Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Improving Exploration in UCT Using Local Manifolds

Sriram Srinivasan
University of Alberta
ssriram@ualberta.ca

Erik Talvitie
Franklin and Marshal College
erik.talvitie@fandm.edu

Michael Bowling
University of Alberta
mbowling@cs.ualberta.ca

Abstract

Monte Carlo planning has been proven successful in many sequential decision-making settings, but it suffers from poor exploration when the rewards are sparse. In this paper, we improve exploration in UCT by generalizing across similar states using a given distance metric. When the state space does not have a natural distance metric, we show how we can learn a local manifold from the transition graph of states in the near future. to obtain a distance metric. On domains inspired by video games, empirical evidence shows that our algorithm is more sample efficient than UCT, particularly when rewards are sparse.

1 Introduction

A useful approach to sequential decision problems involving large state spaces, is to search in the action space, by building a search tree using samples from a model (Browne et al. 2012). One such recently developed algorithm is UCT (Kocsis and Szepesvári 2006). The popularity of these planning algorithms is due to the fact that the complexity of the action selection step is independent of the size of the state space.

However, UCT performs poorly when the rewards are *sparse*, i.e., when only a small fraction of trajectories observe non-zero reward. Consider the grid world domain shown in Figure 1(a). The agent starts at the solid square in blue. The goal state is marked by a red X and the agent receives a reward of $+1$ on reaching the goal state, leading to episode termination. The agent has 4 actions available - move UP, DOWN, LEFT and RIGHT that work as suggested by their names. In the absence of reward from rollouts, UCT essentially follows random behavior. For a reasonably distant goal state, a uniform random policy is unlikely to observe any reward. Figure 1(b) shows the heat map of states visited by UCT after 100 rollouts from the start state where each state is given by a pair (i, j), $0 \leq i, j < 40$. Due to its random rollout policy, UCT keeps visiting states similar to those it has already seen before. As a result, UCT needs a large number of rollouts to see reward.

An alternative approach would be to systematically explore the state space, collecting returns from states that are different from the ones already visited. However, a full expansion of the search tree is impractical. We want to retain the strengths of Monte Carlo planning in using samples, while still achieving efficient exploration. In this paper we aim to exploit similarity between states, and augment UCT with this information, to achieve efficient exploration during planning. If the state representation does not provide a similarity metric, we learn a manifold of the state space, thus embedding states in Euclidean space where we can compute similarity using a distance metric. We first review some background material in the next section, before describing our algorithm in Section 3.

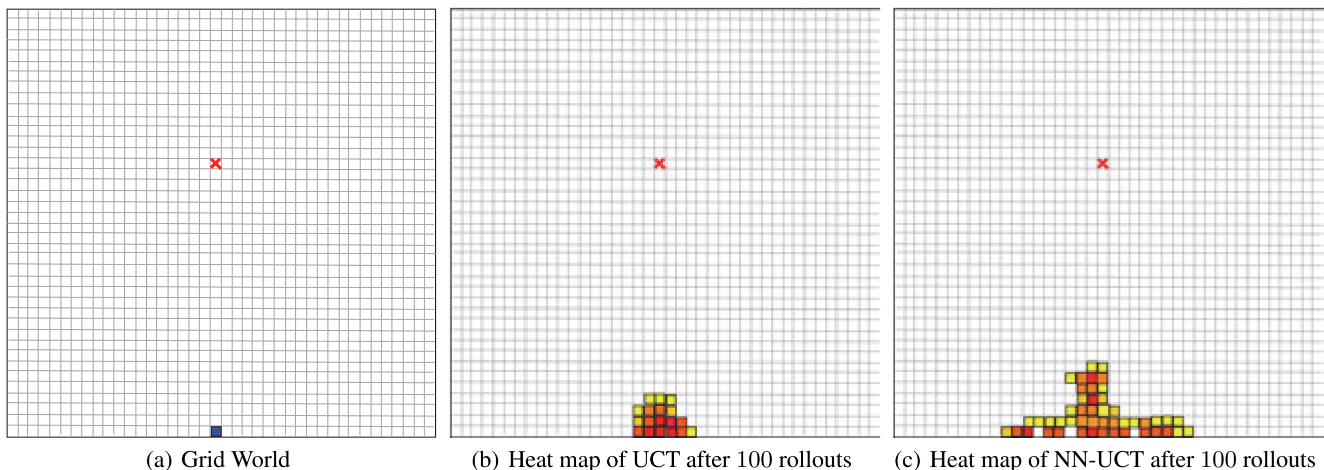
Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

2 MDPs and UCT

In this work, we focus on finite, deterministic Markov Decision Processes (MDPs), defined as a tuple $\langle S, A, T, R, \gamma \rangle$, where S is the finite state space, A is the finite action space, $T : S \times A \mapsto S$ is the transition function, $\rho : S \times A \mapsto \mathbb{R}$ is the reward function; $\rho(s, a)$ is the immediate reward obtained by taking action a from state s, and γ is the discount factor that trades off between short-term and long-term rewards. We define a policy $\pi : A \times S \mapsto [0, 1]$ to be a probability distribution over actions, given a state. The state-action value function of π is defined as $Q^\pi(s, a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \rho(s_t, a_t) | s_0 = s, a_0 = a]$ where $a_t \sim \pi(\cdot, s_t)$, $s_t = T(s_{t-1}, a_{t-1})$, $t \geq 1$. An optimal policy, π^*, has action-value function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$, $s \in S, a \in A$.

Monte Carlo Tree Search (MCTS) algorithms build a search tree with nodes corresponding to states and edges corresponding to actions. The tree is built iteratively using the *tree policy*. The tree policy dictates action selection at each level of the tree, leading to the creation of a leaf node. Once created, a *rollout* (usually random actions) is performed from the leaf node up to the end of planning horizon or episode termination. Let D denote the set of nodes in the tree. Note that two or more nodes can share the same state $s \in \mathcal{S}$. Let $S(d)$ denote the state corresponding to node $d \in D$. Each node maintains the returns obtained by rollouts starting from that node, $R(d)$, as well as the visit count, $n(d)$. The value of the node is estimated as $V(d) = \frac{R(d)}{n(d)}$. The action value estimate is $Q(d, a) = \rho(S(d), a) + \gamma V(d')$, where d' is the child node reached by taking action a.

THIS IS A PRELIMINARY VERSION OF THIS PAPER
THE FINAL VERSION WILL APPEAR IN THE AAAI DIGITAL LIBRARY



(a) Grid World (b) Heat map of UCT after 100 rollouts (c) Heat map of NN-UCT after 100 rollouts

Figure 1: UCT on Grid World

UCT (Kocsis and Szepesvári 2006) is an MCTS algorithm, that uses the UCB1 algorithm (Auer, Cesa-Bianchi, and Fischer 2002) for its *tree policy*. At each step in the tree policy, UCT treats the action selection step, as a multi-armed bandit problem. It computes a UCB score and picks the action corresponding to the highest score. The score for node d and action a is given by

$$Q_{ucb}(d,a) = Q(d,a) + C\sqrt{\frac{2\ln\sum_{a'\in A} n(d,a')}{n(d,a)}}\,.$$

The second term is a so-called exploratoration bonus, which encourages actions taken less frequently from a node, where C is a positive parameter, $n(d,a)$ is the number of times, action a was taken from node d and hence $n(d,a) = n(d')$, where d' is the child node reached by taking action a. We also have $\sum_{a\in A} n(d,a) = n(d)$. After the computational budget is exhausted in building the tree the greedy action at the root, $\arg\max_{a\in A} Q(d_0,a)$, is picked, breaking ties by choosing the most frequent action $\arg\max_{a\in A} n(d_0,a)$.

3 NN-UCT

By adding generalization in the UCB score, a good estimate of the node's statistics can be obtained from the statistics of other nodes in the tree, thus making better use of samples. A natural way to incorporate generalization is to combine the statistics of *similar* nodes. One way to achieve this is to compute a nearest neighbor estimate of the return and visit counts for each node d, as given by

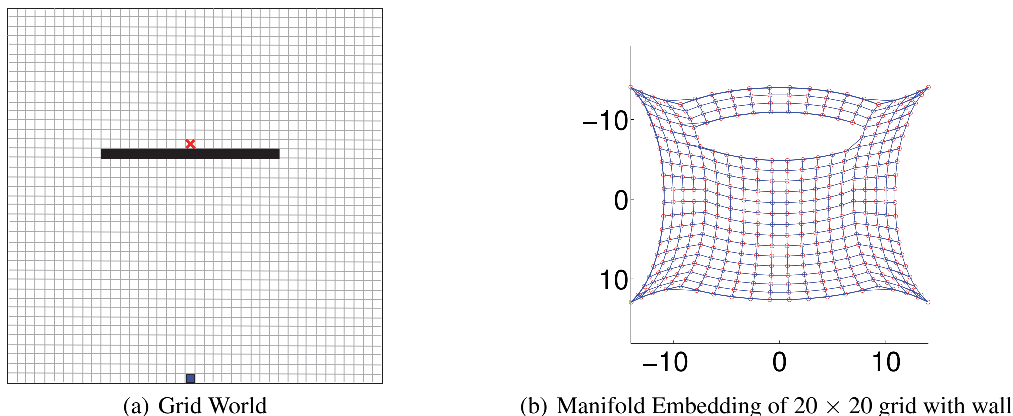
$$R_{nn}(d) = \sum_{d'\in D} K(S(d), S(d'))R(d')\,,$$
$$n_{nn}(d) = \sum_{d'\in D} K(S(d), S(d'))n(d')\,,$$
$$V_{nn}(d) = \frac{R_{nn}(d)}{n_{nn}(d)}\,,$$

where $K(s,s')$ is a nonnegative valued kernel function that measures similarity between states s and s'. A popular choice for the kernel function when states are real-valued vectors, is the Gaussian kernel function, $K(s,s') = \exp\left(\frac{\|s-s'\|_2}{\sigma}\right)$, where σ is the Gaussian width parameter which controls the degree of generalization. As $\sigma \to 0$, the nearest neighbor estimate approaches the combined MC estimate of all the nodes that share the same state. Note that the nearest neighbor estimate adds bias, which could prevent the guarantee of continual exploration of all actions. Hence, asymptotically, we want the returns and visit count estimates to be closer to the unbiased MC estimates. One resolution is to employ a *decay scheme* that changes the similarity over time so that the similarity between distinct states converges to zero. For example, with a Gaussian kernel function, the Gaussian widths of the nodes can be shrunk to zero as planning continues. In Section 3.1, we discuss one such scheme for reducing σ smoothly over time. The new UCB score is computed as

$$Q_{ucb}(d,a) = Q_{nn}(d,a) + C\sqrt{\frac{2\ln\sum_{a'\in A} n_{nn}(d,a')}{n_{nn}(d,a)}}\,.$$

Here $Q_{nn}(d,a) = \rho(S(d),a) + \gamma V_{nn}(d')$, where d' is the child node of node d when action a is taken from state $S(d)$. The quantity, $n_{nn}(d,a) \equiv n_{nn}(d')$, is taken from the child node d'. Thus, the nearest neighbor estimate of the number of times action a was taken from state $S(d)$ is obtained from the number of state visits to the after-state $S(d')$. In our algorithm, unlike UCT, $\sum_{a\in A} n_{nn}(d,a) \neq n_{nn}(d)$. Neither the actual visit count ($n(d)$), nor the nearest neighbor estimate ($n_{nn}(d)$) of the parent node is used in the exploration term. Only the estimates of the child nodes (after-states) are used for action selection and actions leading to after-states similar to the ones already explored are not preferred. The idea of using after-state counts in the exploration bonus has been explored previously (Méhat and Cazenave 2010), but they did not incorporate generalization. Figure 1(c) shows the heat map of states visited by NN-UCT on the grid world domain after 100 rollouts from the start state and a Gaussian kernel with $\sigma = 100$ is used with a decay scheme (explained

THIS IS A PRELIMINARY VERSION OF THIS PAPER
THE FINAL VERSION WILL APPEAR IN THE AAAI DIGITAL LIBRARY



(a) Grid World (b) Manifold Embedding of 20×20 grid with wall

Figure 2: Grid World with Wall

in the next section) that uses $\beta = 0.9$. NN-UCT directs the tree to be grown in areas farther from states that have been encountered already. In the absence of reward, with ties broken using visit counts, the action picked at the root would be one that leads to a state in the direction of under-explored states, in this case "UP". Thus the NN estimates not only help guide tree expansion, but also action selection when no rewards are encountered during planning.

3.1 Decay Scheme

In this section we discuss a decay scheme for Gaussian kernels. Given the initial Gaussian kernel width σ, we want a scheme which shrinks the Gaussian widths of nodes such that the value estimates and visit counts are closer to the MC estimate, asymptotically. We use a simple exponential decay with rate governed by a parameter $\beta \in (0, 1)$. In this scheme, we shrink the Gaussian width of the child nodes whenever the tree policy is executed at the parent node. Thus, the Gaussian width of a node d' is given by $\sigma(d') = \sigma\beta^{n(d)}$, where $n(d)$ is the count of the visits to the parent node d. The Gaussian width of the root node is shrunk after each rollout. This scheme ensures that if erroneous generalization has starved visits to d', it will eventually be explored once its parent has been visited sufficiently often.

3.2 Generalization in Complex Domains

In the grid world example shown in Figure 1(a), the Euclidean distance between two points on the grid was a good choice for a distance metric. However, in general, the grid coordinates may not be the best representation. For example, consider a grid world with a wall as shown in Figure 2(a). In this case, distance between grid coordinates is not a good distance metric, as states on either side of the wall are farther apart than those on the same side. Furthermore, not all state representations have a natural metric space for defining similarity. High dimensional state spaces pose an additional problem of tractably computing a distance metric. Hence, we need a state representation that captures the underlying topology of the environment that is low dimensional, allowing fast computation of distances between states. Low dimensional manifolds are a natural choice that satisfy all the requirements.

4 mNN-UCT

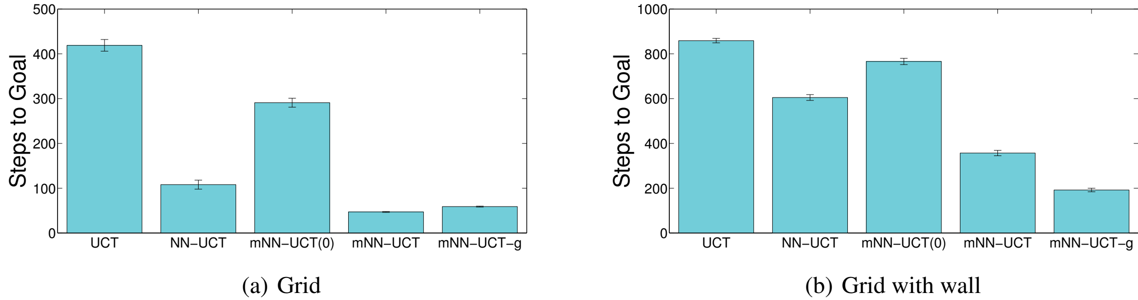
A manifold $\mathcal{M} \subset \mathbb{R}^m$ is a set of points such that the Euclidean distance between nearby points on the manifold approximates the true distances between those points. Figure 2(b) shows an example of a 2D manifold of a 20×20 grid, with a wall as shown. Manifold learning (Ma and Fu 2012) typically involves learning a low-dimensional representation, subject to distance preserving constraints. Manifold learning algorithms rely on the geodesic distances between neighboring points, while approximating other distances. Euclidean distance between points on the manifold can then serve as a distance metric for measuring similarity of points. Thus, the similarity measure needed in our NN-UCT algorithm can be obtained by embedding the states on a manifold.

We can embed the states by applying a manifold learning algorithm on the distance matrix constructed from the transition graph of states in the MDP. The computational complexity of most manifold learning algorithms is $\mathrm{O}(m^3)$, where m is the number of points needed to be embedded. This cost is due to the distance matrix construction and subsequent eigen-decomposition. Hence, it is computationally infeasible to embed all of the possible states in large problems. One way to deal with the high computational cost of embedding states on a manifold is to embed only those states that may be encountered in the near future. As we are operating in the planning setting, with access to a model, we can sample the states reachable in the near future and construct a manifold that is local to the current state. We next describe an algorithm to construct such a manifold.

4.1 Local Manifolds

For computational feasibility, we learn a local manifold over the state space. Local manifolds provide a topology of the state space in the local neighborhood of the current state. In our work, at every decision step, we learn a local manifold of

THIS IS A PRELIMINARY VERSION OF THIS PAPER
THE FINAL VERSION WILL APPEAR IN THE AAAI DIGITAL LIBRARY



(a) Grid

(b) Grid with wall

Figure 3: Performance on Grid World Domains

the deterministic MDP from the sample transitions encountered during a short breadth-first walk of the state space, starting from the current state. We construct the distance matrix of the all-pairs shortest path distances between states in the BFS tree with distance between two states measured as the minimum number of actions required to get from one to the other. We apply the Isomap algorithm (Tenenbaum, De Silva, and Langford 2000) on the distance matrix to generate a Euclidean embedding of the states. This ensures that states that are nearby in time are nearby in space as well.

During UCT planning, we may encounter states that were not encountered in the BFS walk, and therefore do not have an embedding. This situation is referred to as the out-of-sample problem and is more likely to occur as the UCT tree grows larger. If we were to embed these states on a manifold, we would need a deeper BFS walk from the start state, resulting in a possibly exponential increase in the number of states to be embedded.

We address the out-of-sample problem in the following way. We generate the approximate embeddings of out-of-sample states by applying an operator learnt for each action in the action set. We learn this operator from the manifold embeddings of the states encountered during the BFS walk for the local manifold. Let M_k denote the set of tuples $\langle x, k, y\rangle$, such that y is the manifold embedding of the state obtained by taking action $k \in A$ from the state with manifold embedding x. We learn a simple translation operator $f_k(x) = x + b_k$ for this action where b_k is given by

$$b_k = \frac{1}{|M_k|} \sum_{t \in M_k, t=\langle x,k,y\rangle} \|y - x\|_2.$$

We found the simple translation operator to be useful in the domains tested here. Applying more sophisticated solutions to the out-of-sample problem is a direction for future work.

NN-UCT with manifolds, referred to as mNN-UCT, uses the manifold embedding to compute the kernel function. If a state outside of the BFS walk is encountered during planning, its embedding is computed by applying the translation operator to the state from which the current action was taken.

5 Experiments

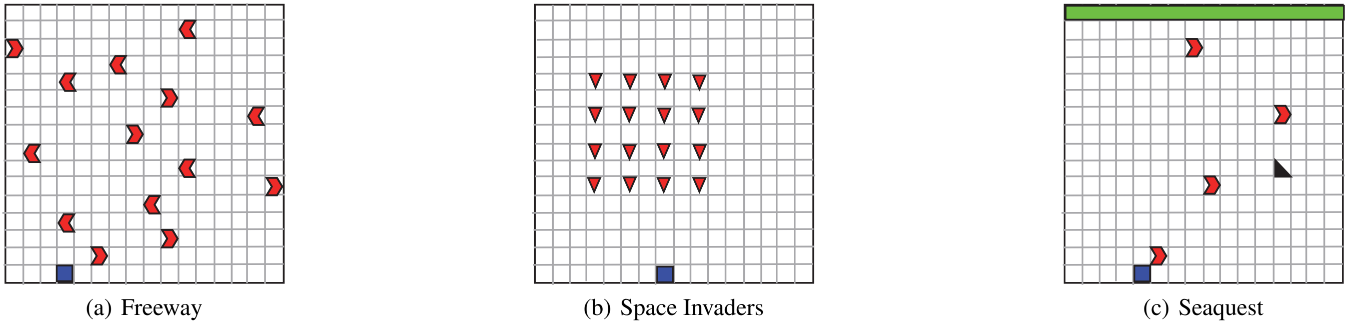
In our UCT implementation, whenever a new node is created, a rollout (random actions) was run for a constant length of length 50, rather than out to a fixed horizon. This helped prevent finite horizon effects from affecting generalization. The optimal branch of the UCT tree and the BFS tree were retained across planning steps. A wide range $\{10^{-5}, 10^{-4}, \ldots, 10^3, 10^4\}$ was tested for the UCB parameter with 500 trials performed for each parameter setting.

5.1 Grid World Domain

We first validate our intuitions about the impact of state generalization in UCT by evaluating the algorithms on the two grid world domains shown in Figures 1 and 2 and described in Sections 1 and 3.2. An episode terminates when the agent reaches the goal state, or 1000 time-steps have elapsed, whichever occurs first. We report the number of steps taken to reach the goal state using 100 rollouts for each planning step. For the UCT algorithms with generalization, the Gaussian kernel function with an initial Gaussian width σ chosen from $\{10, 100, 1000, 10000\}$ was used, and the decay rate β chosen from $\{0.1, 0.5, 0.9, 0.99\}$, and the best result is reported. The Euclidean distance was used for the distance metric in the kernel function. For the mNN-UCT algorithm, the BFS walk included at most 400 states for each planning step. The mNN-UCT algorithm with $\sigma = 10^{-6}$, referred to as mNN-UCT(0) essentially shares statistics only between nodes with the same state. This tests whether generalizing from similar states is helpful. mNN-UCT-g is the mNN-UCT algorithm using a global instead of a local manifold, where a full BFS walk is used to construct the manifold, allowing us to evaluate the effectivness of local manifolds.

Figure 3 summarizes the results of our grid world experiments. The mean number of steps (fewer steps being better), with standard error bars is shown. Sharing statistics between nodes with exactly the same states (mNN-UCT(0)) improves UCT slightly. Adding more widespread generalization to UCT improves the performance dramatically in both the domains, with the NN-UCT algorithm reaching the goal state in less than half the number of steps taken by UCT in the first domain. The best performing σ for the mNN-UCT and mNN-UCT-g algorithms was substantially greater than 0, suggesting that generalizing over nearby states in the neighborhood of the manifold is useful. All the algorithms, as expected, take more steps to reach the goal when the wall is added. The NN-UCT algorithm, which uses the Euclidean distance metric between grid coordinates, does not perform

THIS IS A PRELIMINARY VERSION OF THIS PAPER
THE FINAL VERSION WILL APPEAR IN THE AAAI DIGITAL LIBRARY



(a) Freeway (b) Space Invaders (c) Seaquest

Figure 4: Games Domains

as well in the second domain. The naive distance metric is not as useful since states on either side of the wall are farther apart than the Euclidean distance indicates. mNN-UCT does not suffer from this problem, as it better captures the topology of the state space. In both the domains, using local manifolds rather than global manifolds does not substantially affect planning performance.

5.2 Game Domains

We also test our algorithm on three domains inspired by video games. Each domain is a single player games played on a 16×16 board. The initial configuration of the games is shown in Figure 4. All the games last for 256 time-steps or until the agent loses all of its lives, whichever is shorter. The agent is depicted by the blue square.

1. Freeway: Cars are depicted by red chevrons. Their positions on each row and direction of movement is set arbitrarily at the start. On reaching the end of the board, they reappear back on the opposite side. The agent has 3 lives. On collision, the agent loses a life and its position is reset to its start position. The agent has 3 actions: NO-OP (do nothing), UP and DOWN. The agent receives a reward of $+1$ on reaching the topmost row after which the agent's position is reset to start state. The maximum possible score is 16 in the absence of cars, although with cars, the optimal score is likely lower.
2. Space Invaders: The aliens are depicted by red triangles. These aliens fire bullets vertically downward, at fixed intervals of 4 time-steps. The aliens move from left to right. They change directions if the rightmost (or leftmost) alien reaches the end of the screen. The agent loses its only life if it gets hit by a bullet from the alien. The following actions are available to the agent: NO-OP, FIRE, LEFT, RIGHT, LEFT-FIRE (fire bullet and move left) and similarly RIGHT-FIRE (fire bullet and move right). The maximum possible score is 32 (reward of 1 for shooting an alien) in the absence of the alien's bullets, with a realizable score likely lower.
3. Seaquest: Fish depicted by red chevrons, move from left to right and reappear back on reaching the end of the screen. The diver, indicated by a black triangle, appears at either end, alternately. The diver appears when the current diver is collected by the agent or if the current diver reaches end of the screen. The agent receives a reward of +1 if it collects the diver and drops it at the drop point (top row, indicated in green). The agent loses its only life on collision with a fish, or on reaching the drop point without the diver. The actions available to the agent are: NO-OP, UP, DOWN, LEFT and RIGHT. The maximum possible score is 8 in the absence of fish, as it is impossible to capture any 2 successive divers and drop them at the drop point, though this score may not be realizable.

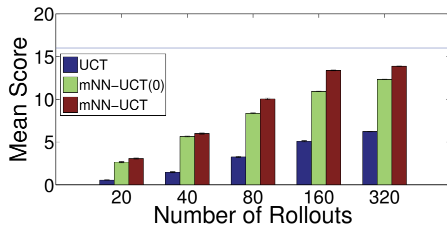
In all of these domains the agent does not receive the positions of the various objects, and instead needs to rely on the manifold over the transition graph to define similarity between states. Furthermore, learning a global manifold would likely be high dimensional and computationally infeasible.

Performance Comparison. We compare mNN-UCT with UCT on the domains described above. We again evaluate the effect of generalization by running our algorithm with $\sigma = 10^{-6}$ referred to mNN-UCT(0). We ensure that all of the algorithms are given the same number of samples per planning step. We vary the number of rollouts from $\{20, 40, 80, 160, 320\}$. The discount factor was set at 0.99. We fix the other parameters for the mNN-UCT algorithm with $\sigma = 100$, and $\beta = 0.9$. We learn a local manifold for each planning step from a BFS tree comprising of at most 400 states. For each of the domains, we measure the mean score obtained per episode over all 500 trials. The standard errors are indicated as error bars. The optimistic maximum score possible is indicated by a blue line. Note that it may not be possible to achieve this score in all domains.

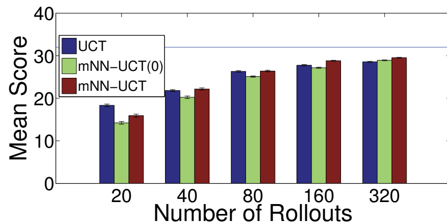
Figure 5 shows the performance of the 3 algorithms against the number of rollouts. The results are reported for the parameters which produced the best mean score obtained by performing 500 trials. mNN-UCT outperforms UCT, especially in Freeway and Seaquest. Both these games require the agent to perform a specific sequence of actions in order to obtain non-zero reward. UCT's rollout policy is sufficient to achieve a high score in Space Invaders as the game does not have sparse rewards. Generalization using manifolds provides only a marginal improvement in this game.

Parameter Sensitivity. We evaluate the sensitivity of the algorithm with respect to the parameters σ and β, which together control the degree of generalization in the UCT al-

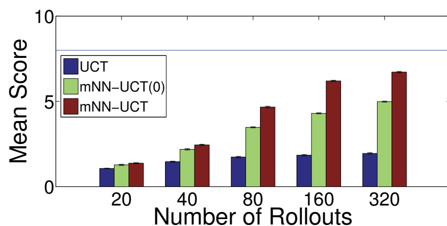
THIS IS A PRELIMINARY VERSION OF THIS PAPER
THE FINAL VERSION WILL APPEAR IN THE AAAI DIGITAL LIBRARY



(a) Freeway

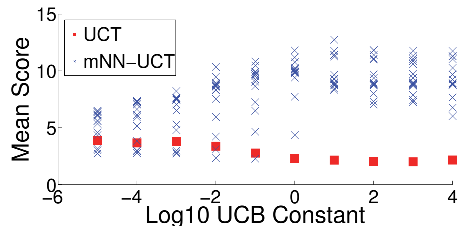


(b) Space Invaders

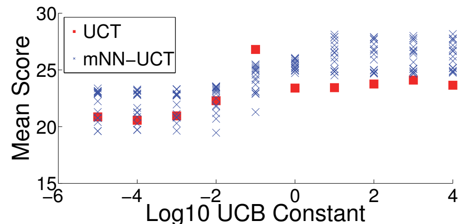


(c) Seaquest

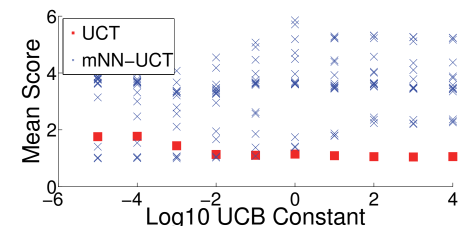
Figure 5: Performance on Games Domains



(a) Freeway



(b) Space Invaders



(c) Seaquest

Figure 6: Parameter Sensitivity

gorithm. In this experiment, we fix the number of rollouts at 100. We vary σ from $\{10, 100, 1000, 10000\}$ and β in $\{0.1, 0.5.0.9, 0.99\}$. For each parameter setting, we measure the performance of the mNN-UCT algorithm and compare it to UCT using various settings of the UCB parameter.

Scatter plots are shown in Figure 6 where each red square represents the mean score per episode obtained by UCT with a specific setting of the UCB parameter and blue x represents the score obtained by mNN-UCT with a specific setting of σ and β. We see that the mNN-UCT algorithm performs better than UCT for a wide range of parameter settings in all the domains. Using widespread generalization achieves significantly higher score on sparse reward games such as Freeway and Seaquest, while the improvement is modest when rewards are easy to obtain in the game such as Space Invaders.

6 Discussion

We observed that adding generalization improves UCT significantly in sparse reward conditions. By using the nearest neighbor heuristic, we are effectively making the assumption that states located nearby in space also have similar values. Such an assumption may not hold in all domains, although in our experiments it proved helpful. Although the decay scheme to shrink the Gaussian kernel widths of nodes in the UCT tree should overcome any adverse effects of generalization, we do not have consistency guarantees on the value estimates. We defer a formal proof to future work.

While we have shown empirically that state generalization helps UCT, the improvement comes at an additional cost. In our algorithm, the cost is due to manifold learning and computing the nearest neighbor estimates. The former is controlled by the size of the BFS walk. The latter is controlled by the number of rollouts, which determines the number of nodes in the tree. In our experiments, calls to the simulator dominated the runtime cost, so the cost of manifold learning was insignificant compared to the cost of planning. Furthermore, experimental results have shown that fewer rollouts may be needed to achieve good performance and hence the cost incurred by computing the nearest neighbor estimate may be justified. For large UCT trees we can be more efficient by having a distance threshold beyond which neighbor estimates are ignored using K-d trees (Bentley 1975).

6.1 Related Work

A common enhancement to UCT is the use of transposition tables (Childs, Brodeur, and Kocsis 2008). Transposition tables offer a basic form of generalization, by sharing statistics of nodes in the UCT tree having the same state. Our algorithm with $\sigma \approx 0$ closely resembles UCT with transposition tables. Although transposition tables help speed up search, they provide limited exploration benefit as they can only recognize state equality. In large state spaces, the likelihood of

THIS IS A PRELIMINARY VERSION OF THIS PAPER
THE FINAL VERSION WILL APPEAR IN THE AAAI DIGITAL LIBRARY

encountering transpositions is low, and hence transposition tables may be less useful. This was noticed even in our relatively small experimental domains, with more aggressive generalization improving on basic transposition tables.

State abstraction using homomorphisms in UCT adds another layer of generalization by grouping states with the same transition and reward distributions. The idea of using local homomorphisms (Jiang, Singh, and Lewis 2014) as an approximate state abstraction was an inspiration for the development of using local manifolds. The homomorphism groups states together but, like transposition tables, it does not have a degree of similarity, only a hard notion of equivalence. In the grid world domain in Figure 1(a), the only homomorphism is the grid itself. Thus, homomorphisms are no better than transposition tables in this case.

Another approach to generalization in UCT is Rapid Action Value Estimation (RAVE) (Gelly and Silver 2011). RAVE uses the *All-Moves-As-First* (AMAF) heuristic which states that the value of an action is independent of when it is taken, i.e. if an action is profitable at a subtree $\tau(s)$, it is profitable immediately at the root s as well. RAVE combines the Monte Carlo estimate with the AMAF estimate, $Q_{rave}(s,a) = (1-\beta)Q(s,a) + \beta Q_{AMAF}(s,a)$, where $0 < \beta < 1$ is a weight parameter that decreases with increasing number of rollouts. The RAVE estimate is used in place of $Q(s,a)$ for the tree policy. RAVE has been shown to be successful in Computer Go and has also been applied in continuous action spaces (Couetoux et al. 2011). In that case, the RAVE estimate for action a is obtained by computing a nearest-neighbors estimate of the returns in the subtree where action a was taken. However, the AMAF heuristic does not modify the exploration bonus. Thus when no rewards have been encountered, its exploration strategy suffers from the same problems as UCT.

Previous work on learning manifolds in MDPs includes proto-value functions (PVFs) (Mahadevan and Maggioni 2007). Proto-value functions are used to learn a low dimensional representation from a transition graph generated by following a fixed policy which is subsequently improved using policy iteration. PVFs assume that a random walk of the state space is sufficient to construct basis functions that reflect the underlying topology. However, in many domains with large state spaces, such as video games, a policy consisting of random actions does not provide a sufficient view of the state space. Additionally, it may be computationally infeasible to embed all of the states encountered by following the policy. To address these concerns, we introduced local manifolds which, in our experiments, seem to offer similar benefits to using global manifolds.

7 Conclusion

In this paper, we presented a method of incorporating state generalization into UCT by using nearest neighbor estimates of the action-value and the visit counts in the tree policy. We also showed that in domains whose states do not have a natural distance metric we can learn a local manifold of the state space that gives us a Euclidean representation of the states in the near future. We demonstrated the benefit of learning such local manifolds on video game domains. We found that our algorithm substantially outperformed UCT with no generalization, especially in environments with sparse rewards.

Acknowledgements

The authors would like to thank Ujjwal Das Gupta and Marlos Machado for their inputs. We would also like to thank Vadim Bulitko and Csaba Szepesvári for their feedback. This research was supported by the Alberta Innovates Center for Machine Learning (AICML) and computing resources provided by Compute Canada through Westgrid.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47(2-3):235–256.

Bentley, J. L. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18(9):509–517.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of Monte Carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on* 4(1):1–43.

Childs, B. E.; Brodeur, J. H.; and Kocsis, L. 2008. Transpositions and move groups in Monte Carlo tree search. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, 389–395. IEEE.

Couetoux, A.; Milone, M.; Brendel, M.; Doghmen, H.; Sebag, M.; Teytaud, O.; et al. 2011. Continuous rapid action value estimates. In *The 3rd Asian Conference on Machine Learning (ACML2011)*, volume 20, 19–31.

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

Jiang, N.; Singh, S.; and Lewis, R. 2014. Improving uct planning via approximate homomorphisms. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, 1289–1296. International Foundation for Autonomous Agents and Multiagent Systems.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*. Springer. 282–293.

Ma, Y., and Fu, Y. 2012. *Manifold learning theory and applications*. CRC Press.

Mahadevan, S., and Maggioni, M. 2007. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research* 8(2169-2231):16.

Méhat, J., and Cazenave, T. 2010. Combining uct and nested Monte Carlo search for single-player general game playing. *Computational Intelligence and AI in Games, IEEE Transactions on* 2(4):271–277.

Tenenbaum, J. B.; De Silva, V.; and Langford, J. C. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science* 290(5500):2319–2323.

THIS IS A PRELIMINARY VERSION OF THIS PAPER
THE FINAL VERSION WILL APPEAR IN THE AAAI DIGITAL LIBRARY