Reinforcement Learning versus a Genetic Algorithm for Dynamic Ride Surge Pricing

Alabi Taiwo Oluwatamilore
Student ID: x24335592
H9AIDM: Artificial Intelligence Driven Decision Making
National College of Ireland
x24335592@student.ncirl.ie

Video presentation: https://youtu.be/Q_6xYLD4ykA

Abstract

Surge pricing is an algorithm that decides to charge a multiplier m, because of the existing situation. For example, riders are willing to pay if it is raining, plus there is a Friday evening rush, plus the demand is high as well. Ride platforms must choose a price multiplier that maximises revenue without pushing so many riders away that revenue falls. This project formulates the problem as a Markov Decision Process over a 24-state description of market conditions, and compares two decision-making approaches; both: tabular Q-learning; reinforcement learning, and a genetic algorithm. Against two baselines; a fixed price, and a rule-based surge heuristic. I built a generative simulator from 638,000 real Boston ride records, so the agents can be trained and tested safely. It runs across eight seeded worlds; my observation is both learned strategies beat both baselines, the Q-learning policy raised average daily revenue by 18.2% and the genetic algorithm by 21.4% over the fixed baselines. Both methods independently discovered a state-dependent surge policy. Pricing up in high-demand and rainy conditions, as well as holding price in quiet periods.

Index Terms

reinforcement learning, Q-learning, genetic algorithm, Markov decision process, dynamic pricing, surge pricing, decision making, metaheuristics

I. Introduction

Ride platforms adjust prices in real time through a 'surge multiplier' applied to a base fare. The decision is not so easy because demand reacts to price. If the multiplier is too high, it suppresses bookings, while a low multiplier leaves revenue on the table during peaks. The right multiplier depends on the current market state; time of the day, weather, demand. So a single fixed rule is sub-optimal.

This work treats surge pricing as an automated decision-making problem and asks:

- can an agent learn a state-dependent pricing policy that outperforms simple baselines,
- and how does a learning-based method (RL) and a search-based method (GA) compare?

My contribution is (i) a Markov Decision Process formulation of surge pricing over an interpretable state space, (ii) a data-driven simulator, (iii) implementations of Q-learning, as well as a genetic algorithm. (iv) a comparison against fixed and rule-based baselines.

II. Related Work

Reinforcement learning tackles problems where an agent must make repeated decisions over time not knowing exactly what will happen next and learn from the results [1]. Q-learning is a model-free algorithm, meaning it learns the best action for each situation purely from experience, without needing to know the rules of the environment in advance [2]. Given enough exploration, it is guaranteed to find the optimal action-value estimates, making it a natural fit for my 24-state pricing environment here. Genetic algorithms [3] keep a whole population of candidate solutions, applied here to search the space of all possible 24-state pricing policies directly and improve them over generations through selection, crossover and mutation [4]. They simply breed the better solutions and discard the weaker ones. Revenue management and dynamic pricing are not new ideas; airlines and hotels have used them for years. Surge pricing is a version of this applied to ride-hailing, where the price adjusts in real time to balance

how many riders want a ride against how many drivers are available. This project tests whether an RL agent and a genetic algorithm can learn a better surge pricing policy than simple human rules.

III. Problem Formulation (Model)

The problem is modelled as a finite Markov Decision Process (S, A, P, R).

A. State (S)

Each hour is described by three discrete variables derived from the data

- *time bucket* $\in$ {morning_rush (07-09), day (10-15), evening_rush (16-19), night (else)};
- *weather* $\in$ {dry, wet} (wet if measured rain is greater than 0);
- *demand level* $\in$ {low, mid, high}

Combining the three variables gives $4 \times 2 \times 3 = 24$ possible states. This was kept small on purpose: 24 states $\times$ 5 actions $= 120$ Q-table entries, small enough to read and inspect directly.

B. Action (A)

A discrete set of five surge multipliers,

$$A = \{1.00,\ 1.25,\ 1.50,\ 1.75,\ 2.00\}.$$

C. Reward (R)

The reward for charging multiplier m in state s is:

$$\begin{aligned} \text{reward}(s, m) = m \cdot \text{base_fare}(s) \\ \cdot \text{rides_per_hour}(s) \cdot \text{acceptance}(m, s) \end{aligned} \tag{1}$$

$\text{base_fare}(s)$ is the average ride fare in that state with the historical surge removed, it is calculated as price $\div$ surge_multiplier. This step matters because the raw price in the dataset already has a surge baked in. Dividing it out gives the clean base fare, so the agent applies its own multiplier from scratch rather than on top of one that is already there. $\text{rides_per_hour}(s)$ is the average number of rides that happen per hour in that state.

D. Demand response (K modelling)

One thing the data cannot tell directly is how sensitive riders are to price changes. The dataset only shows demand at the surge levels Uber actually charged, we never see what demand would have been at a different price on the same day. On top of that, Uber tended to raise prices precisely when demand was already high, so high surges and high demand appear together in the data in a way that makes it impossible to cleanly separate cause and effect.

Because of this, a linear acceptance model is assumed:

$$\text{acceptance}(m, s) = \max\big(0,\ 1 - K(s) \cdot (m - 1)\big) \tag{2}$$

K gets calculated by multiplying things together:

$$K = \text{base} \times \text{wet_factor} \times \text{rush_factor}$$

So for low demand, wet, rush hour: $K = 0.8 \times 0.7 \times 0.8 = 0.448$. K is fine here, but stacking multiplications can push K to extremes, so I set:

- **Lower bound to 0.15** if K gets too close to zero, it means riders would pay literally any price no matter how high. Which to be honest is not realistic. Even the most desperate rider has a limit. So K is never allowed to drop below 0.15.
- **Upper bound to 1.2** if K gets too high, even a tiny 1.1x surge drives away almost everyone. It is also not realistic. So K is never allowed to go above 1.2.

In summary $K(s)$ controls how fast demand drops as the price rises. It is made state-dependent to reflect the reality that not all riders react the same way. A rider stuck in the rain during rush hour needs the ride and will absorb a surge; a rider on a quiet dry afternoon has other options.

E. Transition (P)

After each hour, the clock moves forward by one step. The next hour's weather and demand levels are not carried over directly, they are sampled fresh from probabilities estimated from the data:

- how often each time slot is wet, and
- how likely each demand level is, given the current time and weather.

Charging a high multiplier does not affect what the next state will be, since weather and demand follow their own patterns. One episode is one 24-hour day.

IV. Methodology

A. The simulator (environment)

Running an untrained agent on a real platform is not an option, so a simulator was built from the cleaned data instead.

The simulator stores four lookup tables computed from the data: the average base fare per state, the average rides per hour per state, the probability of rain given the time of day, and the probability of each demand level given the time and weather. At each step, an agent picks a multiplier, the simulator looks up the relevant values, calculates the reward, and moves the clock forward one hour. It returns the next state, the revenue earned, and a flag indicating whether the 24-hour day is finished. A fixed random seed ensures that any evaluation run produces the same result every time.

B. Approach 1: Q-learning (Reinforcement Learning)

Q-learning works by keeping a table with one row per state and one column per multiplier. Each cell holds the agent's current estimate of how much long-run revenue that multiplier earns in that state. The table starts at zero and gets updated after every hour of training using [2]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a) \right] \tag{3}$$

the bracket is the gap between what the table currently says and the target. α (learning rate) $= 0.1$ means only 10% of that gap is applied as a correction each time, cautious enough not to overreact to any single result. γ (discount) $= 0.9$ means next hour's expected revenue counts as 90% of this hour's, so the agent cares about the rest of the day but weights the near future more, appropriate for a 24-hour episode where decisions in early hours shape or can affect conditions later.

The agent does not always pick the highest-value action. ε starts at 1.0, fully random, the agent knows nothing yet, and decays by 0.999 each episode until it hits a floor of 0.05. Early on it mostly explores, trying all five multipliers in all situations. By episode 3,000 it mostly exploits what it has learned, with a 5% exploration kept to avoid getting permanently stuck on a suboptimal habit.

I must state that the next hour's weather and demand do not change based on what price was charged. The best action in each state therefore comes down to whichever multiplier earns the most revenue right now.

C. Approach 2: Genetic Algorithm (Metaheuristic)

The genetic algorithm approaches the problem differently. Instead of updating one state at a time, it evolves a genome: a length-24 vector that assigns one multiplier index to every state at once, giving a complete pricing policy in a single structure. Fitness is measured as average daily revenue over ten fixed-seed simulator days. Each generation runs four steps: tournament selection; three candidates compete, the fittest becomes a parent, uniform crossover for each of the 24 state-positions, the child inherits from either parent with equal probability, mutation; each position is randomly replaced with probability 0.1, and elitism; the top two genomes carry over unchanged.

D. Baselines

Two non-learning strategies serve as my baseline. A fixed baseline charges 1.0x at every state regardless of conditions. A rule-based baseline applies a surge of 1.5x whenever it is wet or a rush period, and 1.0x otherwise. Together they establish what a no-AI approach can earn before any learned policy/system is introduced.

TABLE I
AVERAGE DAILY REVENUE BY STRATEGY

Strategy	**Avg daily revenue ($)**	**Std ($)**	**Uplift vs Fixed**
Fixed: always 1.0×	704,646	6,797	0.0%
Rule: 1.5× if rain/rush	760,117	9,098	+7.9%
RL: Q-learning	833,103	13,465	+18.2%
GA: evolved	855,244	13,442	+21.4%

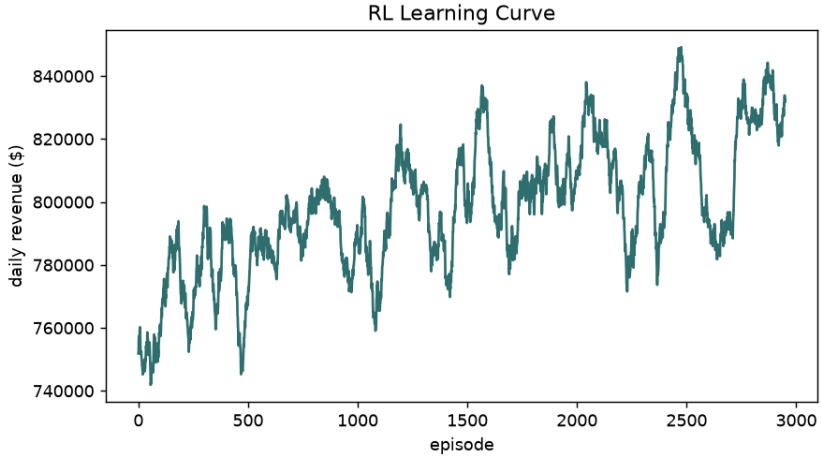


Fig. 1. RL Learning Curve: average daily revenue per training episode. The Q-table reaches a stable policy over 3,000 episodes.

V. EXPERIMENTAL SETUP

The experiments use the public Uber and Lyft Boston Cab Prices dataset [5]. Cleaning removed 55,095 Taxi-tier rows (8%) that carry no quoted price, leaving 637,976 rides. A millisecond-to-second timestamp fix was applied before the weather join, and duplicate weather readings were collapsed to one entry per location-hour, reducing them from 6,276 to 3,960 rows. All four strategies are compared on the same seeded evaluation: 8 seeds $\times$ 30 days per seed, so each strategy faces the exact same sequence of market conditions. Revenue is reported as mean $\pm$ standard deviation across seeds. My codebase is Python: pandas, NumPy, Matplotlib, with a single config file holding every constant and all randomness.

VI. RESULTS AND DISCUSSION

Table I and Fig. 3 together show the full picture: both learned strategies clearly outperform both baselines. The standard deviations are small relative to the means. The rule-based heuristic gains +7.9% over the fixed price by conditioning on weather and time of day, confirming that market conditions carry real pricing signals. The learned methods roughly double that gain by tuning the multiplier per state rather than applying a single surge level across all rain or rush conditions. Fig. 1 shows the RL agent's average daily revenue rising and converging over 3,000 training episodes, confirming the Q-table reaches a stable policy.

The genetic algorithm edges Q-learning +21.4% vs +18.2%. The GA's advantage comes from how it learns: it evaluates all 24 states simultaneously as a single policy unit, so even rarely-visited state combinations are shaped by the fitness signal.

VII. CONCLUSION AND FUTURE WORK

Both methods answered my research question: a Q-learning agent and a genetic algorithm can both learn state-dependent pricing policies that outperform simple baselines, achieving 18.2% and 21.4% higher revenue than the

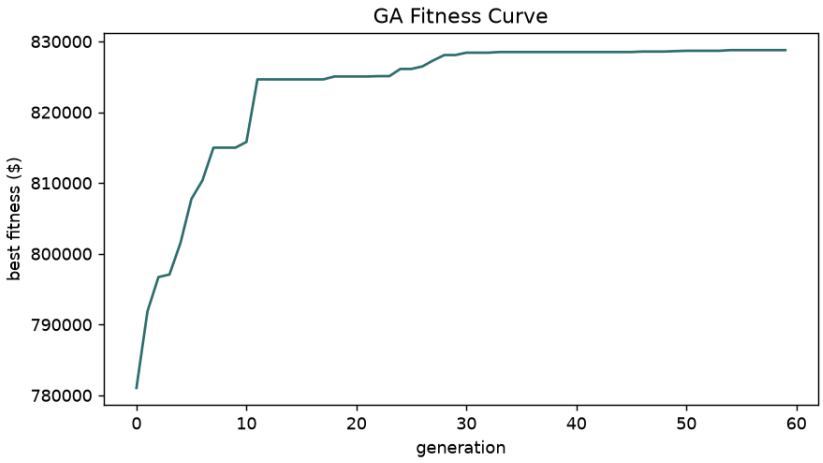


Fig. 2. GA Fitness Curve: best policy fitness per generation. The curve climbs steeply then plateaus near generation 30.

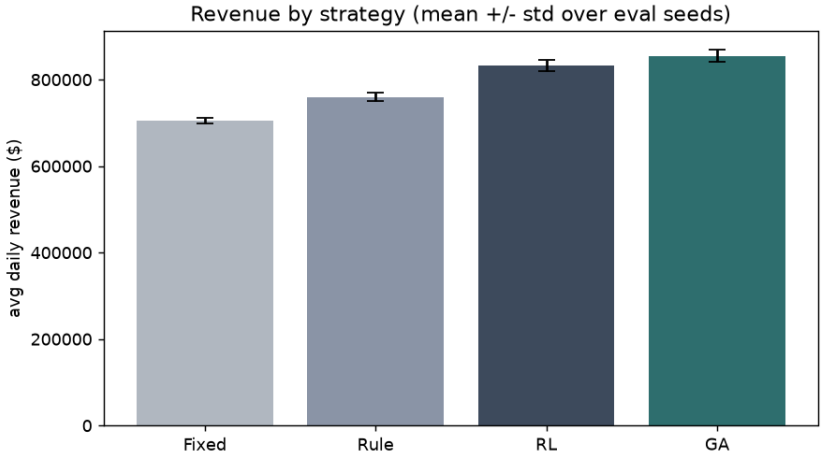


Fig. 3. Revenue Comparison: average daily revenue for all four strategies.

fixed price. Both independently discovered the same pricing structure: a surge in high-demand and rainy conditions, and holding the price in dry periods, without being told any rule. The GA produced the better result here, although Q-learning's online update mechanism makes it more adaptable to a live, shifting market.

There are two limitations. First, the demand model, K, the function that controls how fast bookings drop as price rises, is assumed; I modelled it myself. Second, the state and action spaces are deliberately small and discrete, which keeps the problem interpretable but limits how precisely the policy can respond to market conditions.

Distribution of Work

This project was completed individually. All problem formulation, data cleaning, simulator design, algorithm implementation, evaluation and writing are my own work.

Video Presentation

A five-minute video walkthrough of this project is available at: https://youtu.be/Q_6xYLD4ykA

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
[2] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, May 1992.

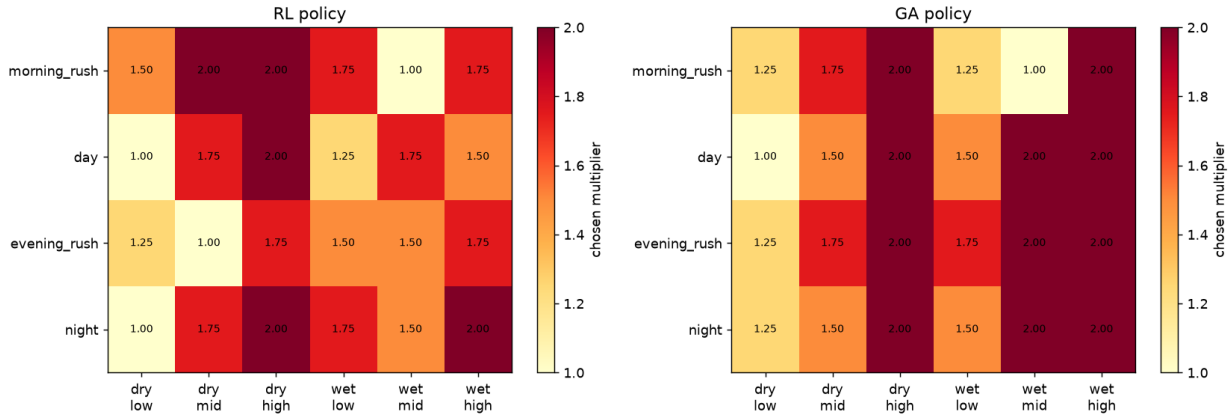


Fig. 4. Policy Heatmaps: shows the chosen multiplier per state for RL (left) and GA (right). Both agents independently discovered a high surge in high-demand and rainy conditions, and a lower surge when it is dry. They both found this structure without being given any pricing rule.

[3] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: Univ. of Michigan Press, 1975.
[4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA, USA: Addison-Wesley, 1989.
[5] R. Munde, "Uber and lyft cab prices," Kaggle dataset, 2019, available: https://www.kaggle.com/datasets/ravi72munde/uber-lyft-cab-prices.