Offline Monte Carlo Tree Search for Statistical Model Checking of Markov Decision Processes

Michael W. Boldt, Robert P. Goldman, David J. Musliner,
Smart Information Flow Technologies (SIFT)
Minneapolis, USA
email: {mboldt,rpgoldman,dmusliner}@sift.net

Abstract

To find the optimal policy for large Markov Decision Processes (MDPs), where state space explosion makes analytic methods infeasible, we turn to statistical methods. In this work, we apply Monte Carlo Tree Search to learning the optimal policy for a MDP with respect to a Probabilistic Bounded Linear Temporal Logic property. After we have the policy, we can proceed with statistical model checking or probability estimation for the property against the MDP. We have implemented this method as an extension to the PRISM probabilistic model checker, and discuss its results for several case studies.

Introduction

Model-checking has been very successful in identifying and removing faults in conventional hardware and software. For small problems, dynamic programming methods can give exact analytic results. For large problems, where state explosion makes dynamic programming infeasible, we turn to statistical methods.

We bring two contributions to the field of statistical model checking:

Time-sensitive policies One weakness of state of the art statistical model checking methods is that they typically only consider memoryless policies. However, bounded-time properties in general have time-sensitive optimal policies. Our approach takes time into consideration when learning the optimal policy.

Smarter sampling Seeing the success of Monte Carlo Tree Search (MCTS) in fields like computer Go, we bring MCTS to the field of statistical model checking. State of the art MCTS methods do a good job balancing exploration of the model with the exploitation of actions which look promising. We use this property of MCTS methods to take smarter samples while learning the optimal policy for a model with respect to the property of interest.

Background

Markov Decision Processes

Definition 1 *State-Labeled Markov Decision Process (MDP):*[1] *a tuple* $\mathcal{M} = \langle S, \bar{s}, A, \tau, \mathcal{L} \rangle$*, where*

- S *is the set of* states*;*
- $\bar{s} \in S$ *is the* initial state*;*
- A *is a finite set of* actions*;*
- $\tau : S \times A \times S \to [0, 1]$ *is a transition function such that for each* $s \in S$ *and* $a \in A$*, either* $\sum_{s' \in S} \tau(s, a, s') = 1$ *(*a *is enabled) or* $\sum_{s' \in S} \tau(s, a, s') = 0$ *(*a *is disabled), and* $\forall s \in S$ *there is at least one enabled action;*
- $\mathcal{L} : S \to 2^{\lambda}$ *is a labeling function, mapping each state to the set of atomic propositions (*λ*) true in that state.*

Definition 2 *Memoryless policy (scheduler) of an MDP: a resolution of the nondeterminism over the actions in each state, depending only on the current state. That is, a function* $\sigma : S \times A \to [0, 1]$ *such that* $\sum_{a \in A} \sigma(s, a) = 1$ *and* $\sigma(s, a) > 0 \Rightarrow \tau(s, a) > 0$*, where* $a \in A$ *and* $s \in S$*.*

The optimal policy for a time-bounded property of an MDP is not necessarily memoryless, as the optimal policy can depend on the time left. See Figure 1 for a simple, concrete example.[2] So, we consider time-sensitive policies.

Definition 3 *Time-sensitive policy (scheduler) of an MDP: a resolution of the nondeterminism over the actions in each state, depending on the current state and the number transitions taken so far. That is, a function* $\sigma : \mathbb{N} \times S \times A \to [0, 1]$ *such that* $\sum_{a \in A} \sigma(t, s, a) = 1$ *and* $\sigma(t, s, a) > 0 \Rightarrow \tau(s, a) > 0$*, where* $t \in \mathbb{N}$ *is the count of transitions taken,* $a \in A$ *and* $s \in S$*.*

A *deterministic policy* is a policy where the range of σ is $\{0, 1\}$. It is known that if there is an optimal policy for a MDP, there is an optimal deterministic policy.

For the purposes of verification, we wish to assess claims in *probabilistic bounded-time linear temporal logic*.

Definition 4 *Bounded-Time LTL Syntax:*

$$\phi := p \mid \neg\phi \mid (\phi \vee \phi) \mid F^{\leq n}\phi \mid G^{\leq n}\phi \mid (\phi U^{\leq n} \phi)$$

where:

- $p \in \mathcal{L}$ *is a set of propositions;*
- n *is a positive integer;*

[1] Definition follows (Henriques *et al.* 2012).

[2] We note that, equivalently, if one can modify the model, one can add a time counter to the state space of the model.

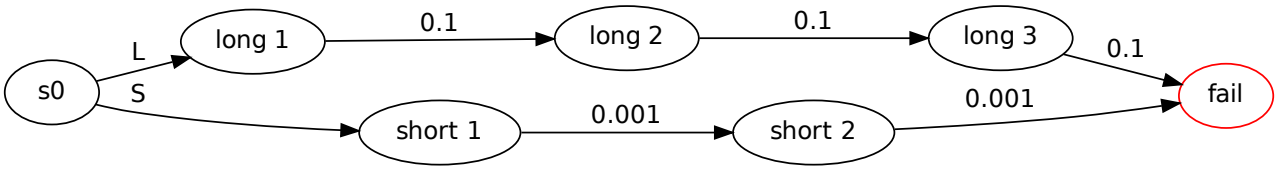

Figure 1: A simple partial model with a time-sensitive optimal policy. Starting at state $s0$, with the goal of reaching `fail`, with available actions L and S, the optimal policy is to take action L if there are 3 or more time units left, and S if there are 2 time units left.

- F *is the* eventually *modal operator, meaning* ϕ *will be satisfied at some point before* n *steps;*
- G *is the* always *modal operator, meaning* ϕ *will remain true for all of the next* n *steps;*
- U *is the* until *modal operator, meaning the expression on the left will remain true (at least) until the expression on the right becomes true.*

For more details on LTL, see, e.g., (Emerson 1990).

Probabilistic Bounded LTL (PBLTL) adds a probability modality, P, so assertions are of the form, $P(\phi) \leq \psi$ where ϕ is a Bounded-Time LTL expression and ψ is a probability threshold For example, $P(F^{\leq 12}(\text{fail} \vee \text{deadlock})) \leq 0.1$ can be read: "there is less than a 10% chance of reaching a failure or deadlock state within 12 time steps." In most cases, the verification challenge is to prove that the system will not reach an unsafe state, within a given time, and with greater than a certain probability.

Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) algorithms build a search tree using random samples to heuristically find optimal decisions in a given domain (Browne *et al.* 2012). MCTS has found much success in fields like computer Go. Inspired by its success in these related fields, we apply MCTS to sampling large MDPs.

MCTS involves four stages:

Select Start at the root node of the tree, and use a policy called the *Tree Policy* to choose actions until it finds a node with a child that is not yet in the tree.

Expand Add a child to the tree based on the available actions on the selected node.

Simulate Take a sample of the domain from the newly expanded node, using a policy called the *Default Policy* to choose actions.

Backup Propagate the result of the sample through the tree nodes it followed.

Upper Confidence Bound for Trees (UCT) is the most popular MCTS algorithm. It treats the choice of action in the Tree Policy as a multi-armed bandit problem (Mahajan & Teneketzis 2008), using the UCB1 equation to choose which action to execute in a given state. The equation is:

$$UCT = \arg\max_{i \in A} \overline{X}_i + 2C_p \sqrt{\frac{2 \ln n}{n_i}}$$

where

- $\overline{X}_i$ is the mean reward[3] from sampling action i so far;
- n is the number of times the current node has been visited;
- n_i is the number of times child node i has been visited;
- C_p is a constant exploration factor, for which the value $1/\sqrt{2}$ satisfies the Hoeffding inequality when rewards are in the range $[0, 1]$.

This equation balances exploitation of actions with relatively high observed reward against exploration of actions which have not been visited much. The first term, $\overline{X}_i$, tries to exploit actions with high rewards so far. n_i in the denominator of the second term tries to explore actions which have been under-explored.

Statistical Model Checking MDPs

For small MDPs, one can use dynamic programming (like value iteration) for model checking PBLTL properties. For large MPDs, however, state space explosion makes dynamic programming infeasible, so we must turn to statistical methods, like those found in (Legay, Delahaye, & Bensalem 2010).

Kearns *et al.* use statistical methods to find a near-optimal policy for infinite-horizon (discounted) MDPs with respect to maximizing rewards (Kearns, Mansour, & Ng 2002). Their method involves sampling each action a large, constant number of times, based upon the desired accuracy. Lassaigne and Peyronnet use Kearns' method to find a policy, then use that policy to perform statistical verification on linear temporal properties (Lassaigne & Peyronnet 2012). The complexity of these methods are not affected by the size of the MDP, but are exponential in the time horizon.

One of our research goals is to find a near-optimal policy with fewer samples, by using MCTS to choose actions instead of sampling every action and relying on the law of large numbers.

Henriques *et al.* introduced statistical model checking of PBLTL properties of MDPs (Henriques *et al.* 2012). Their algorithm alternates between using Q-learning to improve upon its policy with respect to the property, and performing statistical model checking on the learned policy. If the policy is *good enough* to (dis)prove the property, the algorithm stops and the property is (dis)proven. Otherwise, it might be the case that it has not learned the optimal policy yet, so it tries to improve the policy by sampling the MDP more.

[3] With respect to our PBLTL properties, a reward is 1 if the property is satisfied in a sample, and 0 if it is not.

One shortcoming of that method is that it only considers memoryless policies. In general, PBLTL properties require a policy with memory, as they can depend on the time left. See Figure 1 for an example.

Approach

Our approach uses UCT to learn a policy for an MDP, trying to maximize a given PBLTL property, and uses that policy for statistical model checking.

To learn the policy, we build a MCTS tree where each node represents a state and a particular time, and each edge represents a transition to a state for the next time step. Since this is an MDP, each move includes both a non-deterministic action and a probabilistic transition.

The root of the tree corresponds to the initial state of the model at time zero. From there, we use the UCT algorithm for the *select* and *expand* steps of MCTS. For the *simulate* step we use a uniform random action. For the *backup* step, we update two pieces of data on each node in the path: (1) increment n—the number of times the node has been visited, and (2) if the sample satisfies the property, increment r—a cumulative rewards counter. From this data, we can calculate the mean reward from sampling for a node, simply r/n.

Currently, our algorithm takes the number of learning samples/expansions to perform as an input parameter. After it takes the given number of learning samples, it stops expanding the tree and starts taking samples for statistical model checking. It chooses actions for these samples greedily in the MCTS tree, i.e., the action which maximizes expected value. Formally, it chooses the action:

$$\arg\max_{a \in A} \sum_{s' \in C_a} \tau(s, a, s') \frac{r_{s'}}{n_{s'}}$$

where

- A is the set of actions,
- s is the current state,
- C_a is the set of children of the current MCTS node reachable by action a,
- $r_{s'}$ is the cumulative rewards in the MCTS node corresponding to the child node corresponding to s',
- $n_{s'}$ is the number of times the child node corresponding to s' has been visited.

Note that it may be the case that the tree does not cover all states encountered by samples. When a sample falls off the tree, if there is another node in the tree with the same underlying model state, we use the greedy action in that node. If no node has the same state, we fall back on a uniform random action. In future work, we would like to analyze the frequency and effects of falling off the tree more thoroughly.

With these samples from the greedy policy, we can use standard statistical model checking techniques to evaluate probabilistic queries about the property. See (Legay, Delahaye, & Bensalem 2010) for an overview of statistical model checking methods.

Name	Backoff	States	Transitions
wlan0	0	6,063	1,0619
wlan1	1	10,978	2,0475
wlan2	2	28,598	57,332
wlan3	3	96,420	204,744
wlan4	4	345,118	762,420
wlan5	5	1,295,336	2,930,128
wlan6	6	5,007,666	11,475,916

Figure 2: Sizes of Wireless LAN models.

Experiments

We have implemented our approach and evaluated it on several domains.

Implementation

We implemented our approach as extensions to the PRISM probabilistic model checker (Kwiatkowska, Norman, & Parker 2011). PRISM includes a discrete-event simulation engine, which we use to traverse MDP models for sampling.

For our properties, we use the PRISM extension from (Henriques *et al.* 2012) which adds support for checking satisfaction of PBLTL properties to PRISM.

As an optimization, we collapse identical states at the same depth in the tree (i.e., at the same time step). This saves memory since there is only one copy, and improves accuracy by aggregating results across different paths to the same state.

Aside from this one improvement, we did not attempt to optimize the CPU or memory consumption of our implementation; we leave optimization and performance analysis to future work.

Results

To study the effectiveness of our algorithm, we varied the number of samples used for policy learning, and used the learned policy to compute the expected value of the probability of the property using 1000 samples.

Wireless LAN This domain is from a PRISM case study of the IEEE 802.11 Wireless LAN protocol (Kwiatkowska, Norman, & Sproston 2002). The protocol involves a randomized exponential backoff for retransmission of collided packets, to avoid repeated collisions. To scale up the size of the domain, we increase the maximum backoff value from 0 to 6. See Figure 2 for the size of the models in this domain.

The property we analyze is the probability of having two collisions within 100 time steps. The underlying probability of this property is the same across the varying maximum backoff parameter.

The results for this model are shown in Figure 3. We see that 100,000 learning samples are sufficient to get very close to the underlying probability, regardless of model size. This indicates that, even though the size of the model grows by several orders of magnitude, the MCTS used for learning the policy focuses in on the important parts of the space and finds a good policy.

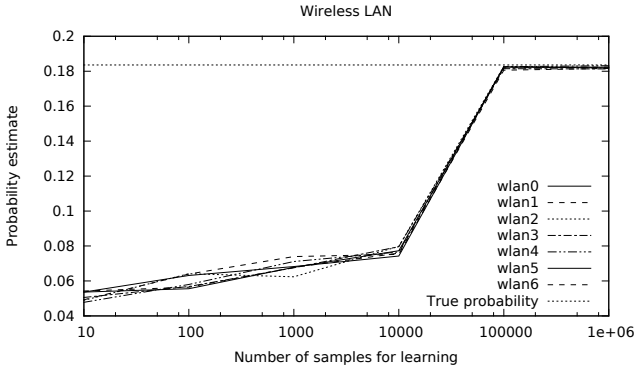

Figure 3: Results for Wireless LAN domain.

Name	Proc., Backoff	States	Transitions
csma2/2	2 , 2	1K	1K
csma2/4	2 , 4	7K	10K
csma2/6	2 , 6	66K	93K
csma3/2	3 , 2	36K	55K
csma3/4	3 , 4	1,460K	2,396K
csma3/6	3 , 6	84,856K	147,200K
csma4/2	4 , 2	761K	1,327K
csma4/4	4 , 4	133,301K	258,474K
csma4/6	4 , 6	[Timed out]	

Figure 4: Sizes of IEEE 802.3 CSMA/CD models.

IEEE 802.3 CSMA/CD This is another network protocol for avoiding collisions from the PRISM case studies (Kwiatkowska *et al.* 2007). In this case, we scale up the model both by the number of processes sending messages over a shared bus, and by the maximum backoff after a collision. See the model sizes in Figure 4. Note that PRISM timed out building the model for the largest example we gave it.

We analyze the property that all processes send all their messages within 100 time steps, with fewer than two collisions. In Figure 5, we show our algorithm against the largest model for which we have the true probability. Even with over 133 million states, our algorithm finds a near-optimal policy in 1 million samples.

IPv4 Zeroconf Protocol The IPv4 Zeroconf Protocol, another PRISM case study example, dynamically configures IP addresses in an ad-hoc network, to avoid setting up static IP addresses for each device or a DHCP server (Kwiatkowska *et al.* 2006). The state size is scaled up by the number of probes it sends out before claiming an IP address. The sizes of the models are in Figure 6.

Our algorithm does not perform well on the Zeroconf models. As seen in Figure 7, even on the smallest model with under 32,000 states, the probability estimate from the learned policy is nearly 0.1 below the true probability. We need further investigation to understand why our algorithm fails to find a near-optimal policy for this model.

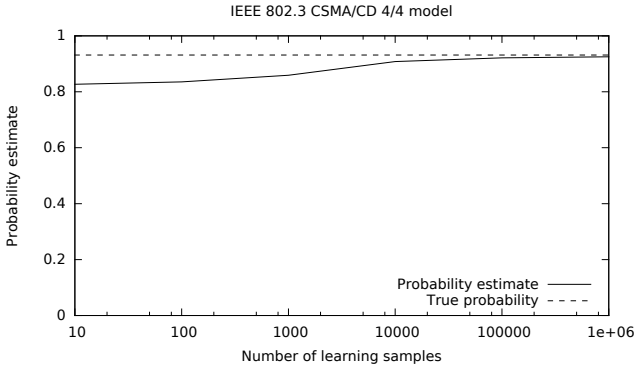

Figure 5: Results for IEEE 802.3 CSMA/CD 4/4 model.

Name	Probes	States	Transitions
zeroconf1	1	31,954	73,318
zeroconf2	2	89,586	207,825
zeroconf4	4	307,768	712,132
zeroconf6	6	798,471	1,833,673
zeroconf8	8	1,870,338	4,245,554

Figure 6: Sizes of IPv4 Zeroconf Protocol models.

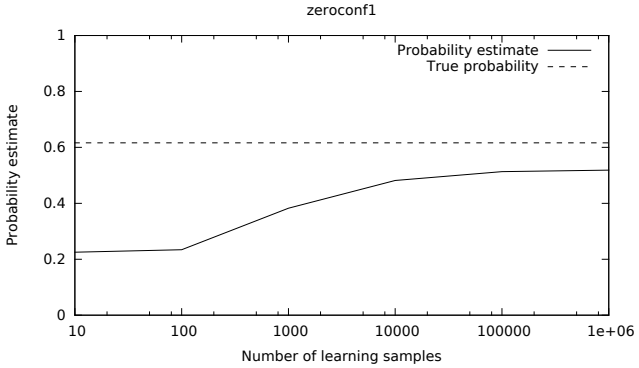

Figure 7: Results for 1-probe IPv4 Zeroconf Protocol model.

Conclusions and Future Work

We have presented and implemented an algorithm using MCTS to learn a near-optimal policy for an MDP for a given PBLTL property. We have collected and presented results for several case studies of varying size and difficulty. Our algorithm works very well for most models, while there is room for improvement on others.

We would like to derive a mathematical bound on the quality of the policy learned via MCTS. We will explore combining the near-optimality guarantees of the policy learning in (Kearns, Mansour, & Ng 2002) with the regret bounds of UCT in (Kocsis & Szepesvári 2006).

Another, perhaps related, improvement would be to dynamically determine when to stop policy learning. Perhaps the bound on the quality will give us a number of samples required for a given confidence level, or we could investigate methods like Wald's sequential probability ration test to tell when to stop sampling (Wald 1945).

We have not optimized the efficiency of our algorithm. Since samples are independent, we could take multiple samples in parallel. We could also look into extracting the optimal policy from the MCTS tree for reuse without recomputing the greedy action for each node.

We are also currently investigating using online MCTS methods for sampling. Here, instead of performing MCTS offline and learning a policy to use for statistical model checking, we use online MCTS to directly take samples for statistical model checking, eliminating policy learning.

Acknowledgments

This research was sponsored by the Air Force Office of Scientific Research and AFRL via FA9550-12-1-0146. In addition, we would like to recognize Dave Parker for his guidance working with PRISM. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

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.

Emerson, E. A. 1990. Temporal and modal logic. In van Leeuwen, J., ed., *Handbook of Theoretical Computer Science (Vol. B)*. Cambridge, MA, USA: MIT Press. 995–1072.

Henriques, D.; Martins, J. G.; Zuliani, P.; Platzer, A.; and Clarke, E. M. 2012. Statistical model checking for Markov decision processes. In *Quantitative Evaluation of Systems (QEST), 2012 Ninth International Conference on*, 84–93. IEEE.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning* 49(2-3):193–208.

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

Kwiatkowska, M.; Norman, G.; Parker, D.; and Sproston, J. 2006. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design* 29:33–78.

Kwiatkowska, M.; Norman, G.; Sproston, J.; and Wang, F. 2007. Symbolic model checking for probabilistic timed automata. *Information and Computation* 205(7):1027–1077.

Kwiatkowska, M.; Norman, G.; and Parker, D. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In Gopalakrishnan, G., and Qadeer, S., eds., *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, 585–591. Springer.

Kwiatkowska, M.; Norman, G.; and Sproston, J. 2002. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In Hermanns, H., and Segala, R., eds., *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, volume 2399 of *LNCS*, 169–187. Springer.

Lassaigne, R., and Peyronnet, S. 2012. Approximate planning and verification for large Markov decision processes. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, 1314–1319. ACM.

Legay, A.; Delahaye, B.; and Bensalem, S. 2010. Statistical model checking: An overview. In *Runtime Verification*, 122–135. Springer.

Mahajan, A., and Teneketzis, D. 2008. Multi-armed bandit problems. In *Foundations and Applications of Sensor Management*. Springer. 121–151.

Wald, A. 1945. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics* 16(2):117–186.