A Simple Reinforcement Learning Framework for Repeated Budget-Aware Ad Auction Bidding

Julian Lee
Harvard College
COMPSCI 136, Fall 2021
slee5@college.harvard.edu

Abstract

The COMPSCI 136 repeated generalized second-price ad auction tournament consists, by necessity, mostly of non-adaptive agents with hard-coded strategies. In this project, we attempt to develop an adaptive, reinforcement-learning based agent using the Q-Learning framework to compete in the tournament, with the goal of winning it. We present a wide range of design choices that we have made in constructing this adaptive agent, notably defining actions as target slots, determining bid price independent of the Q-Table, and discretizing the budget to keep the state at a manageable dimension. We discuss experimental results, which are mixed and show roughly equal performance for the RL-enabled agent amongst the top 10 non-adaptive agents submitted by the class.

1 Introduction

As part of Problem Set 6 of COMPSCI 136, students were invited to develop strategies for a repeated budget-aware advertising auction, and enter agents implementing these strategies into a class-wide tournament. Because of the restrictions on the Python libraries permitted as well as the short timeframe, the vast majority of these agents were 'dumb' and non-adaptive: that is, their strategies were pre-determined by the human programmer based on certain auction conditions such as remaining budget, and did not change throughout the course of the auction regardless of what strategies the opponents appeared to be implementing.

In this project, we attempt to implement a 'smart' bidding agent that adapts its bidding strategy over time, based on what it has found most profitable in the past against the actions of its opponents. In theory, if implemented correctly, the agent will gradually learn the best response strategy given any opposing field and exploit this best response for the remainder of the auction. In practice - at least for my project - the results are somewhat mixed.

This is by no means the first attempt to implement adaptive bidding agents: my project is heavily influenced by Cai et al. (2017), who developed a Markov Decision Process-based framework to allocate advertising budget over the entire timeframe of an auction in a manner that maximizes long-term expected reward. Cai et al. showed a significant improvement in profit when bidding with their model, which uses dynamic programming to update action utilities via the Bellman equation, over non-adaptive strategies used in industry. Notably, though, they utilize a model-based solution, where the state of the auction is measured through some feature vector and the prediction of the transition probabilities from state to state are leveraged when calculating the optimal next action.

Because of the added complexity of modeling transition probabilities between auction states, we adopt a model-free reinforcement learning approach. Specifically, we use Q-Learning, a simple off-policy RL strategy to learn actions. While the specifics of the model will follow in the next section, Cai et al. note that model-free approaches may suffer from 'problems of transition dynamics

Preprint. Under review.

of the enormous state space, the sparsity of the reward signals and the highly stochastic environment.' We note all these potential pitfalls, and describe some ways in which we attempt to mitigate them in the construction of our agent.

2 Experiments

2.1 Auction Setup

In the problem set tournament, the auction involved groups of 5 agents, competing across 200 iterations, or 'days', each with 48 auction periods. The daily budget was set at \$600, and agent-specific values were resampled at the beginning of each day from the uniform distribution on (0.25, 1.75).

We make three slight adjustments to this auction setup.

1. Our best results are reported using 500 iterations instead of 200, which is logically coherent as a higher number of iterations gives the agent more time to learn the value function and the subsequently exploit it.
2. We initialize each agent only once per block of 500 days, and the agent retains the same value throughout those 500 days rather than sampling a new value each day. We believe this to be more realistic, as companies' click-values do not change drastically from day-to-day; moreover, this drastically simplifies the state space, as we do not have to include the agent's inherent per-click value in the state. To ensure statistical significance of our experiments, we reinitialize agents and run the 500-day-block once for every possible permutation of values to agents.
3. Finally, because the second requirement means that the time complexity of running a group auction grows super-exponentially, we limit our groups to three agents instead of five.

2.2 Model Setup

Our model's commented code can be found at this repository. It involves several parts working in conjunction, which we will describe sequentially.

2.2.1 Q-Table

The most important part of our agent is the Q-Table, which stores learned utilities for each action in each state. Crucially, we define a state as a two-dimensional vector comprising a discretized representation of the agent's remaining budget and the current round of the daily auction, while we define an action as targeting a specific slot of the auction.

We chose to 'bucket', or discretize, the remaining budget, because as Cai et al. mentions, failing to do so leads to a continuous state space with an infinite number of possible states, since fractional payments are permitted. While learning a continuous value function is possible, it would require more sophisticated methods like the deep Q-network which are beyond the scope of this project. We trialled a variety of different bucketing numbers, ranging from 10 to 60, and found a value of 15 to be optimal.

It was not immediately apparent what the best way to define an action would be. Learning an exact value for a bid was not feasible for the same reasons as above, but initially we attempted to learn a discrete coefficient, ranging from 0 to 1.0 in increments of 0.1, by which we would 'discount' the optimal bid as calculated by the budget-unaware balanced-bidding formula. The logic behind this was that because balanced-bidding is an equilibrium in a budget-unaware situation, we would simply select certain rounds in which to deviate from it in order to preserve our budget across the entire day. In practice, this was found to be extremely ineffective, perhaps because the implementation made the oversimplifying assumption that opponents would bid the exact same amount every round.

Instead, the Q-Table of our final agent defines an action as targeting a specific rank in the auction. Therefore, for an auction with four agents, there are four possible actions in each state, noting that the fourth slot does not receive any advertising revenue but may still hold strategic importance by driving up the price of the third bidder.

2.2.2 Bid Price Determination

Notably, in our agent, the actual bid price is calculated independently of the Q-Table. We simply keep track of the winning bids for each slot in each time period, and if, in time period t of 48, the Q-Table recommends us to target slot s, we look at the distribution of bids that have occupied slot $s+1$ in the same period t of all previous days, and set $min - bid$ equal to a bid value that beats some proportion p of those past bids. Once again, this p-value, akin to the probability with which we successfully secure the targeted slot, is adjustable, and we find that in practice a value of 0.85 is optimal, as it allows us to secure our targeted slot often enough while not having our bid inflated by a handful of outlier values, as would happen if we set $p = 1.00$. For all slots apart from the first and last, we then construct a balanced bid using $min - bid$, using the more accurate historical number of clicks in round t of previous days rather than round $t-1$ of the same day to calculate position effect. For the first slot, we simply bid $min - bid$, and for the last slot, we bid nothing.

2.2.3 Q-Learning Update Rule

Therefore, at each time period in a day, the agent can encode their state as a tuple of their discretized remaining budget and the time period - for that state, the Q-Table will store the expected utility over all remaining time periods in that day derived from targeting each slot. It's notable that the point of the exercise is for the RL agent to be non-myopic - that is, it must be able to maximize future rewards instead of just short-term rewards, since sacrificing present reward by bidding lower at an earlier round can present significant long-term payoff as the agent will have higher budget remaining at later rounds. The Q-Learning framework accounts for this by updating the Q-Table using a Bellman Equation:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * (r_t + \gamma * max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

We observe that taking the maximum over all possible Q-values in subsequent states enforces this 'farsightedness', as a higher r_t, or reward accumulated in the present state, can come at the cost of a less profitable resulting state. α and γ are Q-Learning specific parameters, with the former corresponding to the learning rate of the algorithm and the second corresponding to the discount rate applied to future rewards. We set $\gamma = 1$ here, as our agents are judged on their cumulative reward at the end of each day, so more time-proximal rewards are not worth any more than time-distant rewards. The α value is less certain but seems to grow in proportion with the number of budget-buckets and thus the number of possible states. This is intuitive because as there are more possible states, each state is explored fewer times and thus each observation needs to be weighted more, while with fewer distinct states, each state is explored many times and thus each observation should have less of an impact on our overall evaluation of the state's utility. In practice, we set $\alpha = 0.05$ for 15 budget buckets, but notice that $\alpha = 0.3$ was more performant when we increased the number of buckets to 48.

2.2.4 Exploration vs. Exploitation

A key tradeoff in reinforcement learning involves balancing exploration, the act finding new actions which might potentially be optimal, with exploitation, the act of executing actions which you currently believe to be optimal. Here, we use a slightly modified version of the decayed-epsilon-greedy method to balance these two ideas. The decayed-epsilon-greedy method specifies that at each time step, with probability $1 - \epsilon$ we exploit by taking the action which we currently believe to be optimal - in other words, the action with maximum value in the Q-Table for the current state. With probability ϵ we explore, meaning that we uniformly take any action from the action space, independent of what we believe to be the best. At each day, we reduce ϵ by multiplying it with a decay factor, so that the auction agent gradually shifts towards exploiting the optimal slot targeting strategy as it builds a more statistically significant picture of the utilities of each action.

In our model, we set the initial ϵ to be 1, meaning that every single action taken on the first day is randomly selected. The ϵ is decayed by a factor of 0.975 each day, but we set a minimum threshold of 0.05 under which it no longer decays. We observe that in general, this is a very high exploration rate, with the threshold of 0.05 being reached only on the 117th day. This is, nevertheless, necessitated by the large size of the state space, as mentioned in Cai et al. - with 48 time periods, 4 actions, and 15 budget buckets, we have a total of 2,880 cells in the Q-Table whose values must be learned.

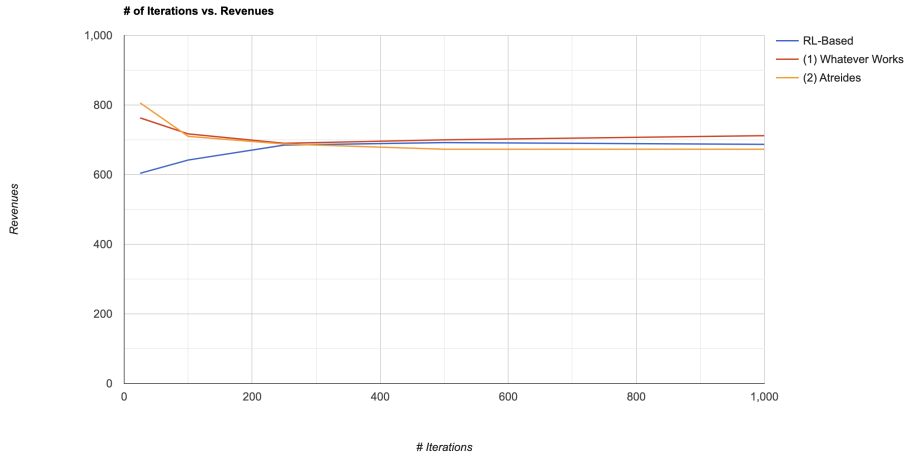
The modification we make to the standard decayed-epsilon-greedy method is the following: we noticed that after many experiments, the agent was expending its entire budget well before the 48th period each day, and often as early as the 30th. A closer inspection of the Q-Table's values revealed zeros in all of the later period, suggesting that the agent was never able to enter those periods with remaining budget with which to bid. Intuitively, we hypothesize that this is because conserving budget to the last period requires targeting a lower-priority slot multiple times consecutively, and therefore has low probability of occurring given the many different permutations by which the agent can expend their budget before the final periods. Therefore we slightly expand the probability assigned to bidding 0 - when the 'explore' option is selected using the epsilon-greedy policy, we bid 0 to target the last slot with 50% probability, and select to target any of the other slots uniformly within the other 50% of cases. Before implementing this modification, the agent would exhaust its budget each day at a mean period of 43.232, with a standard deviation of 2.08; afterwards, this increased to 45.04 with a standard deviation of 1.76.In practice, we found that this significantly improved the agent's performance against other class-submitted agents.

2.3 Competition

The performance of our RL-enabled agent was measured against three subsets of the highest-performing agents in the class tournament. The first group consisted of the top two agents in the class tournament: whatever_works_works_budget, Atreidesbudget. The second group consisted of the third and fourth ranked agents: MBHDbudget, and Jgwswbudget. The third group comprised two slightly less successful agents, who were eliminated at the semifinal stage but still ranked in the top 10 of class-submitted agents: Rllxbudget and ccbudget.

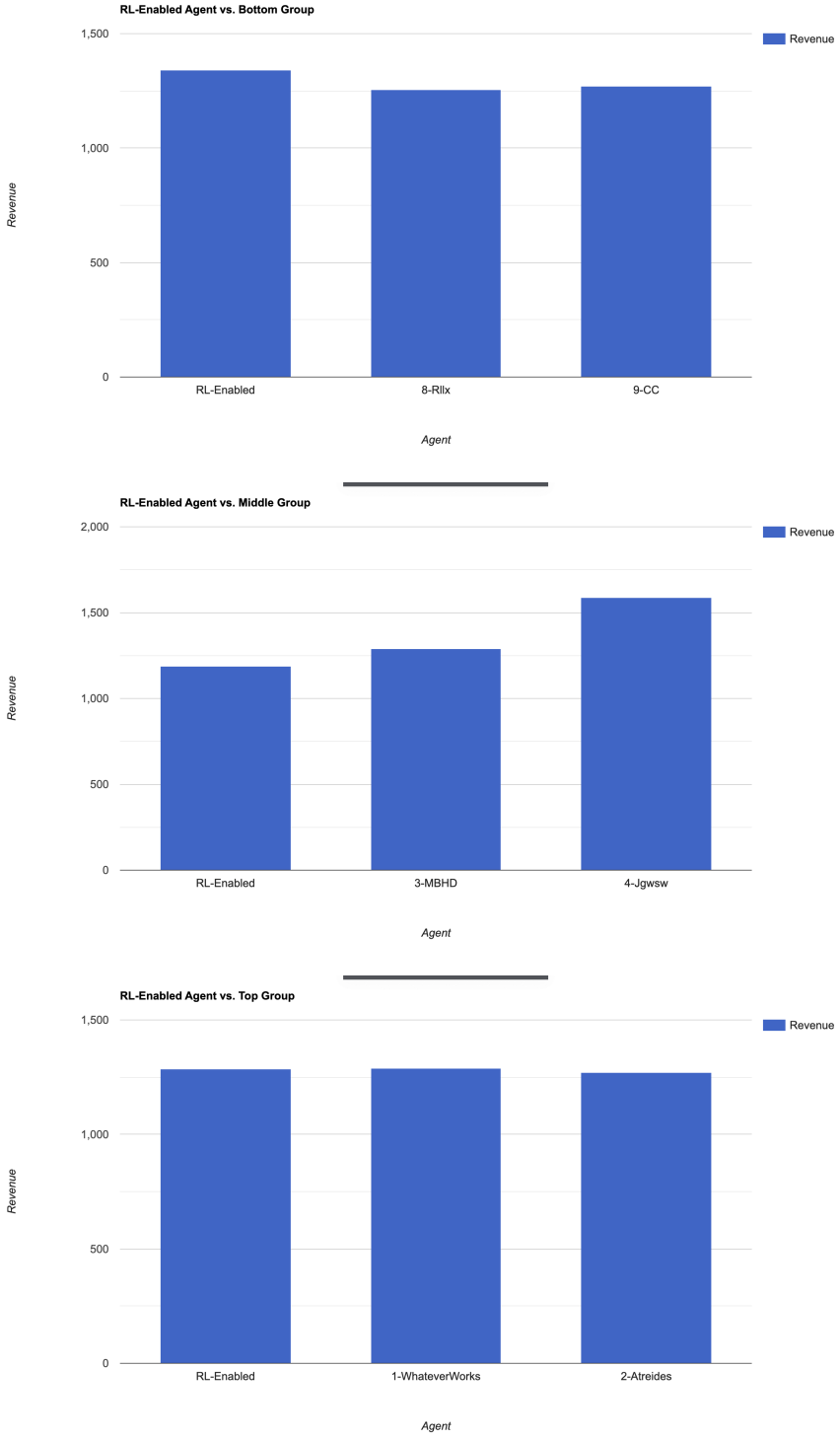
We chose the smaller group size of three because it ran much faster, and therefore it was easier to tune hyperparameters of the RL-based agent. We found that in groups of four and five, the RL-based agent was significantly less successful, but we hypothesize that this is simply because the long execution time for groups of four and five agents precluded us from being able to substantially tune to the hyperparameters of the agent.

First, in order to show the transition from exploration to exploitation over time, we ran the RL-enabled agent against the top two agents from the top two agents from the class tournament over varying numbers of iterations. Below, we can observe a graph of the average daily revenues for each agent, with trials lasting 25, 100, 250, 500, and finally 1000 days. We expect to see that the longer the auction, the higher the revenue for the RL-enabled agent, as it is able to exploit its learned values for more days, and better compensate for the initially unprofitable period of exploration.



This is almost exactly what we see, although rather than continuing to increase, the revenue for the RL-enabled agent converges to approximately 690 somewhere between the 250 and 500 day auction lengths. The proximity of the three lines past the 250-day auction length suggests that while the RL-enabled agent is learning meaningful strategy over its exploratory period, even the best strategy it learns is only enough to equal the top two agents from the class, not to outperform them.

Oddly, however, the RL-enabled agent does not perform any better against the lower-finishing agents in the class tournament than it does against the top 2 agents. Below, we observe the performances of the RL-enabled agent against each of our three competition groups, with each competitor rank in the class tournament.



Oddly, we notice that the RL-enabled agent performs more or less equally to every agent it faces, except for the fourth-ranked agent in the class tournament: Jgwswbudget, which outperforms the RL-enabled agent by almost 30%! Yet, while the figures are omitted for brevity, in every group we

see the same increase in revenues garnered by the RL-enabled agent as the number of days in the auction increases, showing us that the RL-enabled agent does learn better and better strategies as the auction progresses, irrespective of opponent. Therefore, we can make several conjectures. Firstly, it's possible that the Jgswsbudget agent is in fact the best one, and that it was defeated in the tournament by an unfortunate combination of opponents' strategy interactions. Secondly, it's possible that most of the opposing agents were specifically tailored to the 5-in-a-group setup in some way, and that they suffered when transferred over to the competition setting with only 3 agents, whereas Jgwswbudget may be agnostic to the number of opponents.

3 Conclusions & Future Work

In conclusion, we have presented the implementation of a Q-Learning based framework for bidding in a repeated GSP ad auction, with mixed results. While the Q-Learning based framework manages to perform on par with most of the top 10 'dumb' agents submitted in the class tournament, it does not significantly outperform any of them, as might be expected, and is surprisingly, much worse than the fourth-placed agent. Further work can be done in investigating why this might be the case, perhaps by better visualizing the Q-values learned for each state to expose any inconsistencies. Moreover, it would be interested to explore more sophisticated methods of learning the value function than the simple Q-Table, perhaps with a deep Q-Network which would allow us to approximate a continuous value function.

References

[1] Han Cai, Kan Ren, Weinan Zhang, Kleanthis Malialis, Jun Wang, Yong Yu, Defeng Guo. (2017) Real-Time Bidding by Reinforcement Learning in Display Advertising. arXiv:1701.02490 [cs.LG]