Active Reinforcement Learning - Research Project Proposal

Instructor: Dr. Ofra Amir.
Student: Dvir Lafer, 206330342.

1. **Prequities**:
   - Reinforcement Learning - A machine learning technique in which an agent is designed to generate a policy that should maximize a reward/reward function by trial and error and without outside guidance.
   - Active Reinforcement Learning - In some use cases, the reward is unknown unless asked for specifically, so a more realistic approach would be to extend the set of actions of an agent to be able to query for the current reward.
   - Multi-Armed Bandit: An environment in which the agent needs to choose which "arm" to pull (which action to choose) out of a given set of arms, and each hand has its distribution for rewarding the agent.

2. **The Problem**:
   The vast majority of RL algorithms are designed to find a (sub) optimal policy given the ability to observe the reward. Moreover, the theoretical properties of these algorithms are based on this crucial ability.
   Several algorithms were modified or created to solve this problem, and we would like to be able to tell which algorithm is the best for our settings.

3. **The Settings**:
   This research started from a question "Intuition Robotics" asked Dr. Ofra to explore and this is where our settings derived from. They determined we should handle this problem as a multi-armed bandit problem and in addition to choosing the arm, our agent should also choose whether to query for the reward or not.
   For simplicity, we looked at a two-armed bandit with Bernoulli distribution over its arms, p=0.2 for the "left" arm and p=0.8 for the "right" arm.
   The algorithms were tested against two cost functions which will be explained in the next section.

4. **The Research**:
   - Two algorithms: Knowledge Gradient (KD) and Mind-Changing-Cost-Heuristic, were explored by Dr. Ofra's MSc. student in the mentioned settings.
   - Our research focused on testing another algorithm called Bayes-Adaptive Reinforcement Learning using Sample-Based Search (BAMCPP) and evaluating it against Knowledge Gradient.
     - This method utilizes two key elements: Bayesian learning and Monte Carlo tree search. Using Monte Carlo tree search our agent chooses which arm to pull and whether it should query or not. Based on this accumulation of rewards during simulations (if they were queried) the agent should update its belief about the arms' distributions for reward.
   - Both KD and BAMCPP were tested in a fixed cost environment (cost = 0.5), and in a changing cost environment following the policy:

$$cost = 1[query] \cdot cost \cdot increaseFactor + (1 - 1[query]) \cdot max(cost, decreaseFactor \cdot cost, baseCost)$$

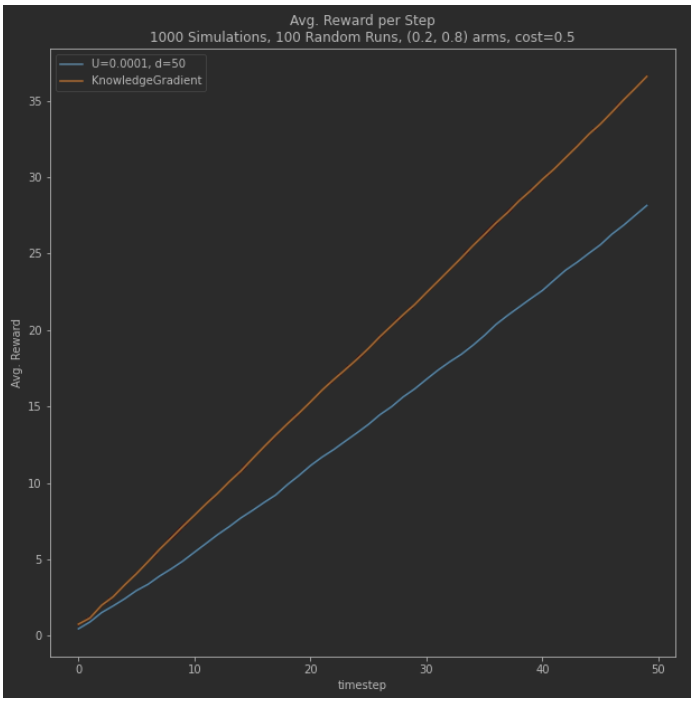
Where the base cost was 1, the increaseFactor was 2 and the decreaseFactor was 0.5.

- The research had two stages:
  - Applying the mentioned algorithm to our settings and comparing the results to the results of Knowledge Gradient.
  - Trying to improve the proposed algorithm so it would win against the current best-performing algorithms.
    - The original second stage was: Trying to design a realistic cost function (instead of a fixed one) that mimics better human behavior.
      But since the algorithm did not perform as expected we took a turn and tried to improve it.

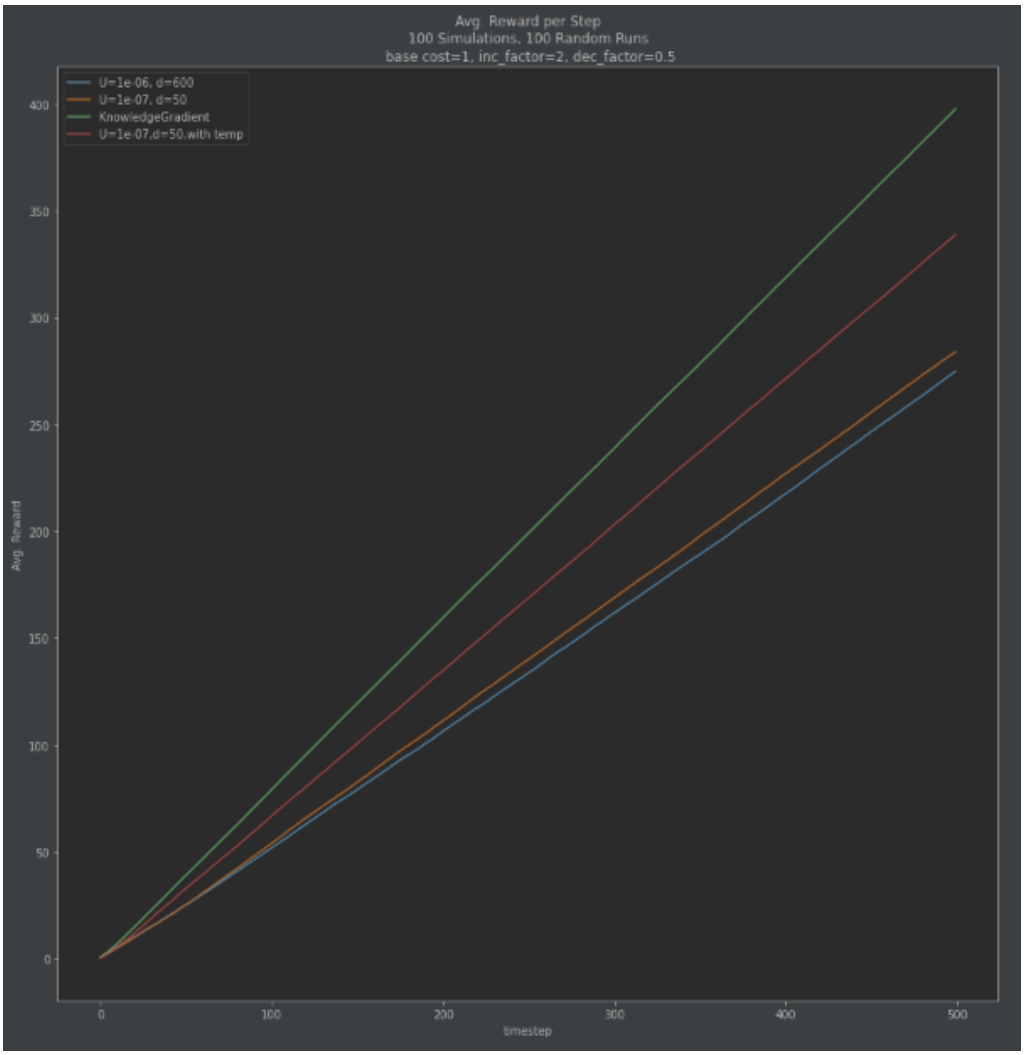
5. **Results:**
   - We evaluated the two algorithms (KD & BAMCPP) using several metrics (all of them were averaged over 100 runs/episodes):
     - Reward - The average accumulated reward each algorithm gained.
     - Regret - The average accumulated reward minus the query cost.
     - Query probability - The average of querying at each timestep over 100 runs.
     - Arms convergence - The probability of each algorithm converging towards the higher-paid arm.

   - During the tests in changing cost experiments, we introduced the improved BAMCPP:
     - The improvement included a temperature-like hyper-parameter which “forces” the algorithm to query more at the beginning.
     - The method was: allowing for regular MCTS to take place, but when the time comes for choosing the current argmax over the arms and querying (after reaching the maximum number of simulations), we added a bias towards querying by dividing the Q-values of querying with the current timestep and dividing the Q-values of non-querying action with the remaining timesteps until arriving the horizon.
     - The new hyper-parameter that was introduced as a result was: the proportion of timesteps out of the horizon that will be biased in that way.

   - For both the fixed cost and the changing cost, the results were consistent:
     - Higher reward for KD:
       (1)Reward in the fixed cost settings (U is the exploration constant and d is the depth limit in each Monte-Carlo tree search):

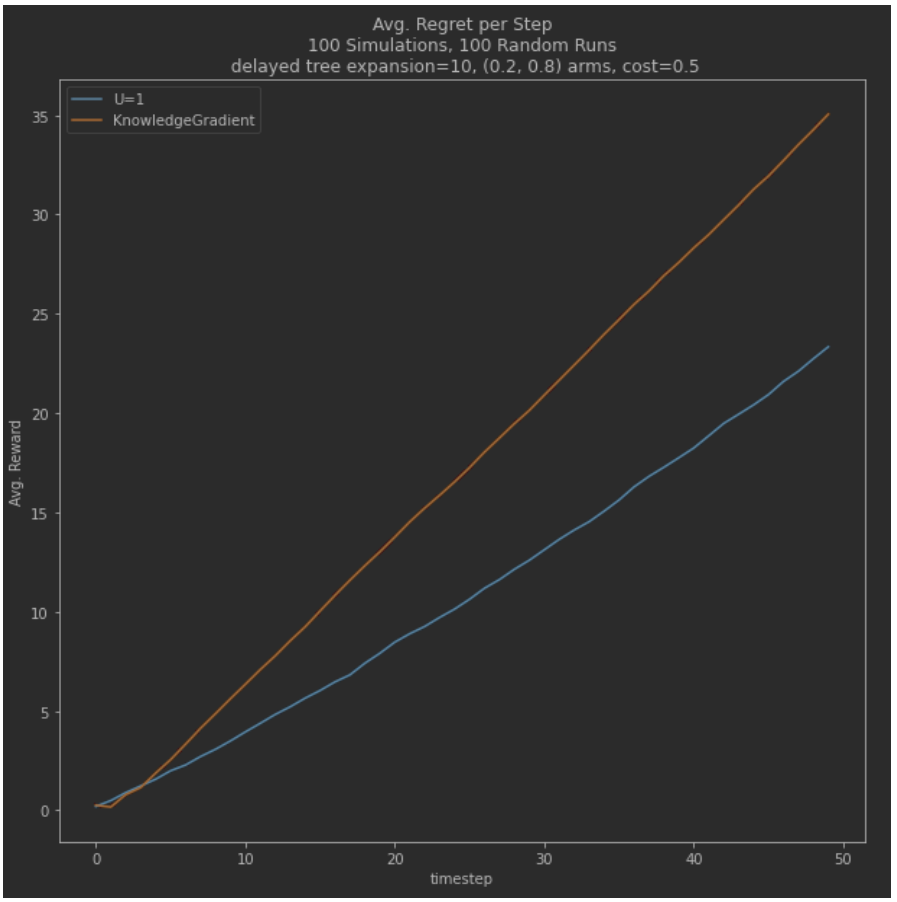


(2)Reward in the changing cost settings:

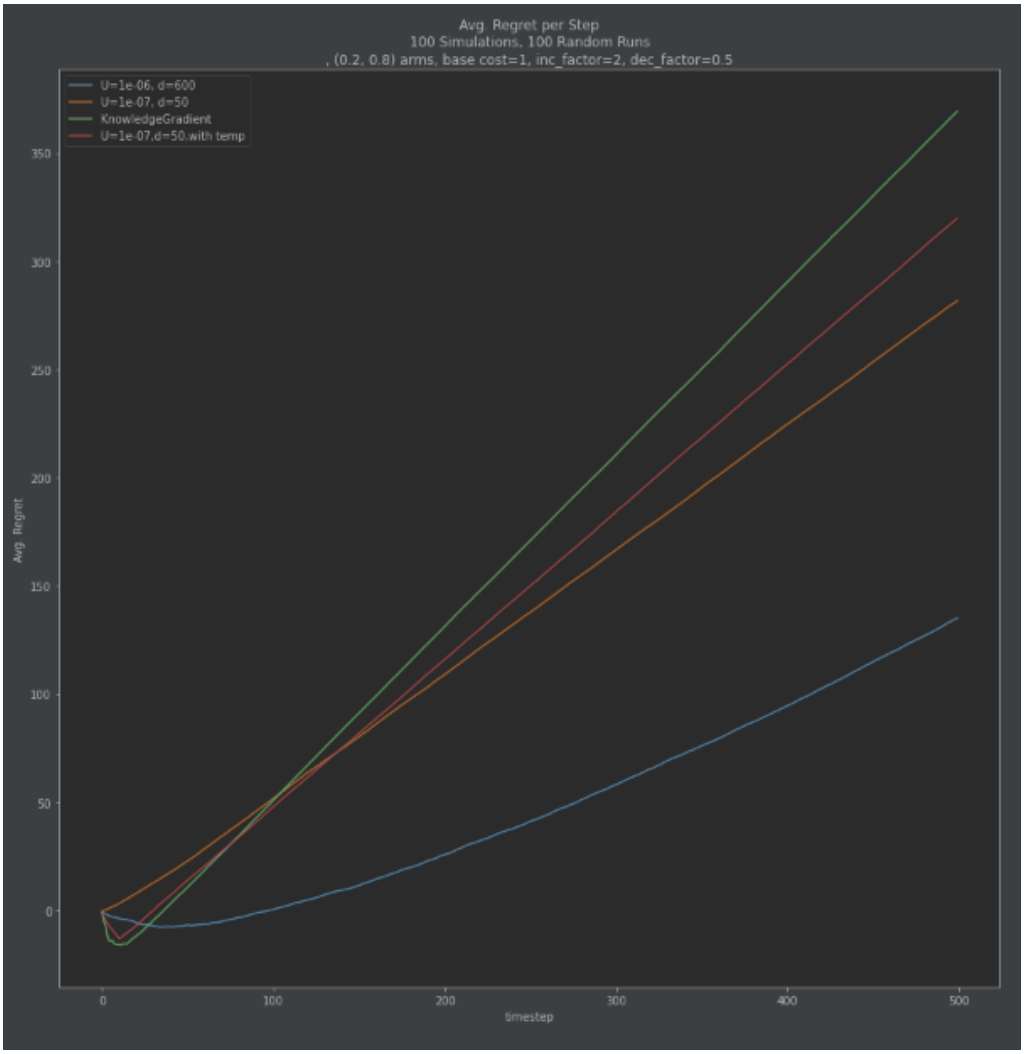


- Lower regret for BAMCPP:

(3)Regret in the fixed cost settings:

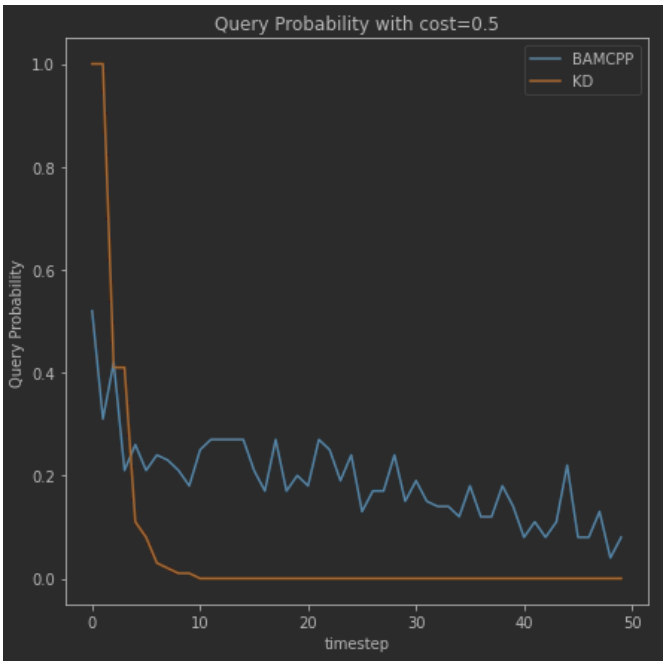


(4)Regret in the changing cost settings:

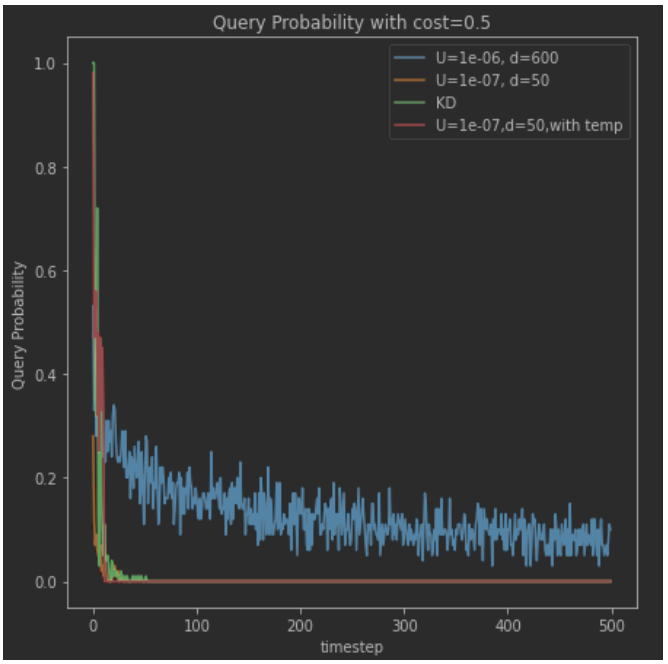


- BAMCPP tended to query more along the way with lower probability, while KD tends the almost always query at the beginning of the run and decrease rapidly to zero probability for querying:

(5)Query probability in fixed cost settings:



(6)Query probability in changing cost settings:



- BAMCPP, in most cases (depending on hyper-parameters), did not have 100% success at converging towards the more rewarded arm while KD did.

6. **<u>Analysis and conclusions:</u>**
   - The original BAMCPP algorithm did not tend to query enough compared to KD.
   - Adding the temperature element indeed pushed it towards querying more.
   - Even after the improvements, BAMCPP with the temperature element wasn't able to achieve greater or equal reward compared to KD's.
   - However, with or without the temperature, BAMCPP did achieve less regret in all settings compared to KD.
   - In terms of convergence to the preferred arm, KD also has the upper hand since it converged in all episodes while BAMCPP, with or without temperature, didn't always converge (depending on hyper parameters and at the expense of other metrics).
   - BAMCPP is super sensitive to hyper parameters:
     - Lower exploration constant = higher querying probability.
     - Bigger depth limit during MCTS = higher querying probability.
     - Higher querying probability leads in most cases to better convergence towards the preferred arm.
7. **<u>Discussion:</u>**

   Our purpose was to compare the performances of two algorithms in the domain of ARL:
   - A known algorithm - KD
   - A relatively new algorithm - BAMCPP

   The results showed that KD still remains superior in most metrics even though BAMCPP incorporates two useful methods - Bayesian learning and MCTS.
   Even after incorporating another strong element - the temperature, KD still performed better.
   A possible direction for trying to improve BAMCPP could be checking the convergence of the Bayesian learner, i.e. if the learner had reached some level of convergence (a metric should be determined) it can stop using MCTS and always choose the converged arm.

8. **Sources:**
    - Reinforcement Learning: A Survey
    - Active Reinforcement Learning
    - Efficient Bayes-Adaptive Reinforcement Learning using Sample-Based Search
    - Active Reinforcement Learning with Monte-Carlo Tree Search