Cognitive project using Monte Carlo Tree Search

Guy Barash
ID: 301894234
Bar Ilan University
guysbarash@gmail.com

June 2020

1 Abstract

In this project we were required to create a general purpose agent, which can be introduce to an entirely new problem and domain, and reach its goal as define in the problem. without any prior assumptions on the world. In this excercise we'll attempt to solve the generic case for a deterministic and for a probabilistic single agent games.

2 Introduction

The problem of general game solving algorithm is not a solved problem. But major development were made in the field in last few years ([7, 9]). In this exercise we'll attempt to solve multiple types of worlds using an agent. Each world the agent will encounter will have actions (perhaps deterministic perhaps not), goals to meet, state description - but no other agent to compete or collaborate. There can be a single goal, or multiple goals, but all goals are known at any given state. The game can have complete knowledge, or semi-complete, meaning that some information is revealed only when a certain condition is met.

2.1 Interaction with the world

In this exercise we were given a very generic task of constructing an agent, perhaps based on Reinforcement learning, which will complete the following loop:

1. Initialize.
2. If goal is completed - conclude run with victory.
3. If steps cap was reached, or there are no legal moves - conclude run with failure.
4. If none of the above are not satisfied:

(a) Fetch current state

(b) perform evaluation

(c) recommend optimal action

(d) apply optimal action and return to part 2.

2.2 Description of the world

The agent is expected to receive 2 files which describe the world in PDDL [10] format. The first file is the *domain*, the seconds file is the *problem* file.

The domain is a file which contains a list of the types of predicates and the available actions in this world. The actions can have preconditions to determine what must exist before this action can take place. The changes in the world it performs, and in the probabilistic world - in what probability we'll receive each outcome.

The problem is a file which contains the object is the specific problem, their types and relations. The problem also contains the initial state of the world and the goals the agent needs to fulfil. In case there is hidden information, it is also written in this file, along with the conditions in which the hidden information is revealed.

3 Algorithm description

The algorithm i've chosen for this work is Monte Carlo Tree Search (MCTS) [3, 4] which is an algorithm that became more and more dominant of the last few years, peaking in popularity after used in AlghaGo [11] to defeate the world champion in Go, Lee Sedol [2].

The algorithm presented in this work is a modification to the MCTS algorithm, with several optimization to match the problems in this paper.

The algorithm has a few steps, elaborated in the following section.

3.1 Initialization stage

In this stage we set the global parameters, read the input, import the existing policy (if any) and determine if we're in EXECUTE mode, or TRAIN mode, the *mode* effects several constants, including but not limited to: W the *width* of the search. How many simulations will the algorithm run before making a choice for the next action.

D the *Depth* of the search, in case no termination state is reached, how many steps will each simulation take before terminating.

A higher D is good for fewer valid actions, but more complex scenarios (such as freecell), and a higher W is good when there are a large number of options at each state, but not necessarily complicated goals (such as the satellite problem).

Of course, setting both of them high will usually yield good results, but will take too long to run.

3.2 State evaluation stage

In this stage we fetch the current state. We evaluate if this state is terminal - if so, end the game. If not, we **HASH** the current state. The hashing state is done by converting each state to an alphabetical ordered list of predicate, turning it into a string and hashing that string using HashLib [6]. The Hash is used as the Unique key. For each Key, a node is generated. Therefore each state has a single node and key associated with it. This part is important to avoid keeping in memory a huge bank of repeating states. This technique is based on **dynamic programming** [8].

Each Node, keeps information regarding that node, including: The state itself, the Hash key, the number of times this state was visited by a simulated agent, the total rewards of that agent, is it a terminal state, and the actions available from that stage.

Each Action in each state contains the string of that action, the number of times it was simulated and the accumulative rewards it gathered.

When a state if fetched, it is processes to node is explained, as well as documented that the current state was visited by the **real** agent (in contrast to the simulated agents).

3.3 Simulation stage

In this section we perform the simulations in which knowledge of the world is gathered. In this section we generate a simulated agent and run it as described below, however we repeat this process W times, as defined in Section 3.1.

3.3.1 Action selection

For each simulated agent $w_i, i \in \{1, ..., W\}$, a simulated agent is created. This agent is in the same state as the real agent. The agent is given all the possible action, with their stats (visits and accumulated rewards). It uses Multi Armed Bandit [1] formula to determine the optimal action to explore:

$$\underset{a \in Actions}{\operatorname{argmax}} \frac{\#\ Rewards\ for\ a}{\#\ visits\ in\ a} + \gamma \sqrt{\frac{\#visits\ in\ a}{\sum_{a \in Actions} visits\ in\ a}} \tag{1}$$

where γ is the exploration factor and *Actions* are the available actions for that state. As the exploration factor decrease the chosen action will be based on the expected results of that action, and as the exploration rate increase the chosen action will be chosen be the least visited action. The exact value of the exploration rate was determined empirically.

3.3.2 Play-out

After the initial action is chosen, the agent execute that action on a simulated environment. After this action, the following actions are completely random from the valid actions. Each time a new state is encountered, it is processed and add to the dynamic programming library as explained in 3.2. The simulation is completed when a certain number of steps is taken, a goal state is reached, there are no available action or when a loop is encountered. The path of the simulated agent is kept as well as the termination reason.

3.3.3 Back propagation

After a path is recorded, the algorithm start the back propagation. For a path of length N, we fetch the node that matches the $N-1_{th}$ state, we increase the number of visits of the last action by 1, and the accumulated rewards by the value of the current reward. We add a step penalty to the reward, and we recursively move to the $N-2$ state of the simulated path. The algorithm now returns to section 3.3.1, until all agent simulations are completed.

3.4 Optimal action selection stage

In this stage the algorithm needs to give it's recommendation for the optimal action for the real action. The stage is almost identical to the action selection in the simulation (Section 3.3.1) except that for this the exploration factor is exactly 0.

$$\operatorname*{argmax}_{a \in Actions} \frac{Rewards\ for\ a}{visits\ in\ a} + 0 \tag{2}$$

In this case the action is chosen only be the expected reward.

3.5 Exporting policy

When a learning section is concluded, the agent can export the dictionary with all the states to a file, and so next time it is re-activated, it has the knowledge from prior iterations.

4 Rewards and their reasoning

Each simulation is concluded due to one of following, each reason with a fitting reward, as detailed in Table 1.

4.1 Comparison to other methods

Q learning, one of the other leading technique is a robust and versatile method, but it tends to be more fitting for smaller worlds with less available options. Monte Carlo Tree search is designed for domains with a state-space to large to explore, as some of the domains in this exercise.

Description	Reward	Terminal?
Goal reached	+60.0	True
New predicate solved	+10.0	False
Dead end	-2.0	True
Loop encountered in simulation	-4.0	True
Step penalty	-0.01	False
Visiting a state visited by the real agent	-2.0	True

Table 1: Rewards and penalties

5 Experiments and analysis

The algorithm, in it's current form has manage to solve **all** of the domains available for this exercise, in reasonable time and space complexity. There for in this section we will analyse how the algorithm react to different scenarios.

5.1 Rewards over time

The algorithm is built to focus on areas "more likely to win", there for we expect that the reward for each action will increase as the agent come close to it's goal.

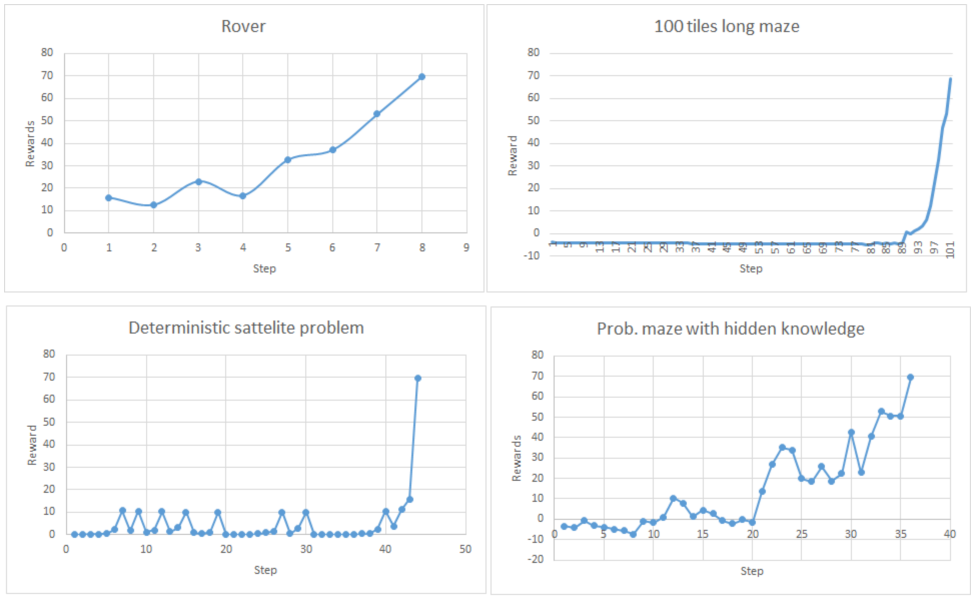


Figure 1: Rewards for each step, for 4 different worlds

In Figure 1 we can see the rewards over time of 4 entirely different domains. In the top left, the **Rover** domain is perhaps the simplest problem, the agents "run around", meaning most of the agents do not end with any significant success, but a minority of agent do reach at least one goal. As the real agent closes on the target the fraction of the agents with a good termination result increases and the reward is increased with each step. We get an interesting visualization on how the algorithm is "going for the kill", when the goal is in sight the decision becomes easier and easier, represented by the increase in reward for each step.

In the **100 tiles long maze** we can see that the agents usually do not see the end goal, this is why the rewards are usually below 0, but the penalty is larger for backtracking, that is why "forwards" action has slightly reduced penalty and the agent keeps in this direction, until the goal is in sight and it "runs" toward it.

In the **prob. maze with hidden knowledge** we can see a similar behaviour to the Rover, except now because of the hidden knowledge and probabilistic moves not all agents acts the same and we see a more "noisy" behaviour. If the algorithm was given an infinite amount of time it will also converge to a smooth line, but it was limited on purpose.

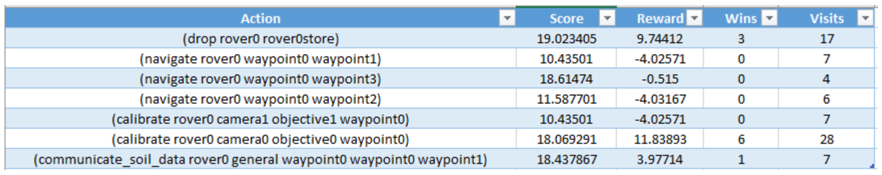
In the **Satellite** problem, we have multiple, un-related sub-goals, and we can see that sometimes the agents randomly looks for clues, without success (the flat lines) until it find a meaningful clue (an agent that solves a single predicate) and it pursues that lead. the agent repeat this "search-and-run" policy until it find the goal and runs after it.

In all 4 example scenarios we see the same gradual exploration of the world until a meaningful path is discovered and then it runs toward it. The philosophy of the algorithm, in intuitive words is "If you find a target - seek it. If not - look for one in a new area".

5.2 Explore vs exploit

In Figure 2 we can see a snapshot of the state right before making an action. It has 7 valid actions. One can see that the more positive (wins) simulation this action got - the more visits it gets.

If we were to choose which action to **explore** we would go for the one with the highest "score". But, if we want the action which is most likely to win - we would choose the action with the highest score. In this example, the optimal action to explore is "Drop rover0", which means that the algorithm is suspicious that this action is good, but it is not sure. On the other hand, if we are to choose the optimal move for the win - the "Calibrate rover0 camera0" is the right action, it has the highest reward, which is also reflected in the highest score on the "Win" column.



Action	Score	Reward	Wins	Visits
(drop rover0 rover0store)	19.023405	9.74412	3	17
(navigate rover0 waypoint0 waypoint1)	10.43501	-4.02571	0	7
(navigate rover0 waypoint0 waypoint3)	18.61474	-0.515	0	4
(navigate rover0 waypoint0 waypoint2)	11.587701	-4.03167	0	6
(calibrate rover0 camera1 objective1 waypoint0)	10.43501	-4.02571	0	7
(calibrate rover0 camera0 objective0 waypoint0)	18.069291	11.83893	6	28
(communicate_soil_data rover0 general waypoint0 waypoint0 waypoint1)	18.437867	3.97714	1	7

Figure 2: possible actions and expected reward. ”Reward” is the expected reward with exploration = 0 , ”Score” is the reward with a constant exploration constant. ”Wins” is how many simulated agents had reached a goal, and ”visits” is how many agents we simulated for that action.

6 limitations

The original algorithm was not designed to encounter loops and non deterministic world, while the modifications presented to the algorithm in this paper seems to make it robust to these challenges it is not thoroughly explored.

7 Conclusion

In this paper we have shown a version of the Monte Carlo algorithm which has successfully solved **all** of the domains and problems that are available. It is not certain that the algorithm will manage to solve other domains, but we believe it manages to capture the notions of the state-of-the-art along with several unique key sight to make the algorithm a good fit to the challenges presented in this project.

8 Another aspect of sending exploring agents for robust learning

I would be very surprised if someone actually read the entire paper. But if so, thank you and enjoy this quote from my favorite book, ”The little prince”.
Your planet is very beautiful,” he said. ”Has it any oceans?”
”I couldn’t tell you,” said the geographer.
”Ah!” The little prince was disappointed. ”Has it any mountains?”
”I couldn’t tell you,” said the geographer.
”And towns, and rivers, and deserts?”
”I couldn’t tell you that, either.”
”But you are a geographer!”
”Exactly,” the geographer said. ”But I am not an explorer. I haven’t a single explorer on my planet. It is not the geographer who goes out to count the towns, the rivers, the mountains, the seas, the oceans, and the deserts. The geographer is much too important to go loafing about. He does not leave his desk. But he receives the explorers in his study. He asks them questions, and

Figure 3: An example of an agent too busy in exploring, but never exploiting.

he notes down what they recall of their travels. And if the recollections of any one among them seem interesting to him, the geographer orders an inquiry into that explorer's moral character."

[5]. See Fig 3.

References

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[2] Steven Borowiec. Alphago seals 4-1 victory over go grandmaster lee sedol. *The Guardian*, 15, 2016.

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

[4] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008.

[5] Antoine de Saint-Exupéry. *The little prince: A new translation by Michael Morpurgo*. Random House, 2018.

[6] https://docs.python.org/2/library/hashlib.htmlmodule hashlib. *HashLib*, 2009 (accessed February 3, 2014).

[7] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[8] Samuel Karlin. The structure of dynamic programing models. *Naval Research Logistics Quarterly*, 2(4):285–294, 1955.

[9] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Communications Surveys & Tutorials*, 21(4):3133–3174, 2019.

[10] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language, 1998.

[11] Fei-Yue Wang, Jun Jason Zhang, Xinhu Zheng, Xiao Wang, Yong Yuan, Xiaoxiao Dai, Jie Zhang, and Liuqing Yang. Where does alphago go: From church-turing thesis to alphago thesis and beyond. *IEEE/CAA Journal of Automatica Sinica*, 3(2):113–120, 2016.