learning methods. We also note that incorporating action abstractions in MCTS as in PGSS provides a general framework that is applicable to other variants of MCTS as well (UCT, FSSS). These methods would only stand to gain performance speed-ups from the use of domain heuristics in the form of temporally extended actions.

In this paper we have described the compatibility between action abstractions learned from humans and the requirements of MCTS. We presented a unifying framework that combines two different kinds of action abstractions and used them as pruning heuristics and intelligent roll-out policies in MCTS. Our experiments in the PacMan domain show that the PGSS algorithm can be used to solve problems of non-trivial horizon depths and thus have a dramatic effect on the performance of the planner. PGSS can also be applied to other domains, ones that can benefit from action abstractions, in a straightforward manner. We would like to highlight that our approach can be viewed as an addendum to existing tree search algorithms, i.e. integrating them with action abstractions in a specific manner and showing its advantages. Extending it to other state-of-the-art techniques in MCTS literature like UCT is a promising area of future work. We are also interested in exploring other kinds of action abstractions that PGSS can utilize.

References

[1] R. Aler, O. Garcia, and JM Valls. Correcting and improving imitation models of humans for robosoccer agents. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, 2005.

[2] B.D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[3] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *J. Mach. Learn. Res.*, 3:397–422, March 2003.

[4] Aijun Bai, Siddharth Srivastava, and Stuart J. Russell. Markovian state and action abstractions for mdps via hierarchical MCTS. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3029–3039, 2016.

[5] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[6] Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 316–324, 2014.

[7] M. Cakmak and A.L. Thomaz. Optimality of human teachers for robot learners. In *Proceedings of the IEEE International Conference on Development and Learning (ICDL)*, 2010.

[8] Guillaume Chaslot, Christophe Fiter, Jean-Baptiste Hoock, Arpad Rimmel, and Olivier Teytaud. Adding expert knowledge and exploration in monte-carlo tree search. In *ACG*, pages 1–13, 2009.

[9] Marco Dorigo and Marco Colombetti. Robot shaping: Developing autonomous agents through learning. *Artif. Intell.*, 71(2):321–370, 1994.

[10] H. Finnsson and Y. Bjornsson. Simulation-based approach to general game playing. In *Proceedings of the 23rd national conference on Artificial intelligence*, pages 259–264, 2008.

[11] S. Gelly and Y. Wang. Exploration exploitation in go: Uct for monte-carlo go. In *Twentieth Annual Conference on Neural Information Processing Systems (NIPS 2006)*. Citeseer, 2006.

[12] D.H. Grollman and O.C. Jenkins. Dogged learning for robots. In *IEEE International Conference on Robotics and Automation*, pages 2483–2488. Citeseer, 2007.

[13] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in Neural Information Processing Systems 27*, pages 3338–3346. Curran Associates, Inc., 2014.

[14] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[15] Arya J. Irani. *Utilizing Negative Policy Information To Accelerate Reinforcement Learning*. PhD thesis, Georgia Institute of Technology, 2015.

[16] M. Kearns, Y. Mansour, and A.Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2):193–208, 2002.

[17] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, pages 282–293, 2006.

[18] George Konidaris, Scott Kuindersma, Andrew G. Barto, and Roderic A. Grupen. Constructing skill trees for reinforcement learning agents from demonstration trajectories. In *NIPS*, pages 1162–1170, 2010.

[19] Zhihui Luo, David A. Bell, and Barry McCollum. Skill combination for reinforcement learning. In *IDEAL*, pages 87–96, 2007.

[20] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *ICML*, 2004.

[21] A.Y. Ng, H.J. Kim, M.I. Jordan, S. Sastry, and S. Ballianda. Autonomous helicopter flight via reinforcement learning. *Advances in Neural Information Processing Systems*, 16, 2004.

[22] Doina Precup. *Temporal abstraction in reinforcement learning*. PhD thesis, UMass Amherst, 2000.

[23] Khashayar Rohanimanesh and Sridhar Mahadevan. Decision-theoretic planning with concurrent temporally extended actions. In *UAI*, pages 472–479, 2001.

[24] Khashayar Rohanimanesh and Sridhar Mahadevan. Learning to take concurrent actions. In *NIPS*, pages 1619–1626, 2002.

[25] Jesse Rosalia, Guliz Tokadli, Charles L. Isbell Jr, Andrea L. Thomaz, and Karen M Feigh. Discovery, evaluation, and exploration of human supplied options and constraints. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1873–1874. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[26] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[27] Kaushik Subramanian, Charles Isbell, and Andrea Thomaz. Learning Options through Human Interaction. In *Workshop on Agents Learning Interactively from Human Teachers at IJCAI*, 2011.

[28] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

[29] R.S. Sutton and A.G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[30] Güliz Tokadlı and Karen M Feigh. Application of abstraction hierarchies to incorporate human knowledge for machine learning a general form for mario bros. & pac-man. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 59, pages 657–661. SAGE Publications, 2015.

[31] T.J. Walsh, S. Goschin, and M.L. Littman. Integrating sample-based planning and model-based reinforcement learning. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI)*, 2010.

[32] Peng Zang, Peng Zhou, David Minnen, and Charles Lee Isbell Jr. Discovering options from example trajectories. In *ICML*, page 153, 2009.