Lower Bounding Klondike Solitaire with Monte-Carlo Planning

Ronald Bjarnason and **Alan Fern** and **Prasad Tadepalli**
{ronny, afern, tadepall}@eecs.oregonstate.edu
Oregon State University
Corvallis, OR, USA

Abstract

Despite its ubiquitous presence, very little is known about the odds of winning the simple card game of Klondike Solitaire. The main goal of this paper is to investigate the use of probabilistic planning to shed light on this issue. Unfortunatley, most probabilistic planning techniques are not well suited for Klondike due to the difficulties of representing the domain in standard planning languages and the complexity of the required search. Klondike thus serves as an interesting addition to the complement of probabilistic planning domains. In this paper, we study Klondike using several sampling-based planning approaches including UCT, hindsight optimization, and sparse sampling, and establish empirical lower bounds on their performance. We also introduce novel combinations of these approaches and evaluate them in Klondike. We provide a theoretical bound on the sample complexity of a method that naturally combines sparse sampling and UCT. Our results demonstrate that there is a policy that within tight confidence intervals wins over 35% of Klondike games. This result is the first reported empirical lower bound of an optimal Klondike policy.

Introduction

Klondike Solitaire (commonly referred to simply as "Solitaire") has recently been named the most important computer game of all time (Levin 2008). Solitaire is easily represented, easily described and has appeared on every major distribution of Microsoft Windows. Despite its broad appeal and near-ubiquitous presence, very little is known about the odds of winning a game. Modern researchers have declared that "it is one of the embarrassments of applied mathematics that we cannot determine the odds of winning the common game of Solitaire" (Yan et al. 2005).

In addition to being highly stochastic, Solitaire involves complicated reasoning about actions with many constraints. Unlike many of the probabilistic planning domains introduced by planning researchers, it is independently developed and holds broader interest. There also exist many variants of the game which allows for a systematic study of these variants with different planning approaches. As we argue below, Klondike poses several problems for probabilistic planning including representation and search. In this paper, we study several sampling based planning approaches and establish empirical lower bounds on their performance. We also introduce some novel combinations of these approaches and provide theoretical and empirical evaluations. Our results show that there is a policy that within tight confidence intervals wins over 35% of Klondike games, thus establishing the first empirical lower bound of an optimal Klondike policy.

Klondike Solitaire

Klondike Solitaire is a simple game played with a standard deck of 52 playing cards. Initially, 28 cards are dealt to 7 tableau stacks, 1 card to stack 1, 2 to stack 2, ... etc, with the top card of each stack face-up. The remaining 24 cards are placed in a deck. Four foundation stacks (one for each suit) are initially empty. An instance of the common Windows version of Klondike can be seen in Figure 1. The object of the game is to move all 52 cards from the deck and the tableau to the foundation stacks. Variants of Klondike define specific elements of game play. This research deals exclusively with a common variant that allows unlimited passes through the deck, turning three cards at a time, and allowing partial tableau stack moves.

The Klondike Solitaire domain is problematic due to a large branching factor and the depth required to reach a goal state. Klondike is played with 52 cards, 21 of which are initially face-down. For each initial deal, there are 21! possible hidden configurations. This amount effectively eliminates the possibility of any pure reasoning regarding the odds of winning any initial deal. Klondike games are also relatively lengthy. Even the most straightforward games require a minimum of 52 moves to reach a solution, as each of the 52 cards must eventually reach a foundation stack. Anecdotal evidence suggests that typical human players win around 15% of games (Diaconis 1999). One common method of scoring games, known as "Las Vegas" style, pays players five fold their up-front per-card cost for each card that reaches a foundation stack. Such a payoff suggests that a strategy that wins 20% of games should be considered a success. In reality, there is a complete lack of real evidence to demonstrate bounds on Klondike strategies of any kind.

Prior to this, we have conducted some research (Bjarnason, Tadepalli, and Fern 2007) in a version of Solitaire that

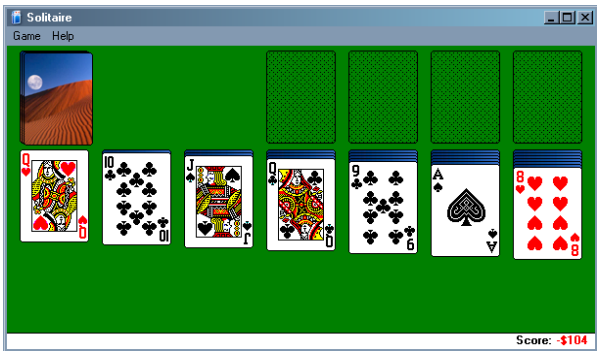


Figure 1: The Windows version of Klondike Solitaire.

allows a planner to know the location of all 52 cards, known as Thoughtful Solitaire. Using a deterministic planner, we demonstrated a policy that wins at least 82% of games in this deterministic setting, but offered no conclusions regarding the original probabilistic Klondike Solitaire. Recent *determinization* approaches to probabilistic planning, such as FF-Replan (Yoon, Fern, and Givan 2007) and Hindsight Optimization (HOP) (Yoon et al. 2008), create deterministic instances of probabilistic problems by fixing the stochastic elements of the domain, allowing these converted problems to be solved using established classical planners. Another Monte-Carlo method, UCT (Kocsis and Szepesvári 2006) has recently been shown to be very successful in the complex game of Go (Gelly and Silver 2007).

Our contributions in this paper are three-fold. First, we present the domain of Klondike Solitaire as a challenging stochastic planning problem and establish some baseline performance bounds. Second, we outline the shortcomings of modern planning solutions and present novel solution techniques that combine UCT, HOP and sparse sampling to solve a significant percentage of Klondike instances. Third, we extend theoretical guarantees associated with sparse sampling and UCT to a method we call "Sparse UCT" that combines and generalizes these methods.

The remainder of the paper proceeds as follows: We discuss the probabilistic Klondike problem, highlighting the difficulties a contemporary probabilistic planner will likely encounter. We describe two Monte-Carlo planning methods, Hindsight Optimization and UCT, and present planning algorithms based on these methods for the Klondike problem. We then describe and show results on Sparse UCT, including a proof bounding its sample complexity. We will conclude with a discussion of the possible future extensions and resolutions of the shortcomings of these approaches.

Probabilistic Planners for Klondike

Klondike can be represented as a probabilistic planning problem, where the outcome of actions uncovering a face-down card are determined by a uniform distribution over the set of unseen cards. Most of the standard moves in Klondike are deterministic, moving face-up cards from one stack to another without revealing the identities of face-down cards. Once all cards are face-up, probabilistic actions are no longer available and the planning problem becomes deterministic. Of course, in traditional play, the identity of each of the face-down cards is already fixed by the initial shuffle of the cards. The most straightforward representation of this domain is as a partially observable deterministic Markov decision process (POMDP). We can alternatively represent each state as a uniform distribution over a set of "belief states" corresponding to the permutations of possible locations of face-down cards. For purposes of finding an optimal strategy, we represent the same problem as a stochastic planning problem where each action that reveals a card randomly assigns the identity of that card based on a uniform distribution over the entire set of face-down cards.

Representing Klondike in a standard probabilistic planning language, such as PPDDL, is problematic because there is no standard distribution for each action. Unlike standard probabilistic planning domains, each action may require a different distribution of possible outcomes based on the number and identity of the current set of face-down cards. Defining Klondike in such a manner is cumbersome and complicated, as the representation can potentially increase exponentially in the size of the state description. Without some language extension that allows the enumeration of a set of unseen objects or events and removal of those objects from the set once they have been encountered, the application of standard planning algorithms to Klondike and other POMDP style domains will be problematic.

While it is not clear how to compactly describe Klondike in PPDDL, it is not difficult to implement a simulation model for Klondike. Moreover, Monte-Carlo sampling techniques such as Hindsight Optimization and UCT have shown great promise in recent years and only require simulation models. Monte-Carlo techniques approximate the values of states and actions by evaluating trajectories through the modeled space, thus making it possible to reason about actions without exploring entire sections of a search space.

With the availability of an existing deterministic planner for Thoughtful Solitaire, a straightforward attempt at a probabilistic planner for Klondike would apply the deterministic planner to the game of Klondike using a planner that formulates plans in stochastic domains by determinizing instances of the planning problem. Two such stochastic planners are FF-Replan and Hindsight Optimization. FF-Replan (Yoon, Fern, and Givan 2007) determinizes a probabilistic problem, formulates and follows a plan until it encounters a state outside of its plan, at which point it formulates a new plan from the current state. While this is relatively straightforward for the game of Klondike, it seems unlikely that it will find much success in Klondike. Such a planner would require frequent re-planning due to the small chance of the planner guessing the card locations correctly. A more promising approach would be to employ a deterministic planner utilizing Hindsight Optimization (HOP) (Yoon et al. 2008). Similar to FF-Replan, HOP uses a deterministic planner to solve determinized instances of the probabilistic problem. At each state, HOP independently samples each available action k times to approximate an average value for taking each action, eventually choosing the action that has the high-

est value. This approach, utilizing Monte-Carlo samples to estimate the value of each action has been shown to be very successful, especially in more difficult domains that have been determined to be "probabilistically interesting". Because of the success of the Monte-Carlo HOP algorithm, we will also consider another successful Monte-Carlo algorithms. UCT (Kocsis and Szepesvári 2006) is a method of generating search trees based on Monte-Carlo trajectories in stochastic domains. UCT intelligently samples trajectories based on an upper confidence bound calculated by an implementation of Hoeffding's inequality and is widely known as the base of the most successful computer players of the game of Go.

Planning in Klondike

In our work with Thoughtful Solitaire, (Bjarnason, Tadepalli, and Fern 2007) we demonstrated that altering the action space can significantly improve search efficiency without compromising the essential play of the game. This was accomplished by creating macro actions that absorbed the *turn-deck* actions. In the traditional representation only a single card in the deck is visible at any given time. Additional cards in the deck can only be reached by turning the deck, three cards at a time. By absorbing all *turn-deck* actions into a macro, all actions that would be available after some number of *turn-deck* actions are available at all times. Because of the demonstrated improvements, we adopt this action space in our work. Solutions to problems in this space can be directly translated to solutions in the traditional representation by re-inserting the *turn-deck* actions. We have provided a simulator[1] that represents this action space by turning all deck cards face-up and highlighting those cards that are playable through this macro. This representation can be seen in Figure 2.

In addition to this action space modification, we placed the following default preference over available actions:

1. moves from a tableau stack to the foundation stack that reveal a new card
2. moves to a foundation stack
3. moves from a tableau stack to another tableau stack that reveal a new card
4. moves from the deck to a tableau stack
5. moves from the foundation stack to a tableau stack
6. moves from a tableau stack to another tableau stack that do not reveal a new card

We adopt this ordering in this work as a simple greedy heuristic for action selection. Throughout all of our experiments, a simple search for a goal state using this greedy heuristic is used to quickly explore for the goal in those states with no face-down cards. If the greedy search discovers a win, the game is considered solved. If not, the action is determined by the default exploration method. Adopting this simple deterministic search helped improve our performance. In addition to these heuristics we also adopted a simple mechanism to prevent cycles by disqualifying all actions that repeat states previously visited in the sampled trajectory.

In the absence of any meaningful performance baseline, we used this greedy heuristic and a random search as a baseline for our own work. We found that a random strategy won 7.135% of games while a greedy strategy based on the previously described prioritized actions won 12.992% of games, each tested on one million randomly generated games. These results appear to confirm the estimates of human play suggested by other sources (Diaconis 1999). As with our other strategies, the random action selection mechanism utilized the greedy search method in states with no face-down cards.

Hindsight Optimization

Hindsight Optimization (HOP) is a straightforward way to use the existing deterministic planner in this stochastic environment. The general idea behind HOP is to estimate the value of each action in a state via calls to a deterministic planner on different determinizations of a probabilistic planning problem. In particular, a value of a state is estimated by forming a set of determinized problems from the state, solving each one with a deterministic planner, and then averaging the results. HOP uses these estimates to select actions via one-step greedy look-ahead.

In the case of Klondike, we can determinize a problem given a particular state by shuffling the identity of all face-down cards and revealing them to the planner so that all uncertainty is removed. Each determinized problem thus corresponds to an instances of Thoughtful Solitaire, for which we can apply our previously developed deterministic planner. The result of applying this planner to many determinized problems at a particular state is an estimate of the win probability of that state.

Our deterministic planner is based on a nested rollout search algorithm that utilizes a hand-coded weighted linear value function over binary features, defined in (Bjarnason, Tadepalli, and Fern 2007). The time complexity of this search grows exponentially with the nesting or search level. Search level 0 corresponds to a greedy 1-step lookahead search while a level 1 search is equivalent to a traditional roll out search. The increase in search time hampered our ability to apply this method to its fullest capability within Hindsight Optimization due to the fact that many determinized problems must be solved for each action selection. In particular, in some of our tests, we solved 100 determinized problems for each action at each decision point resulting in many thousands of calls to the deterministic planner during the coarse of a single game.

Finally, in our HOP experiments, as well as our UCT experiments, we found the greedy deterministic search previously described to greatly improve performance. Specifically, when all of the cards are finally revealed in a game, our greedy search is used to attempt to solve it, and if it can't then HOP is resumed to select the next action.

Results Our results for HOP based tests can be seen in Table 1. These results are impressive compared to our baseline, more than doubling the performance of the greedy method,

[1] at web.engr.orst.edu/∼ronny/k.html

Results for HOP Klondike Trials				
#Samp / decis	Search Level	Win Rate (99% conf.)	# Games	Av. sec / Game
10	0	22.96±0.34	1000000	43.54
100	0	27.20±0.80	20614	689.78
10	1	25.24±0.78	20618	833.26
100	1	26.13±2.66	1814	9056.58

Table 1: Results for various HOP tests

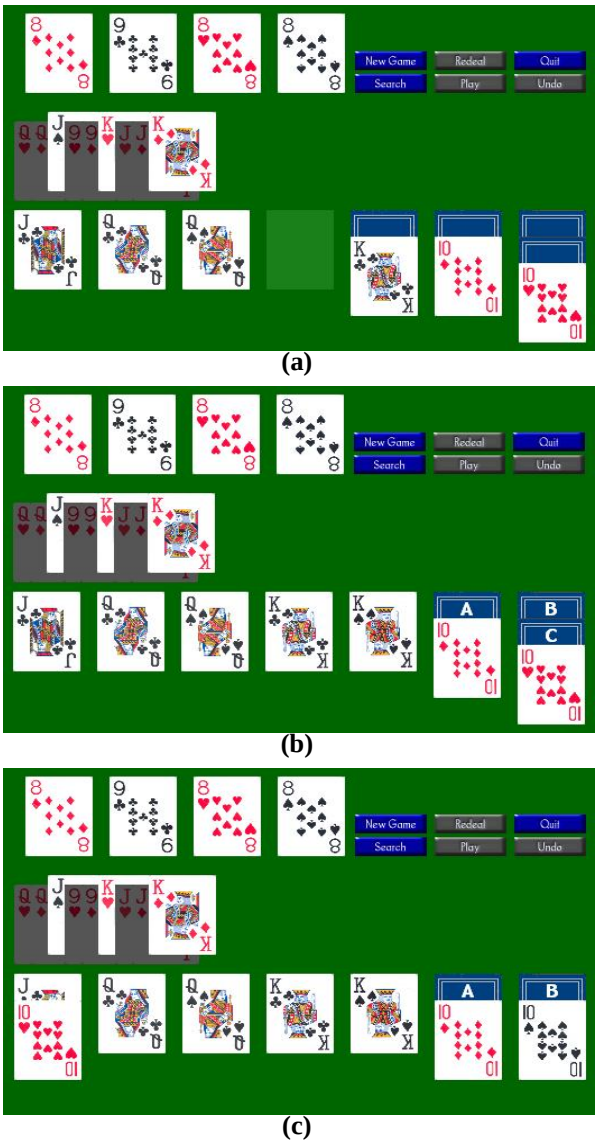
even outperforming the "Las Vegas" standard of 20%. From these results, it appears to be more effective to increase the sampling amount than increasing the search rate.

Averaging Over Clairvoyance

One potential problem with HOP is that it can be overly optimistic in some cases due to the determinization process. As described in (Yoon et al. 2008) the degree to which this optimism is detrimental can be impacted by the type of determinization process used. In particular, the strongest guarantees were shown for determinization processes that produced independent futures. Unfortunately, the determinization process we used for Klondike, which was selected to make our Thoughtful Solitaire planner applicable, does not have this property. Because of this, there is reason to believe that the HOP approach can fall into some of the pitfalls associated with being overly optimistic. In particular, below we show an example where our HOP approach will select an action that leads to eventual failure, even when there are actions available that will lead to certain success. This type of failure is discussed in (Russell and Norvig 1995) as a downfall of averaging over clairvoyance.

In order to validate this effect, we have designed a simple instance where the deterministic planner may optimistically choose an action that can lead to a state where the goal can be reached only by random guessing. We illustrate this effect in Figure 2. From Figure 2**a**, the following sequence of actions will lead to a state in which a win is guaranteed, independent of the unseen identities of the four face-down cards: K♢ →T4, Q♣ →K♢ (leaving T2 empty), J♢ →Q♣, J♡ →Q♠, K♡ →T2, Q♠ →K♡ (leaving T3 empty), 9♢→8♢, 9♡→8♡, 10♢→9♢, 10♡→9♡, K♣ →T3. Alternatively, choosing K♣ →T4 from 2**a** may lead to a state (shown in 2**b**) in which a guaranteed solution exists only in the case that the identity of all cards is known to the planner. By guessing incorrectly from 2**b**, the planner may reach the dead end shown in 2**c**. Because the planner fixes the location of the cards prior to searching for a solution, the future states are correlated, and both of these initial actions (K♢→T4 and K♣→T4) will be valued equally, despite the fact that K♢→T4 is clearly preferred in this situation.

Unfortunately, it is not clear how to implement HOP trajectories in Klondike with independent futures with the provided value function and search mechanism. Because of these possible problems we would like to investigate alternative methods that do not rely so heavily on deterministic planners.



(a) **(b)** **(c)**

Figure 2: **(a)** A state in Klondike Solitaire. **(b)** A possible state after K♣ →T4. **(c)** A dead end forced by having to guess.

UCT

UCT is a Monte-Carlo planning algorithm (Kocsis and Szepesvári 2006), which extends recent algorithms for multi-armed bandit problems to sequential decision problems including general Markov Decision Processes and games. Most notably UCT has received recent stature as the premiere computer algorithm for the game of Go (Gelly and Silver 2007), resulting in huge advances in the field. Given a current state, UCT selects an action by building a sparse look-ahead tree over the state-space with the current state as the root, edges corresponding to actions and their outcomes, and leaf nodes corresponding to terminal states. Each node in the resulting tree stores value estimates for each of the available actions, which are used to select the next action to

be executed. UCT is distinct in the way that it constructs the tree and estimates action values. Unlike standard minimax search and sparse sampling (Kearns, Mansour, and Ng 2002), which typically build depth bounded trees and apply evaluation functions at the leaves, UCT does not impose a depth bound and does not require an evaluation function. Rather, UCT incrementally constructs a tree and updates action values by carrying out a sequence of Monte-Carlo rollouts of entire decision making sequences starting from the root to a terminal state. The key idea behind UCT is to intelligently bias the rollout trajectories toward ones that appear more promising based on previous trajectories, while maintaining sufficient exploration. In this way, the most promising parts of the tree are grown first, while still guaranteeing that an optimal decision will be made given enough rollouts.

It remains to describe how UCT conducts each rollout trajectory given the current tree (initially just the root node) and how the tree is updated in response. Each node s in the tree stores the number of times the node has been visited in previous rollouts $n(s)$, the number of times each action a has been explored in s in previous rollouts $n(s,a)$, and a current action value estimate for each action $Q_{UCT}(s,a)$. Each rollout begins at the root and actions are selected via the following process. If the current state contains actions that have not yet been explored in previous rollouts, then a random action is chosen from among the unselected actions. Otherwise if all actions in the current node s have been explored previously then UCT selects the action that maximizes an upper confidence bound given by

$$Q^{\oplus}_{UCT}(s,a) = Q_{UCT}(s,a) + c\sqrt{\frac{\log n(s)}{n(s,a)}}, \qquad (1)$$

where c is a constant that is typically tuned on a per domain basis, which was set to $c = 1$ in our Solitaire domain. After selecting an action, it is simulated and the resulting state is added to the tree if it is not already present. This action selection mechanism is based on the UCB bandit algorithm and attempts to balance exploration and exploitation. The first term rewards actions whose values are currently promising, while the second term adds an exploration reward to actions that have not been explored much and goes to zero as an action is explored more frequently.

Finally, after the trajectory reaches a terminal state the reward for that trajectory is calculated to be 0 or 1 depending on whether the game was lost or won. The reward is used to update the action value function of each state along the generated trajectory. In particular, the updates maintain the counters $n(s,a)$ and $n(s)$ for visited nodes in the tree and update $Q_{UCT}(s,a)$ for each node so that it is equal to the average reward of all rollout trajectories that include (s,a) in their path. Once the desired number of rollout trajectories have been executed UCT returns the root action that achieves the highest value.

Results Performance results for UCT are presented in Table 2 in a later section for direct comparison to the performance of other methods. These results far surpass the results attained by HOP, winning over 34% of games. This is somewhat surprising considering our UCT implementation does not utilize prior domain knowledge as it explores and builds the stochastic search tree.

Combining UCT with HOP

While the UCT results showed significant improvement over HOP using existing deterministic Solitaire planners, performance appears to have leveled off with the number of trajectories. This led us to consider a new Monte-Carlo approach, HOP-UCT, which aims to explore the potential for combining HOP and UCT. As already noted, one of the potential shortcomings of our earlier HOP experiments was that the deterministic planners required correlated futures which can lead to poor decisions in Solitaire. This motivates trying to develop a HOP-based approach that can operate with independent futures, where the outcomes of each state-action pair at each time step are drawn independently of one another. This can be done in a natural way by using UCT as a deterministic planner for HOP.

To understand the algorithm note that an independent future can be viewed as a deterministic tree rooted at the current state. Each such tree can be randomly constructed by a breadth-first expansion starting at the root that samples a single child node for each action at a parent node. The expansion terminates at terminals. Given a set of such trees one could consider running UCT on each of them and then averaging the resulting action values across trees, which would correspond to HOP with UCT as the base planner. Unfortunately each such tree is exponentially large making the above approach impractical.

Fortunately it is unnecessary to explicitly construct a tree before running UCT. In particular, we can exactly simulate the process of first sampling a deterministic tree and then running UCT by lazily constructing only the parts of the deterministic tree that UCT encounters during rollouts. This idea can be implemented with only a small modification to the original UCT algorithm (Kocsis and Szepesvári 2006). In particular, during the rollout trajectories whenever an action a is taken for the first time at a node s we sample a next node s' and add it as a child as is usually done by UCT. However, thereafter whenever a is selected at that node it will deterministically transition to s'. The resulting version of UCT will behave exactly as if it were being applied to an explicitly constructed independent future. Thus, the overall HOP-UCT algorithm runs this modified version of UCT for a specified number of times, averages the action-values of the results and selects the best action.

Ensemble-UCT The HOP process of constructing UCT trees and combining the results begs the question of whether other ensemble style methods will be successful. We constructed an Ensemble-UCT method that generates UCT trees and averages the values of the actions at the root node in a similar manner to HOP-UCT. We would expect Ensemble-UCT to require fewer trees than HOP-UCT to achieve a similar reduction in variance of the estimated action values. In comparing HOP-UCT and Ensemble-UCT the total number of simulated trajectories becomes a basic cost unit and we will be able to gauge the benefit of spending trajectories on new or existing UCT trees.

Results Comparing the performance of HOP-UCT and UCT trials (seen in Table 2) suggests that sampling multi-

ple UCT trees boosts performance and decreases computing time compared to UCT trees with an equivalent number of total trajectories. We compare the 2000 trajectory UCT and the 100×20 HOP-UCT trials which both utilize a total of 2000 trajectories. Not only does the HOP-UCT approach slightly outperform the UCT method, it requires less than one third the time to do it. The performance of the 200×20 Ensemble-UCT trials also illustrate the trade off between performance and time complexity, which averages a higher winning percentage than other methods. It is faster and more successful than the 2000 trajectory UCT method and the 1000×5 HOP-UCT method. However, it is still within the 99% confidence interval of the 100×20 trajectory HOP-UCT method, which requires far less computing time.

Sparse UCT

One observation we made regarding HOP-UCT was that the time required per rollout trajectory was significantly less than the time for regular UCT. The primary reason for this is that the time for rollouts in regular UCT is dominated by the time to sample a next state given a current state and action. While this is generally a fast process, for example, in Klondike requiring that we keep track of unseen cards and randomly draw one, the time per trajectory is linearly related to the sampling time, making the cost of sampling very significant. The modified version of UCT used for HOP-UCT only required sampling a new state the first time an action was selected at a node and thereafter no sampling was required, which lead to a significant speedup.

This motivated us to consider a new variant of UCT called Sparse UCT, which limits the number of calls to the sampling process at each node in the tree to a specified sampling width w. In particular, the first w times an action a is selected at a node s the usual sampling process is followed and the resulting children are added to the tree. However, thereafter whenever action a is selected at a node one of the already generated w children is selected at random. This random selection from w existing children is generally significantly faster than calling the sampling process and thus can lead to a speedup when w is small enough. However, for very small w the approximation to the original UCT algorithm becomes more extreme and the quality of decision making might degrade. Note that when $w = 1$ the behavior of Sparse UCT is equivalent to that of HOP-UCT. The method for building a Sparse UCT tree is outlined in Algorithm 1. This method can also be extended to Ensemble-Sparse-UCT to evaluate root action values based on averaged values from Sparse-UCT trees.

In addition to improved rollout times, there is another potential benefit of Sparse UCT for problems where the number of possible stochastic outcomes for the actions is large. In such problems, UCT will rarely repeat states across different rollouts. For example, if an action has a large number of uniformly distributed outcomes compared to the number of rollouts then it is unlikely that many of those outcomes will be visited more than once. As a result UCT will not accumulate useful statistics for nodes in the tree, leading to poor action-value estimates. Sparse UCT puts an upper

Input: $s =$ initial state
$y =$ # of trajectories that generate uct tree
$w =$ # sampling width
Output: values for each action in s

$s_0 = s$;
for i = *1 to y* **do**
  $s = s_0$;
  while *not* s*.win AND not* s*.dead-end* **do**
    if *all actions of* s *have been sampled* **then**
      a = arg $\max_a Q^{\oplus}_{UCT}(s,a)$;
    else
      a = random unsampled action from s
    if s*.childCount[*a*]* == w **then**
      s' = randomly choose existing child of (s,a);
    else
      s' = transition(s,a);
      new child for (s,a) = s';
      s.childCount[a]++;
    $s = s'$;
  update all visited (s_i,a_j) pairs with (s.win ? 1 : 0);

Algorithm 1: Generate UCT Tree Algorithm

limit on how many children can be generated for each action at a node and thus will result in nodes to be visited repeatedly across rollouts accumulating non-trivial statistics. However, as the sampling width w becomes small the statistics are based on coarser approximations of the true problem, leading to a fundamental trade-off in the selection of w. Below we consider what theoretical guarantees can be made regarding this trade-off.

Results The experimental (shown in Table 2) results seem to indicate that there is little difference between UCT and Sparse UCT when comparing equivalent numbers of samples per UCT tree. A particularly informative example of this can be seen in the progression of four experiments that build a UCT tree with 1000 trajectories. There appears to be a decreased performance if the UCT tree has a sampling width of 1, but even with a small sampling width of 5, performance increases to within the confidence interval of the experiments with sampling width of 10 and infinity (the latter reported in the UCT section of Table 2). These results would seem to indicate that the number of outcomes associated with each action is not a significant limiting factor for UCT in Solitaire. Also interesting is the increase in time required to generate each tree. Experiments with a sampling width of infinity at most double the time of those experiments with sampling width of 1. This may be explained, in our experiments by the cost of generating random sequences, which will be reduced with the frequent re-visits of existing states in those trees with small sampling width.

Analysis of Sparse UCT

The original UCT algorithm has the theoretical property that its probability of selecting a non-optimal action at a state decreases as poly $\left(\frac{1}{t}\right)$ where t is the number of UCT trajectories. Here we consider what can be said about our UCT variants. We consider finite horizon MDPs, with a horizon of

UCT				
#traj /tree	samp. width	Win Rate (99% conf.)	# Games	Av. sec / Game
100	inf	24.64±3.04	1331	44.50
1000	inf	34.24±4.52	733	1042.56
2000	inf	34.41±4.01	933	2980.44
HOP-UCT (sampling width = 1)				
#traj /tree	#tree /dec	Win Rate (99% conf.)	# Games	Av. sec / Game
100	20	35.47±4.00	953	767.57
1000	5	30.58±4.53	690	3531.12
Sparse UCT				
#traj /tree	samp. width	Win Rate (99% conf.)	# Games	Av. sec / Game
100	1	22.65±2.41	2000	40.11
1000	1	28.66±4.21	764	566.65
1000	5	33.98±1.63	5583	495.68
1000	10	33.79±3.11	1536	747.82
2000	4	35.62±2.58	2288	1485.19
Ensemble-UCT (sampling width = inf)				
#traj /tree	#tree /dec	Win Rate (99% conf.)	# Games	Av. sec / Game
200	20	36.97±1.92	4206	2280.55
100	10	32.86±1.57	5986	669.90
Ensemble-Sparse-UCT (sampling width = 2)				
50	50	35.27±2.16	3249	1302.38

Table 2: Results on various UCT algorithms

D, and for simplicity restrict to the case where the range of the reward function is $[0, 1]$. Our first variant, Sparse UCT, is identical to UCT only it considers at most w outcomes of any state action pair when constructing the tree from the current state, which limits the maximum tree size to $O((wk)^D)$ where k is the number of actions.

To derive guarantees for Sparse UCT, we draw on ideas from the analysis of the sparse sampling MDP algorithm (Kearns, Mansour, and Ng 2002). This algorithm uses a generative MDP model to build a sparse expectimax tree of size $O((wk)^D)$ rooted at the current state s and computes the Q-values of actions at the root via expectimax search. Note that the tree construction is a random process, which defines a distribution over trees. The key contribution of that work was to show bounds on the sampling width w that guarantee near optimal Q-values at the root of a random tree with high probability. While the analysis in that paper was for discounted infinite horizon MDPs, as shown below, the analysis extends to our finite horizon setting.

For the purposes of analysis, consider an equivalent view of Sparse UCT, where we first draw a random, expectimax tree as in sparse sampling, and then run the original UCT algorithm on this tree for t trajectories. For each such tree there is an optimal action, which UCT will select with high probability as t grows, and this action has some probability of differing from the optimal action of the current state with respect to the true MDP. By bounding the probability that such a difference will occur we can obtain guarantees for Sparse UCT. The following Lemma provides such a bound by adapting the analysis of sparse sampling. In the following we will define $Q^*_d(s, a)$ to be the optimal action value function with d stages-to-go of the true MDP M. We also define $T_d(s, w)$ to be a random variable over sparse expectimax trees of depth d, derived from the true MDP, and rooted at s using a sampling width of w. Furthermore, define $\hat{Q}^w_d(s, a)$ to be a random variable that gives the action values at the root of $T_d(s, w)$.

Lemma 1. *For any MDP with finite horizon D, k actions, and rewards in $[0, 1]$, we have that for any state s and action a, $\|Q^*_d(s, a) - \hat{Q}^w_d(s, a)\| \leq d\lambda$ with probability at least $1 - d(wk)^d \exp\left(-\frac{\lambda^2}{D^2} w\right)$.*

This shows that the probability that a random sparse tree leads to an action value estimate that is more than $D\lambda$ from the true action-value decreases exponentially fast in the sampling width w (ignoring polynomial factors). We can now combine this result with one of the original UCT results. In the following we denote the error probability of Sparse UCT using t trajectories and sampling width w by $P_e(t, w)$, which is simply the probability that Sparse UCT selects a sub-optimal action at a given state. In addition we define $\Delta(s)$ to be the minimum difference between an optimal action value and sub-optimal action value for state s in the true MDP, and define the minimum Q-advantage of the MDP to be $\Delta = \min_s \Delta(s)$.

Theorem 1. *For any MDP with finite horizon D, k actions, and rewards in $[0, 1]$, if*

$$w \geq 32 \frac{D^4}{\Delta^2} \left(D \log \frac{16kD^5}{\Delta^2} + \log \frac{D}{\delta} \right)$$

then $P_e(t, w) \leq$ poly $\left(\frac{1}{t}\right) + \delta$.

Proof. (Sketch) Theorems 5 and 6 of (Kocsis and Szepesvári 2006) show that for any finite horizon MDP the error rate of the original UCT algorithm is $O(t^{-\rho(\Delta)^2})$ where ρ is a constant. From the above lemma if we set λ equal to $\frac{\Delta}{4D}$ we can bound the probability that the action-values of a randomly-sampled sparse tree are in error by more than $\frac{\Delta}{4}$ by $D(wk)^D \exp\left(-\left(\frac{\Delta}{4D^2}\right)^2 w\right)$. It can be shown that our choice of w bounds this quantity to δ. Note that this bounds the probability that the minimum Q-advantage of the sparse tree is greater than $\frac{\Delta}{2}$ by δ. The UCT result then says that for trees where this bound holds the error probability is bounded by a polynomial in $\frac{1}{t}$. The theorem follows by applying the union bound. □

This result shows that for an appropriate value of w, Sparse UCT does not increase the error probability significantly. In particular, decreasing the error δ due to the sparse sampling requires an increase in w that is of only order $\log \frac{1}{\delta}$. Naturally, since these are worst case bounds, they are almost always impractical, but they do clearly demonstrate that the required value of w does not depend on the size of the MDP state space but only on D, k, and Δ. It is important to note that there is an exponential dependence on

the horizon D buried in the constants of the UCT term in the above bound. This dependence is unavoidable as shown by the lower-bound in (Kearns, Mansour, and Ng 2002).

Summary of Results

This work presents the results of a broad family of algorithms. UCT systematically builds a deep search tree, sampling outcomes at leaf nodes and evaluating the result. HOP solves several sampled determinized problems to approximate the value of root-level actions. Similarly, HOP-UCT and Ensemble-UCT sample several UCT trees, to approximate the same action values. In the case of HOP-UCT, the trees are determinized by restricting the UCT tree to a single outcome for each action and state. Sparse UCT represents a compromise between a determinized UCT and a full UCT tree. In Klondike Solitaire, the UCT based methods have been shown to be significantly more successful than HOP.

Our results suggest some general conclusions for this family of UCT algorithms. Performance increases results from increases in 1) the number of trajectories used to generate a UCT tree, 2) the sampling width of the UCT trees and 3) the number of UCT trees used to approximate the action values. Performance can be optimized by balancing these variables in the context of the available sampling time. For Klondike it appears that the time required to construct a UCT tree is disproportionally longer for trees with a larger number of trajectories. We observe improved performance and time complexity for HOP-UCT, Ensemble-UCT and Ensemble-Sparse-UCT methods compared to simple UCT trees generated with a similar number of trajectories. The presented algorithms can be unified under a general framework of UCT-based algorithms parameterized by the number of UCT trees and the sampling width. Future work includes a more thorough exploration of this algorithm space and empirical evaluation in multiple probabilistic planning domains.

Conclusion and Discussion

To the best of our knowledge our results represent the first non-trivial empirical bounds on the success rate of a policy for Klondike Solitaire. The results show that a number of approaches based on UCT, HOP, and sparse sampling hold promise and solve up to 35% of random games with little domain knowledge. These results more than double current estimates regarding human level performance. A better theoretical understanding of why these algorithms are so successful would be very valuable. We were surprised that our method incorporating a sophisticated deterministic search method was handily outperformed by UCT. We expect that the results can be much improved by adding domain knowledge and learning.

Many real world domains such as real-time stochastic scheduling have similar characteristics as Solitaire. For example, in fire and emergency rescue, there are exogenous events such as fires that need to be responded to in a timely manner. There are a variety of policy constraints such as what kind of emergencies can be responded to with different kinds of equipment, and where the resources should return after the service. In addition to the optimal response problem, there are also problems of deciding where to house the resources and how often they might be moved. Our preliminary approach to this problem based on multi-level hill climbing search is described in (Bjarnason et al. 2009). We believe that approaches such as HOP, UCT, and Sparse UCT hold promise here and deserve to be investigated.

Much of probabilistic planning is currently focused on artificial domains designed by researchers. Unfortunately they do not bring out some of the representational issues that are readily apparent in natural domains, for example, the problem of having to represent uniform distributions over variable number of objects in Solitaire or the difficulty of representing the dynamics of a robot arm. We hope that encounters with real world domains might encourage researchers to consider novel problem formulations such as planning with inexact models or using simulators in the place of models.

Acknowledgements We gratefully acknowledge the support of the Army Research Office under grant number W911NF-09-1-0153. We thank the reviewers for their thorough reviews and helpful suggestions.

References

Bjarnason, R.; Tadepalli, P.; Fern, A.; and Niedner, C. 2009. Simulation-based optimization of resource placement and emergency response. In *Innovative Applications of Artificial Intelligence (IAAI-2009), to appear*.

Bjarnason, R.; Tadepalli, P.; and Fern, A. 2007. Searching solitaire in real time. *International Computer Games Association Journal* 30(3):131–142.

Diaconis, P. 1999. The mathematics of Solitaire, www.uwtv.org/programs/displayevent.aspx?rid=1986&fid=571 .

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in UCT. In *Proceedings of the International Conference on Machine Learning*, 273–280.

Kearns, M.; Mansour, Y.; and Ng, A. 2002. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning* 49:193–208.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *15th European Conference on Machine Learning*, 282–293.

Levin, J. 2008. Solitaire-y confinement, www.slate.com/id/2191295.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence, A Modern Approach*. Upper Saddle River, New Jersey 07458: Prentice Hall.

Yan, X.; Diaconis, P.; Rusmevichientong, P.; and Van Roy, B. 2005. Solitaire: Man versus machine. In *NIPS 17*, 1553–1560.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *AAAI-2008*, 1010–1016.

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-replan: A baseline for probabilistic planning. In *International Conference on Automated Planning and Scheduling (ICAPS-2007)*, 352–359.