Monte Carlo Tree Search

Zhengnan Xie and Sebastian Thiem

Contents

1 Introduction

This paper has explored the application of Monte Carlo Tree Search (MCTS) in different domains.

MCTS is known for it's application AlphaGo: the Go playing AI, but it is also a powerful tool in handling other games such as: Pac-man, Magic The Gathering, or Settlers of Catan.

This method also expands to other domains outside of games, on example is story generation. MCTS can be used to generate a high-level plot and believable stories with characters interactions.

Because of MCTS's ability to search through spaces with large branching factors, it has also been explored in the domains of information extraction and malware detection on Android devices.

2 Monte Carlo Tree Search

Monte Carlo Tree Search is an algorithm that combines reinforcement learning and tree search into one method.

It will build and explore a search tree, similar to minimax, but on the leaf nodes it runs a Monte Carlo simulation to sample many possible futures after that node. This sampling results in an estimation of the node's value which can then be propagated back up the search tree. By simulating outcomes at random rather than expanding the search space by the branching factor, MCTS will retrieve useful estimations of a state's value without having to dig down the search tree as deep (or as wide) as minimax would.

The algorithm has four key components:

- Selection
  Starting at a root node, a selection policy is recursively applied to descend the tree until the most urgent expandable node is reached. The node is expandable if it is not a terminal state or it is not visited before.
- Expansion
  During expansion, the selected node or nodes will be expanded.
- Simulation
  Simulation is where the playout would come in. The algorithm would simulate a game from the node to the terminal state to produce a reward for the current sequence of actions.
- Backpropagation The reward values are then back propagete to the tree and give each state in the tree a value.

UCT (Upper Confidence Bound applied to Trees) algorithm is used to balance between exploration and exploitation. It will be discussed in the paper review part.

As an anytime algorithm, MCTS would produce the best result for the given limitation of playout (the times the algorithm randomly play the whole game). It does not mean that the more times it runs the better it will get, it might finally level out given the search space is not completely searched through.

However, given the limitations MCTS has, it is still a very powerful algorithm to search through large space and produce a near-optimal result. https://www.youtube.com/watch?v=lhFXKNyA0QA

3 Reviewed Papers

Mastering the game of Go with deep neural networks and tree search (Silver et al. 2016)[1]

This Nature article presents AlphaGo: the first AI for the game Go that was capable of beating a human professional player.

AlphaGo utilized multiple deep neural networks to create policies and generate values to improve the performance of a standard MCTS agent.

The first network is the policy network, and it is trained through a pipeline. The pipeline begins with a supervised training network that learned to predict moves base on the moves of expert Go players. This network was a convolutional network that read in the board as a 19x19 image. Once that network is trained it is further tuned by reinforcement learning. The agent will play games against random previous iterations of it's policies. This step results in policies that are 80% stronger than policies that only rely on supervised training.

The other network is the value network that will predict the value of a state. The authors again use self-play to train the network by playing against the reinforcement learning trained policy network. This network began approaching the accuracy of MC rollouts of the policy network.

The agent can now run as follows

1. Traverse the search tree prioritizing high Q values + $u(P)$ a lookup of prior probabilities for an edge.
2. Expand the leaf node, send it through the policy network, and store the probabilities for each action
3. Evaluate the node with the value network
4. Run a MC rollout based on a fast-rollout policy
5. Update the Q value using the mean value of all evaluations of the rollouts and value network

AlphaGo ran on a 48-CPU 8-GPU server, and there was even a distributed version that ran on 1202 CPUs and 176 GPUs.

AlphaGo was tested against other Go agents and it won 99.8% of games, and won between 77-99% of games when it was given a handicap (let the other agent get 4 free moves). When AlphaGo played against the world champion Go player Fan Hui, it won 5-0.

The authors make a comparison to Deep Blue, the chess algorithm that beat a human professional. Noting that even though Go's search space is 250^150 vs. chess's search space of 35^80, AlphaGo evaluated 1000x fewer positions than Deep Blue. The combined use of the policy and value network resulted in a far more intelligent selection of moves to evaluate.

Real-Time Monte Carlo Tree Search in Ms Pac-Man (Pepels et al. 2012)[2]

The authors of this paper designed and fine tuned a real time Pac-man agent using MCTS as the foundation of the agent's decision making process.

Motivated by a competition; there was a lot of work put into improving the execution time of the algorithm. The maximum execution time for each turn was 40ms.

Their first set of changes sought to narrow down the search space. They compressed the map space down to a set of nodes at each junction and added a feature that prevented the agent from searching down paths that lead back to the parent nodes after the first ply.

In order to reduce the computational requirements of MCTS they implement a method of reusing the search tree each turn. Clearly a lot of good computation time gets wasted by throwing away the search tree every turn as you would with a normal MCTS. Using the same search tree every time is not the best solution either, as the agent is unable to explore as much. So the answer here is to compromise, and this compromise has two components. 1) Establishing a metric that determines whether or not the overall state of the game has changed enough to warrant starting a new search tree (e.g. Pac-man has eaten a ghost). 2) Including a decay factor that weakens the effect of old samples.

The agent was placed in two competitions where Pac-man agents competed with Ghost agents; they came in second and first place. The first competition was a round robin style tournament again other ghost agents, and the other competition paired Pac-man agents against ghost agents using a Glicko-based pairing system. Interestingly the agent that beat them in the first competition was a minimax agent with a good heuristic function.

Ensemble Determinization in Monte Carlo Tree Search for the Imperfect Information Card Game Magic: The Gathering (Cowling et al. 2012)[3]

In this work the authors apply MCTS to a strongly simplified version of Magic: The Gathering(MTG), as a case study to explore MCTS's effectiveness in the domain of complex card games.

The full scope of MTG is currently too vast to be handled by any modern AI, as the game includes over 10,000 different cards each with their own abilities and effects on gameplay. The authors note that the online MTG does have a proprietary AI system, that uses a game tree with static heuristics at the leaf nodes, which has thus-far been unsuccessful in challenging anything more than beginner players. In order to begin to challenge more expert players the game needed to be simplified to a particular sub-set of cards (creatures and basic lands) and on particular win condition (eliminate the other players life points). This reduces the game to the following set of rules:

1. Players each start with 2000 life points
2. Players each have a 60 card deck (Containing creatures cards and basic lands), a discard pile, space on the board for lands, and space for creatures.
3. Decks are limited to 4 copies of a creature card
4. Creatures have power and toughness scores (These scores are in increments of 100)
5. Toughness is considered when a creature is defending an attack; Power for attacking
6. During an attack if the defending creature has less toughness than the attacking creatures power, then the defending creature will be sent to the defending player's discard pile.
7. Creature cards have a cost associated with them and will require "tapping" land cards in order to summon them
8. Cards get "tapped" when they used in a turn, typically denoted by turning them horizontally
9. Unless the card states otherwise, the creatures summoned will have "summoning sickness" which prevents them from fighting on the same turn that they are summoned
10. Players begin the game by shuffling the decks and drawing 7 cards
11. A turn for player A plays out as follows
    (a) "Un-tap" All cards on player A's side of the field
    (b) Draw a card
    (c) Place a land card on the board if the player has one in their hand and chooses to play it
    (d) Place any creature cards on the board if the player has them in their hand, can afford the cost, and chooses to play them
    (e) Player A may chose to attack player B, which involves player A declaring attackers
    (f) Player B Uses their creatures to block the attack(s)
    (g) Damage is calculated and dealt to player B
    (h) (End of Turn)

The reason for exploring this game in particular was due to it's structure. The stochastic nature of how cards are drawn, the way the board state evolves throughout the game, and the difficulty in heuristically scoring a given state (even in this grossly simplified version of the game) made MCTS a prime candidate for tackling this game.

Some interesting design design decisions of note: In order to make this algorithm computationally tractable they needed to address the incredibly large branching factors in the draw step. That problem is $O((n!)^2)$. They use determination to commit chance nodes, like the draw-card step, to a given state and go forward from there. They also establish this metric of "interesting-ness" that measures how much effect a particular ordering of the deck would have on the game. During simulations they look to see if drawing a given card will have the same effect on the game as not drawing the card, and if it has no effect they weight it as "not-interesting". By commiting chance nodes and focusing their search on more "interesting" card orderings, they seemingly build a model of what the deck order is.

Performance was sitting at around 50% which is far from complete, especially considering the large handicap here, but the techniques used in the paper made it an interesting and potentially useful case study.

Monte-Carlo Tree Search in Settlers of Catan (Szita et al. 2010)[4]

In this paper, the researchers apply MCTS to the multi-player, non-deterministic board game Settlers of Catan. They implement an agent that can play with computer as well as human player and demonstrate two approaches they use to provide the agents with limited domain knowledge.

While using the game as the base, they change the rules to remove the imperfect (partially observed) information. To conform with the original rules will be their future goal.

They implemented the Settlers of Catan game in a Java software module named SmartSettlers. It is used as a standalone module for fast gameplay, move generation, and for evaluation.

They conduct the experiment to see whether seat order is an affecting factor for winning the game. With two preliminary experiment, one using random move, the other choosing moves with 1,000 MCTS simulation, it shows that the seat order does affect the winning chance of an agent. This paper does not go deeper into this factor but they propose that more experiment can be done by simulating more games per move.

In one of their experiments, they use their domain knowledge (game insight) to balance between exploration and exploitation. Yet it lowers the performance significantly. They analyze the drop and propose that it might be the domain-specific heuristics are not eager to settle on certain kind of moves that it is hard for the simulation to explore other strategies. It will remain a part for future experiment for their later work.

Even though they use classic MCTS to simulate games, the performance shows a big improvement against the JSettler baseline.

They test their performance against JSettlers and human.

With random movement, the total winning percentage is 0%, with 1000 simulated games, their performance jump up by 27%, which is almost as good as JSettlers. However, with 10000 simulations, MCTS outperforms JSettlers by 30%

When they test SmartSettler against human, they find that expert player can still easily beat the AI and they propose that it might because the tree is not deep enough to find a better strategy or even the strategy human would usually use.

This paper has experimented a promising road for applying MCTS on these non-deterministic, multi-agent games. It proposes many future work to be done, such as looking into the seat-order presumption, simulating with more games for each movement, and deepening the tree search by utilizing selection heuristics.

Generating believable stories in large domains (Kartal et al. 2013)[5]

If viewed from the perspective of the characters, the story-world would resemble a multi-agent game except with numerous possible actions for each characters. To generate a believable narrative is essential for an immersive experience in virtual environment for entertainment.

However, automated narrative generation suffers from the enormous branching factors for each agents, the exponential combinatorial interactions among the characters, and from the scalability across different domains of stories.

In area of automated narrative generation, plot coherence and character believability are two important dimensions to evaluate the generation system. There are mainly two different approaches towards the generation: story-centric system and character-centric system.

Story-centric, approaching from the thought process of an author, would achieve high plot coherence, while modeling from the goals, beliefs, and plans of the characters, i.e. character-centric system, would have a more believable characters.

This paper is most related to character-centric system. In their work, they propose a new algorithm based on MCTS, new heuristics, as well as a story evaluation metric. Their method, as claimed, performs well both on the search time and memory usage. Besides, their approach is flexible enough to be adapted to different domains without pre-defined characters.

In the previous character-centric systems, researchers focus on creating high quality stories in controlled domains by using plot-agent and filtering. This paper shift their attention from controlled domain to over large story domains with many actors, actions, items, and places. Their goal is to plan the story in a large story-world as well as increasing the believability of the story.

A big story-world means a huge search space for planning. With a huge search space, MCTS is a powerful tool by using random sampling. As an anytime algorithm, it would converge to optimal solution if given enough time and memory. Inspired by its performance on game GO, this paper utilizes MCTS to generate a story.

They base their work on user-specific story domain and support a custom domain based on a simplified Planning Domain Definition Language(PDDL)-type of environment specification. They demonstrate their work through a crime-story inspired domain.

Their story-world has three entities:

- Actors
- Places
- Items

$Actors$ can pick up or use $Items$ and go to different $Places$. $Items$ have different actions available for $Actors$. $Actors$ and $Items$ can be in different $Places$.

Each entity has different attributes to it:

- Move(A, P) A moves to place P.
- Arrest(A, B) B's place is set to jail.
- Steal(A, B, I) A takes item I from B. This increase B's anger.
- Play Basketball(A, B) A and B play basketball. This decreases A's and B's anger.
- Kill(A, B) B's health to zero (dead).
- FindClues(A) A searches for clues at its current location
- ShareClues(A, B) A shares with B any clues he has found.
- Earthquake(P) An earthquake strikes at place P. This causes people at P to die (heath = 0), items to be stuck, and place P to collapse.

Besides the set up above, they assume the user specifies both an initial configuration(e.g. X is in Place Y. X has a gun. Z is in city A in his own house) and goal state(e.g. the murderer kills at least two actors and is arrested) for the story.

The believability of each action is a user-defined measure on a scale from 0 to 1, which they treat as a (Bayesian) probability, namely, how likely is the action based on the current world-state. For example:

- Arrest(A, B) More believable if A is a cop. More believable if A has clues to a crime.
- Kill(A, B) More believable if A is angry. More believable if A has previously killed someone.

To apply MCTS, they first define the evaluation function, which evaluate to what extent the belivability has reached the user's goal. They use the percentage of goals a story achieves times it's believability.

To avoid the evaluation function to be stuck at 0 before find a path that satisfies the goal, they add a random rollout, which is a series of random actions that is added to the partial story until all the goals are met.

They also utilize Upper Confidence Bounds (UCB) to balance the exploration and exploitation in the tree search.

When it comes to the memory usage, they has a prune section to address the issue of running out of memory. They propose an iterative approach to produce one action at the time and then fixed the number of nodes on the tree and then iteratively deepening the search, where the tree size is pruned by fixing a number of nodes every time and thus the memory usage is bounded. However, approaching it in this way, it is no longer probability complete (it almost sounds like greedy MCTS).

They also apply search heuristics for the MCTS, using the history table to weight the selection of action and the rollout.

The result for their performance shows that with low budget (low exploration), weighting the selection of action outperforms normal MCTS; with high budget(high exploration), weighting the rollout actions outperforms the selection of action heuristic and MCTS.

The iterative implementation also pays off when their story domain is large because non-iterative MCTS uses up the memory in an early running stage.

In conclusion, for large search space, MCTS is also powerful yet it needs some modification such as the iterative implementation in this paper, and the heuristics to help search through the tree.

But in this paper, it does not model the relations between actors which is an important factor in a believable story.

Generate Causal Story Plots by Monte Carlo Tree Search Based on Common Sense Ontology (Soo et al. 2016)[6]

In this paper, they use a Fabula model and base their approach on the existing common sense ontology ConceptNet5. A Fabula model is composed of six elements: Goal, Action, Event, Perception, Internal Element, and Outcome. Each element has some directed casual relations such as enabling or movtivating the other element.

They try to generate a story from the common senses in ConceptNet5, using concepts from the net as initial state and goal.

Their task involves:

1. Extraction. Extract Fabula casual element from the ConceptNet5.
2. Classification. Classify the extracted common sense ontology to Fabula elements, using part-of-speech(POS) tagger.
3. Search. Using MCTS to find a possible sequence of events to generate a story.

The common sense entries are formatted in 4-tuple of (concept1, concept2, relation, weighting score), where the weight indicates the association strength of the relation between concept1 and concept 2 is.

They build a knowledge base for generating the story by Extraction and Classification.

During selection for their MCTS, they use UCB to balance exploration and exploitation. Unlike usual UCB, in their work, they allow a decaying alpha, which would gear the tree search from exploration to exploitation around the end of the MCTS.

During simulation stage for MCTS, their function takes the story length and the accountability of the relations among events to propagate back the nodes.

In general, they use a classic MCTS with tweaks to the heuristic applied to the algorithm.

The performance of their experiments show that the more times they run MCTS, the story plot would have a higher performance, where it stabilizes around 1,500,000 iterations. The story with the highest score has a very plausible sequence of event.

Their work is refreshing as in they utilize the common senese ontology to model normal actions which are more believable to the reader.

However, with the limitation of search space, the scope of the story generated by their model is mostly very small. Even if a plot would contain 12 small event, the whole scope of the story is still too small to generate a full story.

Improving SAT Solving Using Monte Carlo Tree Search-Based Clause Learning (Keszocze et al. 2019)[7]

This chapter from a workshop discusses how MCTS has been applied to Boolean Satisfiability (SAT) solvers.

In order to apply MCTS to SAT solving, one needs a little more than a vanilla MCTS, there is this idea in SAT solving of Conflict-Driven Clause Learning (CDCL). CDCL learns new clauses from the information available whenever a conflict occurs during search. The idea here being that whenever there is a conflict, rather than searching somewhere else, the cause for that conflict can be extracted as a new clause. Having more clauses to work with helps direct simulations towards better paths.

This work was mostly a case study to see exactly how it would perform (measure in execution time), since even with a CDCL module included, MCTS under-performs when compared to state-of-the-art SAT solvers.

What they have found is that the MCTS based SAT solver was efficient at finding "Good" clauses. These "Good" clauses are clauses that provide useful information about the full SAT problem. By adjusting the heuristic of the search to focus on finding "good" clauses rather than just "all" clauses, the MCTS+CDCL solver can become a pre-processing step that improves the performance of a traditional state-of-the-art solver.

By using MCTS+CDCL solver as a pre-processor to Sat4j, there was a +25% speedup across 16 benchmarks.

Extracting Knowledge from Web Text with Monte Carlo Tree Search (Liu, Guiliang et al. 20XX)[8]

In this paper, this group of researchers apply Monte Carlo Tree Search on a completely different domain–information extraction.

To extract information from web requires a domain-independent extractor that scales to the entire web corpus. It is known as Open Information Extraction (OIE).

Traditional OIE uses rule-based extractors but it is hard to generalize across different domain. Recent works has used an end-to-end model to generate a sequence of facts from a source text. However, the Seq2seq model is confined in certain combination of candidate words and the traditional seq2seq model uses greedy or beam search which makes it possible to miss some better combination.

In this paper, they incorporate MCTS with Predictor Upper Confidence Bound (PUCB) as search heursitcs to address the limitation of the seq2seq model.

To be one of the first to apply MCTS on information extraction, they also have to define an MDP for knowledge extraction problems; the reward signal is not available off-the-shell by the predictor; MCTS takes time to return a satisfiable result considering the task they are doing.

They define the state of MDP as a state containg both source sentence and up-to-now prediction. The action is selected according to the estimated probability distribution for all candidate words and symbols. The reward signal is framed as the similarity score (Gestalt Pattern Matching) between the ground-truth sentence and the predicted sentence.

They build a Reinforcement Learning framework incorporating MCTS both in training procedure and inference procedure. The framework is composed of the reward simulator, a seq2seq predictor, and MCTS.

Their use of MCTS is combined with PUCB, which acts like a search heuristics for action optimality. Other than that, they also find a way to parallelize MCTS with the neural networks to run the task.

Their approach outperforms the pattern matching rule-based approach by 30%, and +10% better than the end-to-end model.

This work has explored the MCTS in a novel area other than game and story generation. It defines MDP for knowledge extraction on unstructured text. It also improves the prediction by a large margin. However, it is still on their agenda to scale the extractor to work with the web corpus.

SECUR-AMA: Active Malware Analysis Based on Monte Carlo Tree Search for Android Systems (Sartea et al. 2020)[9]

This paper proposes a framework for probing the behaviour of potentially harmful applications by executing targeted actions on the operating system. This model frames the problem of malware detection as a searching task, and the search method utilized is MCTS.

The model is put up against 24 classes of malware (1200 examples) in an adversarial stochastic game. The agent tries to learn what actions provide the most information about the malware. This agent was run on a simulated android device, so the possible actions are things like "Making a call" or "Turning on the GPS". The agent trains to learn the policy of the malware (i.e. what it does in response to system actions) in as few actions as possible. Since there are so many possible actions for an operating system, MCTS fits perfectly.

The performance on the 24 class benchmarks shows that, on average, this model outperforms the others on both precision and recall.

Bandit Based Monte-Carlo Planning (Kocsis, Levente and Szepesvári, Csaba 2013)[10]

For large state-space Markov Decision Problem, MC planning is one of the possible way to find the near optimal solutions.

In this paper, they propose a new algorithm UCB applied to trees (UCT), to guide Monte Carlo Planning.

While MC planning sounds promising in theory, it's hard to store a huge tree in practice, the researchers are interested in improving the vanilla Monte Carlo planning with regard to 1) small error probability if the algorithm is stopped prematurely, 2) convergence to the best action if it is given enough time.

The main idea of the algorithm proposed in this paper is to sample actions selectively, while normal MCTS usually use uniform sampling strategy or some domain-knowledge heuristics. The new algorithm apply a particular bandit algorithm Upper Confidence Bounds1 (UCB1), for rollout-based MC planning.

In UCT, the UCB score is computed for each child node, and the child node with the best score will be expanded. It will do this iteratively.

It runs experiments on P-game (random tree games) with uniform sampling MCTS. Even though the average error of uniform MCTS converges at 0.1, which is not bad, MCTS with UCT lower the error by 1e-4 times.

UCT also allows MCTS to converge faster and using less memories, as shown in their experiments.

In short, UCT is a novel application of UCB1 on tree search and it plays a crucial step stone for later works to balance between exploration and exploitation. The proposal of UCT lays the foundation for the later variations of UCT.

4 Future Directions

We have seen that Monte Carlo Tree Search can be applied to many different domains. Any problem with a high branching factor that is intractable for minimax could benefit from MCTS.

But even with MCTS, the memory usage would still reach its limit when facing domains with large search space and multiple-agents interaction, such as in story generation.

We have applied MCTS mostly on games but for the first time that we know that MCTS can also work in information extraction. In the future, we might see MCTS more on different domains, such as language modeling. It is an easy-to-apply algorithm, but the work would be needed on how we model our tasks into an MDP to utilize MCTS.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez,L. Sifre, G. van den Driessche, J. Schrittwieser,I. Antonoglou, V. Panneershelvam, M. Lanctot,S.Dieleman,D.Grewe,J.Nham,N. Kalchbrenner, I. Sutskever, T. Lillicrap,M. Leach, K. Kavukcuoglu, T. Graepel, andD. Hassabis, "Mastering the game of go with deep neural networks and tree search," Nature, vol. 529, no. 7587, p. 484–489, 2016. [Online]. Available: https://doi.org/10.1038/nature16961

[2] T. Pepels, M. H. M. Winands, and M. Lanctot,"Real-time monte carlo tree search in ms pac-man," IEEE Transactions on Computational Intelligence and AI in Games, vol. 6,no. 3, pp. 245–257, 2014. [Online]. Available: https://doi.org/10.1109/TCIAIG.2013.2291577

[3] P. I. Cowling, C. D. Ward, and E. J. Powley,"Ensemble determinization in monte carlotree search for the imperfect information card game magic: The gathering," IEEE Transactions on Computational Intelligence and AI in Games,vol.4,no.4,pp. 241–257, 2012. [Online]. Available: https://doi.org/10.1109/TCIAIG.2012.2204883

[4] I. Szita, G.Chaslot,and P. Spronck,"Monte-carlo tree search in settlers of catan," in Advances in Computer Games,H. J. van den Herik and P. Spronck,Eds. Berlin, Heidelberg: Springer Berlin Heidelberg,2010,pp.21–32. [Online]. Available: https://doi.org/10.1007/978-3-642-12993-3_3

[5] B.Kartal,J.Koenig,and S.Guy,"Generating believable stories in large domains," 01 2013. [Online]. Available: https://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/view/7434

[6] V. Soo, T. Chen, and C. Lee, "Generate causal story plots by monte carlo tree search based on common sense ontology,"in 2016 Joint 8th International Conference on Soft Computing and Intelligent Systems(SCIS) and 17th International Symposium on Advanced Intelligent Systems(ISIS),2016, pp. 610–615. [Online]. Available: https://doi.org/10.1109/SCIS-ISIS.2016.0133

[7] O. Keszocze, K. Schmitz, J. Schloeter,and R. Drechsler,Improving SAT SolvingUsing Monte Carlo Tree Search-Based Clause Learning.Cham: Springer International Publishing, 2020, pp. 107–133. [Online]. Available: https://doi.org/10.1007/978-3-030-20323-8_5

[8] G.Liu,X.Li,J.Wang,M.Sun,and P. Li, "Extracting knowledge fromweb text with monte carlo tree search,"in Proceedings of The Web Conference 2020, ser. WWW '20.New York, NY,USA: Association for Computing Machinery,2020, p. 2585–2591. [Online]. Available: https://doi.org/10.1145/3366423.3380010

[9] R. Sartea, A. Farinelli, and M. Murari,"SECUR-AMA: Active malware analysis based on monte carlo tree search for android systems," Engineering Applications of Artificial Intelligence, vol.87, p. 103303, Jan.2020. [Online]. Available: https://doi.org/10.1016/j.engappai.2019.103303

[10] L.Kocsisand C.Szepesv ari,"Bandit based monte-carlo planning," in Machine Learning:ECML2006,J.F urnkranz,T. Scheffer, and M. Spiliopoulou, Eds. Berlin,Heidelberg :Springer Berlin Heidelberg,2006, pp. 282–293. [Online]. Available: https://doi.org/10.1007/11871842_29