AI Lab Report-8
Policy Iteration for Gbike Bicycle Rental Problem with Free Shuttle and Parking Constraints

Anuj Saha
Dept. of CSE, IIIT Vadodara
Gandhinagar, India
202351010@iiitvadodara.ac.in

G. Nikhil
Dept. of CSE, IIIT Vadodara
Gandhinagar, India
202351037@iiitvadodara.ac.in

Divyanshu Ghosh
Dept. of CSE, IIIT Vadodara
Gandhinagar, India
202351036@iiitvadodara.ac.in

***Abstract*—In this report, we present our complete solution to the Gbike Bicycle Rental Problem using the Policy Iteration algorithm, which is a classical dynamic programming method in Reinforcement Learning. We modeled the problem as a continuing finite Markov Decision Process (MDP) and solved two variants: first, the basic version with symmetric movement cost of INR 2 per bike, and second, a realistic modified version incorporating a free one-way employee shuttle and an extra parking cost when more than 10 bikes are kept overnight. Our implementation successfully computed optimal policies, revealing that small operational incentives can significantly alter decision-making and, surprisingly, can even reduce long-term expected profit due to conflicting constraints. Complete code implementation can be found at https://github.com/AnujSaha0111/CS307-Lab-Submissions/tree/main/Submission_8**

***Index Terms*—Reinforcement Learning, Markov Decision Process, Policy Iteration, Dynamic Programming, Resource Allocation.**

I. Introduction

The Gbike bicycle-sharing system operates two locations with stochastic daily rental requests and returns. Each night, the manager must decide how many bikes to move between locations to maximize long-term profit. This is a sequential decision-making problem under uncertainty, which is ideally suited for modeling as a Markov Decision Process (MDP) and solving using Policy Iteration.

In this report, we describe how we formulated this real-world problem as an MDP, implemented the Policy Iteration algorithm, and analyzed the results for both the basic and modified versions of the problem. Our findings provide interesting insights into how operational constraints can interact in unexpected ways.

II. MDP Formulation

We formulated the problem as a continuing finite MDP with the following components:

- **States** $s = (b_1, b_2)$: The number of bikes at Location 1 and Location 2 at the end of the day. We have $b_1, b_2 \in \{0, 1, \ldots, 20\}$, which gives us $|S| = 441$ total states.
- **Actions** a: The net number of bikes moved overnight from Location 1 to Location 2. We allow $a \in \{-5, \ldots, 5\}$, constrained by available bikes and a maximum movement limit of 5 bikes.
- **Reward Function**:
  - We earn +10 INR per bike rented during the day.
  - We incur a movement cost of 2 INR per bike (except the first bike from Location 1 to Location 2 is free in the modified version).
  - We pay an extra parking cost of 4 INR if any location has more than 10 bikes after moving.
- **Transition Dynamics**:
  - Morning inventory after movement: $(b_1 - a, b_2 + a)$.
  - Rental requests follow Poisson distributions: Poisson($\lambda = 3$) at Location 1, Poisson($\lambda = 4$) at Location 2.
  - Returns also follow Poisson distributions: Poisson($\lambda = 3$) at Location 1, Poisson($\lambda = 2$) at Location 2.
  - The next state is computed as $\min(\text{inventory} - \text{rentals} + \text{returns}, 20)$.
- **Discount Factor**: We used $\gamma = 0.9$ to balance immediate and future rewards.

III. Implementation: Policy Iteration

We implemented the Policy Iteration algorithm by alternating between policy evaluation and policy improvement until convergence. The overall framework is shown below.

Algorithm 1 Policy Iteration Framework

```
1: Initialize policy $\pi(s) = 0$ and value function $V(s) = 0$ $\forall s$.
2: repeat
3:     Evaluate current policy $\pi$ to compute $V^\pi(s)$.
4:     Improve policy: $\pi(s) \leftarrow \arg\max_a Q^\pi(s, a)$.
5: until policy does not change.
6: Return optimal policy $\pi$ and value function $V$.
```

Explanation: This algorithm represents the core Policy Iteration loop. We start with an arbitrary policy (moving zero bikes everywhere), then repeatedly evaluate how good that policy is and improve it by choosing better actions. We continue this process until the policy stops changing, which guarantees we have found the optimal policy.

A. Computing Expected Returns

For the basic version, we computed the expected return for each state-action pair as follows:

Algorithm 2 Expected Return — Basic Version

```
Input: State $(b_1, b_2)$, action $a$, current value function $V$.
1: Compute movement cost = $2 \times |a|$.
2: Compute morning inventory = $(\max(b_1 - a, 0), \min(b_2 + a, 20))$.
3: Add expected rental reward from precomputed tables.
4: Add discounted future value: $\gamma \sum_{s'} P(s'|s,a)V(s')$.
Output: Total expected return.
```

Explanation: This function calculates how much reward we expect to get if we take action a in state (b_1, b_2) and follow our current policy afterward. We first subtract the cost of moving bikes, then add the expected rental income based on Poisson probabilities, and finally add the discounted value of future states we might end up in. We precomputed the rental rewards for efficiency.

For the modified version with additional constraints, we used:

Algorithm 3 Expected Return — Modified Version (Free Shuttle + Parking Cost)

```
Input: State $(b_1, b_2)$, action $a$, current value function $V$.
1: if $a > 0$ then
2:     Movement cost = $2 \times (a - 1)$     {First bike free!}
3: else
4:     Movement cost = $2 \times |a|$.
5: end if
6: Compute morning inventory.
7: if morning bikes at Location 1 $> 10$ or Location 2 $> 10$ then
8:     Subtract 4 INR (extra parking cost).
9: end if
10: Add rental rewards + discounted future value.
Output: Total expected return under modified rules.
```

Explanation: This modified version incorporates two realistic business constraints. First, we have a free employee shuttle that can move one bike from Location 1 to Location 2 at no cost, so we only charge for additional bikes moved in that direction. Second, we penalize keeping too many bikes at any location overnight (more than 10) with a parking fee. These changes significantly affect the optimal policy, as we will see in the results.

B. Policy Evaluation

We implemented iterative policy evaluation to compute the value function for a given policy:

Algorithm 4 Iterative Policy Evaluation

```
1: repeat
2:     $\Delta \leftarrow 0$.
3:     for all states $(i, j)$ do
4:         $v \leftarrow V(i, j)$.
5:         $V(i, j) \leftarrow$ expected return under current $\pi(i, j)$.
6:         $\Delta \leftarrow \max(\Delta, |v - V(i, j)|)$.
7:     end for
8: until $\Delta < 0.1$.
```

Explanation: This is where we evaluate how good our current policy is. We sweep through all states and update their values based on the expected return of following the current policy. We keep doing this until the changes become very small (less than 0.1), which means the values have converged. This gives us an accurate estimate of the long-term value of each state under the current policy.

C. Policy Improvement

After evaluating the policy, we improved it by choosing the best action at each state:

Algorithm 5 Policy Improvement

```
1: for all states $(i, j)$ do
2:     Try all valid actions $a \in [\text{min\_action}, \text{max\_action}]$.
3:     Choose $a^* = \arg\max_a$ expected return.
4:     Update $\pi(i, j) \leftarrow a^*$.
5: end for
6: if policy unchanged then
7:     Converged.
8: end if
```

Explanation: In this step, we make the policy better (greedy improvement). For each state, we try all possible actions and pick the one that gives the highest expected return based on our current value estimates. If no state changes its action, we know we have reached the optimal policy and can stop iterating.

IV. Results

We ran our Policy Iteration implementation for both versions of the problem and obtained the optimal policies shown in Figure 1.

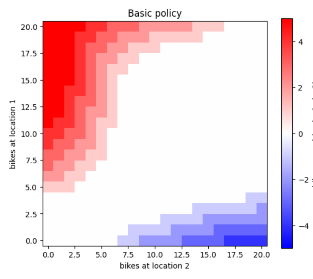
TABLE I: Sample Optimal Actions We Obtained

State (Loc1, Loc2)	Basic Policy	Modified Policy
(10, 10)	0	0
(0, 20)	-4	-5
(20, 0)	$+5$	$+5$

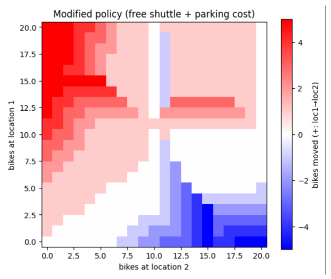
A. Key Findings

From our analysis, we discovered several interesting patterns:

- Out of 441 states, 238 states have different optimal actions between the two versions. This shows that the modifications significantly changed the optimal behavior.



(a) Basic Optimal Policy



(b) Modified Optimal Policy

Fig. 1: Optimal policies we computed. Positive values (blue): move bikes from Location 1 to Location 2. Negative (red): move bikes back. The modified policy is more aggressive due to the free shuttle incentive.

- Surprisingly, the average value function in the modified version was **1.70 INR lower** than in the basic version. This means that despite having a free shuttle, the modified system actually makes less profit overall.
- The free shuttle encourages moving more bikes to high-demand Location 2. However, the parking penalty discourages keeping too many bikes at any location. These two incentives conflict with each other.
- As a result, the ”free help” from the employee shuttle actually reduces overall profit. This counterintuitive finding highlights how different operational constraints can interact in unexpected ways.

V. Conclusion

In this lab, we successfully applied the Policy Iteration algorithm to solve a real-world resource allocation problem under realistic constraints. Through our implementation and analysis, we demonstrated that:

- Optimal policies are highly sensitive to small changes in reward structure. Even minor modifications like a free shuttle or parking cost can drastically change the optimal strategy.
- Local incentives (like the free employee shuttle) can actually backfire when misaligned with other constraints. What seems like a cost-saving measure can reduce overall profitability.
- Dynamic Programming remains a powerful and reliable tool for solving finite MDPs with known models. Despite the problem’s complexity (441 states), we were able to compute exact optimal policies.

This exercise provided us with deep insight into modeling and solving stochastic sequential decision problems using formal Reinforcement Learning methods. We learned that optimal decision-making under uncertainty requires careful consideration of how different factors interact, and that intuitive solutions may not always be optimal.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.

[2] R. A. Howard, *Dynamic Programming and Markov Processes*. New York, NY: John Wiley & Sons, 1960. The original work introducing Policy Iteration.

[3] A. Singla, H. H. Ghosh, A. Ghosh, and D. Ghosh, ”Incentivizing users to rebalance bike sharing systems,” in *Proc. Twenty-Ninth AAAI Conf. on Artificial Intelligence (AAAI)*, 2015. Relevant for real-world constraints and incentives in bike-sharing rebalancing.

[4] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY: John Wiley & Sons, 1994. A definitive, advanced textbook on MDPs and Policy/Value Iteration techniques.