

Scaling Dynamic Programming on Finite MDPs

Jack Vuturo

Abstract—This paper investigates adaptations of Jack’s Car Rental problem, as formulated in Chapter 4 of Sutton and Barto’s Reinforcement Learning: An Introduction, as a vehicle for examining the behavior and scalability of dynamic programming methods applied to finite Markov decision processes. Four experiments are presented. The base problem is replicated and the resulting optimal policy and value function are analyzed. The modified cost structure described in Exercise 4.7 is then implemented, introducing non-linearities that produce measurable shifts in the optimal policy. A scalability analysis compares synchronous and asynchronous policy evaluation across progressively larger state spaces, demonstrating that asynchronous evaluation offers a consistent efficiency advantage that widens as the state space grows. Finally, the agent’s action space is extended to include dynamic pricing decisions, transforming the problem into a joint inventory-and-pricing optimization task governed by a constant elasticity demand model. Results confirm that dynamic programming reliably converges to interpretable optimal policies in tractable problem formulations, but that computational cost grows prohibitively as the state space expands, motivating the use of approximation methods for real-world applications.

Index Terms—Reinforcement Learning, Markov Decision Processes, Dynamic Programming, Policy Iteration, Synchronous Policy Evaluation, Asynchronous Policy Evaluation, Jack’s Car Rental, Price-Demand Elasticity, Inventory Optimization, State Space Scalability



1 INTRODUCTION

THIS paper explores adaptations of Jack’s Car Rental problem as presented in Chapter 4 of Reinforcement Learning - An Introduction Second Edition by Richard S. Sutton and Andrew G. Barto. Jack’s Car Rental problem is written as an optimal problem to solve with dynamic programming (DP), the problem has discrete steps and limited boundaries allowing a DP algorithm to easily find the optimal solution with a relatively small number of iterations. The problem is described below:

Jack manages two locations for a nationwide car rental company. Each day, some number of customers arrive at each location to rent cars. If Jack has a car available, he rents it out and is credited \$10 by the national company. If he is out of cars at that location, then the business is lost. Cars become available for renting the day after they are returned. To help ensure that cars are available where they are needed, Jack can move them between the two locations overnight, at a cost of \$2 per car moved. We assume that the number of cars requested and returned at each location are Poisson random variables, meaning that the probability that the number is n is

$\frac{\lambda^n}{n!}e^{-\lambda}$, where λ is the expected number. Suppose is 3 and 4 for rental requests at the first and second locations and 3 and 2 for returns. To simplify the problem slightly, we assume that there can be no more than 20 cars at each location (any additional cars are returned to the nationwide company, and thus disappear from the problem) and a maximum of five cars can be moved from one location to the other in one night. We take the discount rate to be $\gamma = 0.9$ and formulate this as a continuing finite MDP, where the time steps are days, the state is the number of cars at each location at the end of the day, and the actions are the net numbers of cars moved between the two locations overnight. [1]

The 20-car limit per location constrains the state space to a tractable size, and the discrete nature of the problem permits exhaustive evaluation of all possible actions. This problem gives us a simple Finite MDP for our agent to explore. This finite MDP will serve as the basis for the following experiments as outlined in this paper. Modifications will be made to this simple MDP to explore its adaptability in addition to its scalability.

2 METHODOLOGY

This paper presents a series of experiments organized into two primary investigations of the Jack's Car Rental problem as formulated in Chapter 4 of Sutton and Barto. The first investigation replicates the original problem, visualizes the resulting optimal policy and value function, and then implements the modified cost structure described in Exercise 4.7. The second investigation examines the scalability limitations of dynamic programming when applied to progressively larger variants of the problem, comparing synchronous policy evaluation and asynchronous policy evaluation. A final extension augments the agent's action space to include pricing decisions, transforming the problem into a joint inventory-and-pricing optimization task.

All experiments are implemented in Python using NumPy for numerical computation and Matplotlib for visualization. Policy iteration, as described in Section 4.3 of Sutton and Barto, serves as the primary solution algorithm. Each iteration consists of two phases: policy evaluation, in which the value function is iteratively updated under the current policy until convergence, and policy improvement, in which a new policy is derived greedily from the updated value function. The algorithm terminates when the policy remains stable across consecutive improvement steps.

Convergence of the value function during policy evaluation is defined as the point at which the maximum absolute change across all states falls below a threshold of $\theta = 1 * 10^{-4}$ in a single sweep. All experiments use a discount factor of $\gamma = 0.9$, consistent with the original problem specification. For the scalability experiments in Section 2, the maximum number of cars per location is incrementally increased beyond the original bound of 20, and computational cost is measured in both computation time and total number of value function sweeps required for convergence. These metrics provide a basis for comparing the efficiency of each evaluation strategy as the state space grows.

Results are presented as heat maps of the value function over the state space, policy grids indicating the optimal number of cars to transfer for each state, and convergence plots tracking the value function's residual error across sweeps. Where applicable, policies derived under different methods or problem formulations are compared directly to assess whether they converge to equivalent solutions.

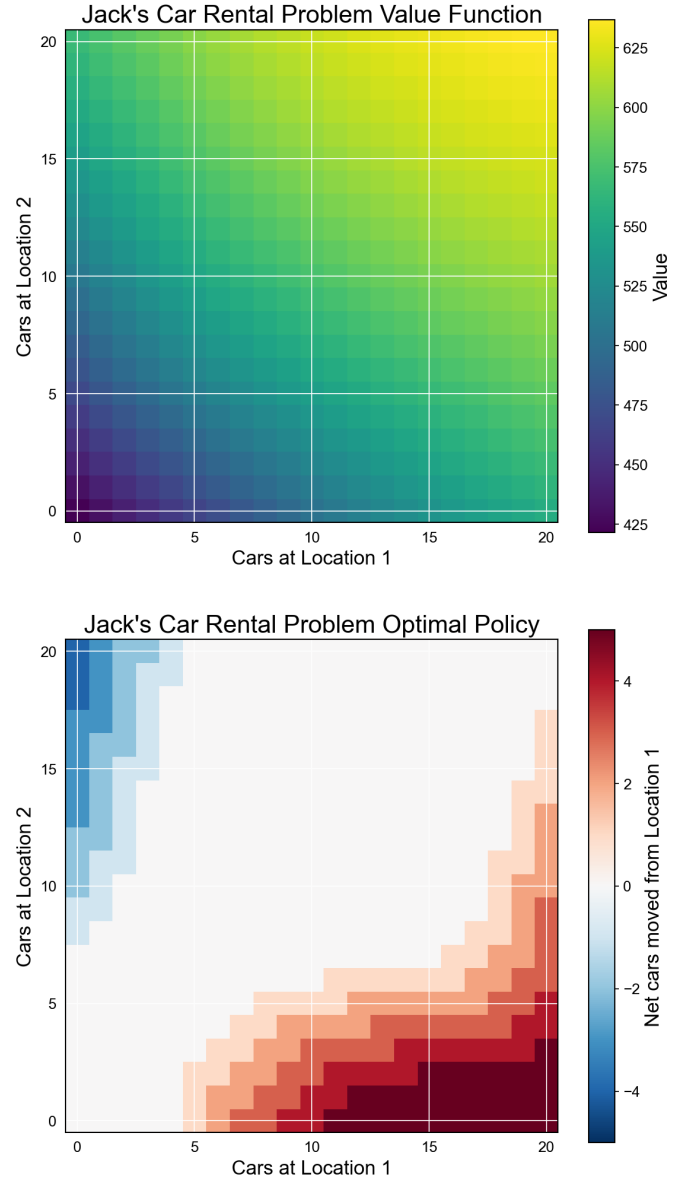


Fig. 1. Value function and optimal transfer policy for the base Jack's Car Rental problem under policy iteration. The upper panel depicts the value function $V(s)$ over the 21×21 state space, where state (i, j) represents i cars at Location 1 and j cars at Location 2. The lower panel displays the optimal policy, where positive values indicate cars transferred from Location 1 to Location 2 and negative values indicate the reverse.

3 THE BASE PROBLEM: POLICY STRUCTURE AND CONVERGENCE BEHAVIOR OF JACK'S CAR RENTAL PROBLEM

3.1 Base Formulation and Optimal Policy Analysis

The base Jack's Car Rental problem, as described in the introduction, defines a 441-state MDP over the combinations of inventory levels at two locations. The value function, shown in the upper panel of Figure 1, increases with total fleet size, reflecting the

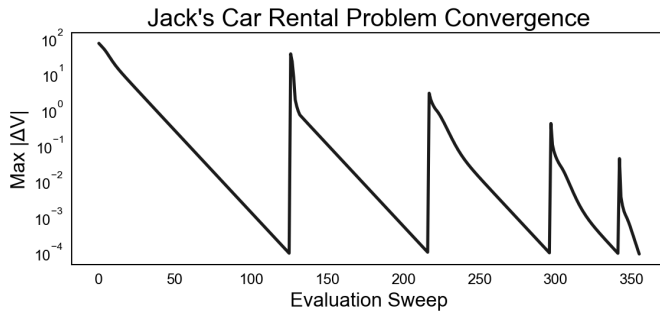


Fig. 2. Convergence of the value function during policy iteration for the base Jack's Car Rental problem. Each spike corresponds to the initialization of a new policy evaluation phase following a policy improvement step. The algorithm converges after approximately 350 evaluation sweeps.

greater expected revenue available to an agent with more cars to rent. The optimal policy, shown in the lower panel, exhibits a diagonal decision boundary separating states in which cars are moved from Location 1 to Location 2 from those in which the reverse transfer is preferred. This structure reflects the asymmetry in rental demand ($\gamma_1 = 3$, $\gamma_2 = 4$), with the agent preferring to maintain higher inventory at the higher-demand location. The policy iteration converges to the optimal policy in five improvement steps, requiring approximately 350 total evaluation sweeps, as shown in Figure 2.

3.2 Non-Linear Cost Structures and Their Effect on Optimal Policy

The Sutton Textbook also provides an expansion of the original problem, Exercise 4.7 is described below:

Write a program for policy iteration and resolve Jack's car rental problem with the following changes. One of Jack's employees at the first location rides a bus home each night and lives near the second location. She is happy to shuttle one car to the second location for free. Each additional car still costs \$2, as do all cars moved in the other direction. In addition, Jack has limited parking space at each location. If more than 10 cars are kept overnight at a location (after any moving of cars), then an additional cost of \$4 must be incurred to use a second parking lot (independent of how many cars are kept there). These sorts of non-linearities and arbitrary dynamics often occur in real problems and cannot easily be handled by optimization meth-

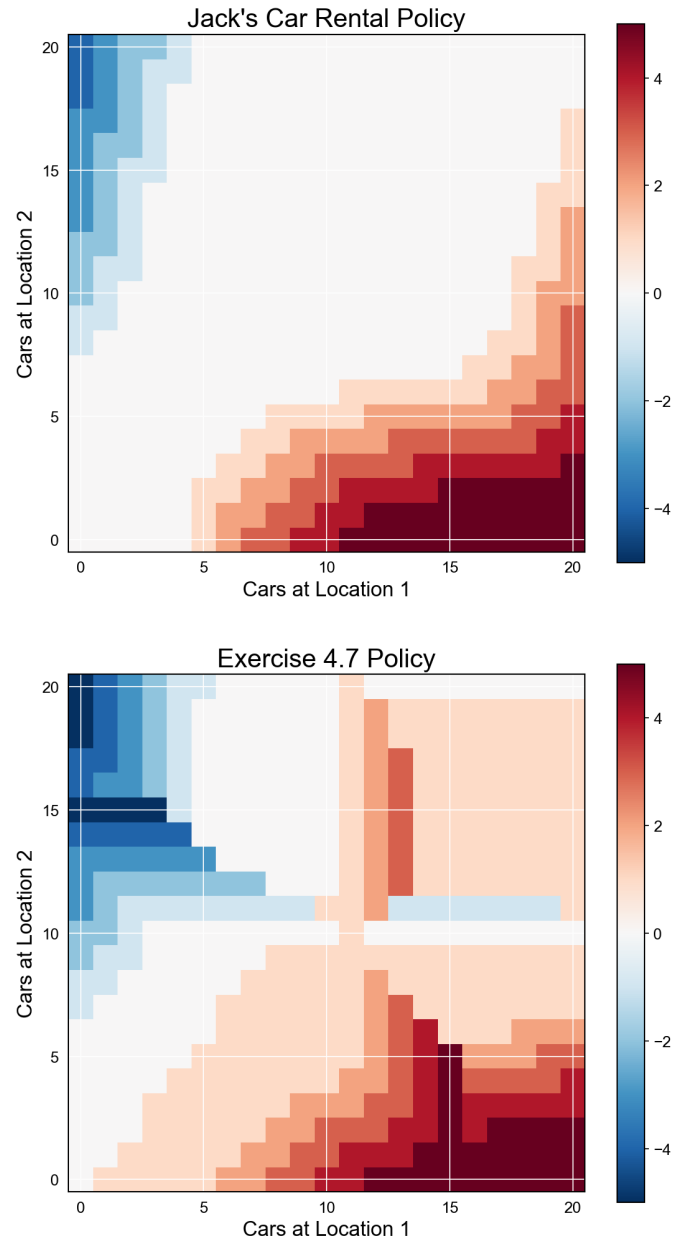


Fig. 3. Comparison of optimal transfer policies under the base formulation (upper) and the Exercise 4.7 cost structure (lower). The modified cost structure introduces a free one-car shuttle from Location 1 to Location 2 and a \$4 overnight parking penalty at locations holding more than 10 cars. The resulting policy exhibits a leftward shift, reflecting the agent's preference for transferring inventory toward Location 2 at reduced cost and avoiding excess overnight stock at either location.

ods other than dynamic programming. To check your program, first replicate the results given for the original problem. [1]

This extension fundamentally alters the cost structure of the problem by introducing two non-linearities: a free shuttle that breaks the symmetry of transfer costs, and a capacity-based parking penalty that discourages excess inventory at either location.

These modifications are expected to shift the optimal policy toward lower overnight car counts and preferential movement from location one to location two.

Looking at Figure 3 a jagged, non-monotonic boundary that doesn't appear in the base policy form in the optimal policy of Exercise 4.7. This boundary occurs due to the fact that the parking penalty cannot be expressed as a smooth function of the state, and the free shuttle breaks the symmetry that would otherwise permit analytical simplification. The resulting policy is irregular in a way that reflects the structure of the reward.

4 EXAMINING THE SCALABILITY OF DYNAMIC PROGRAMMING APPROACHES IN REAL WORLD SCENARIOS.

Jack's Car Rental problem serves primarily as a pedagogical tool. Its state space of 441 states and action space of 11 actions per state are orders of magnitude smaller than those encountered in real-world fleet management. Examining the scalability limits of the dynamic programming approaches applied to this problem is therefore essential for understanding their practical applicability.

Hertz, one of top rental car companies in the United States owns approximately 550,000 cars across 11,400 locations. [2] Naive assumptions can be made to simplify the problem, such as: a location can only hold up to 100 cars; the number of rentals are only dependent on the surrounding population; and so forth. Even with these naive assumptions, calculating an optimal policy is too computationally complex to solve with the strategies outlined in the Jack's Car Rental problem.

In addition to the issue of scalability, if a company were to invest in an agent like the one proposed in Jack's Car Rental problem they would likely want to see an agent that performs more tasks than just fleet management. For example, an agent that can dynamically price rentals to customers to maximize revenue streams. Adding functionality such as this further expands our state space, adding more computational complexity. This rest of this section will explore the various strategies to adapt and scale the Jack's Car Rental problem to be practicable in real world situations by expanding the state space by increasing the scale of the problem and by giving the agent the ability to set prices depending on the location.

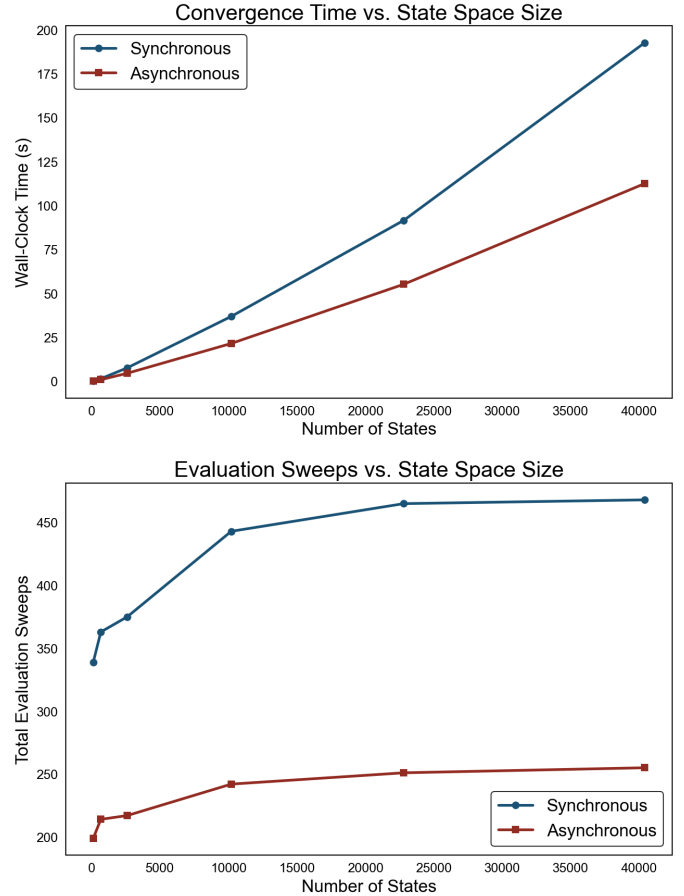


Fig. 4. Computational cost of synchronous and asynchronous policy evaluation as a function of state space size, measured in wall-clock time (upper) and total evaluation sweeps (lower). State space size is determined by the maximum car capacity N per location, with $N \in \{10, 25, 50, 100, 150, 200\}$ producing state spaces of size $(N + 1)^2$. Asynchronous evaluation converges in fewer sweeps and less time across all problem scales, with the performance gap widening as the state space grows.

4.1 Synchronous vs. Asynchronous Evaluation Under Growing State Spaces

In this experiment, the original Jack's Car Rental problem is extended by incrementally increasing the maximum number of cars permitted at each location beyond the original bound of 20, producing state spaces of size $(N + 1)^2$ for:

$$N \in \{10, 25, 50, 100, 150, 200\}$$

The goal is to characterize how computational cost scales under synchronous and asynchronous policy evaluation as the state space grows.

In synchronous policy evaluation, the value function is updated across all states simultaneously using values from the previous sweep, such that no updated estimate influences another within the same iteration. Formally, each sweep computes:

$$V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V_k(s')] \quad \forall s \in \mathcal{S}$$

In asynchronous policy evaluation, states are updated individually in place, and newly computed values are used immediately in subsequent updates within the same sweep. This allows information to propagate more rapidly across the state space, as a single sweep need not revisit a state before its updated value influences neighboring estimates.

As the state space grows, these two strategies are expected to diverge in efficiency. Asynchronous evaluation reduces redundant computation by exploiting already-updated values within a single pass, whereas synchronous evaluation must complete a full sweep before any new information is incorporated. Computational cost is measured along two dimensions: wall-clock time and total number of evaluation sweeps required for convergence, defined as the point at which the maximum absolute change across all states falls below $\theta = 1 \times 10^{-4}$.

The results in Figure 4 confirm this expectation. Both methods exhibit super-linear growth in wall-clock time as the state space increases, reflecting the $\mathcal{O}(|\mathcal{S}|^2)$ complexity of each evaluation sweep under Poisson transition dynamics. However, synchronous evaluation scales more steeply than asynchronous evaluation in both wall-clock time and sweep count, with the gap widening as the state space grows. These findings suggest that for moderately scaled-up versions of the problem, asynchronous evaluation offers a reasonable efficiency advantage, though neither method remains practical at scale.

4.2 Dynamic Pricing as an Extension of the Action Space

In this extension, the agent's action space is augmented to include price selection for car rentals at each location. Rather than earning a fixed \$10 per rental, the agent selects from a discrete set of prices, where higher prices yield greater per-rental revenue but reduce demand according to a price-demand elasticity model.

$$\text{Prices} = \{\$10, \$20, \$30, \dots, \$100\}$$

Using the price-demand elasticity model, the number of customers a location receives will be a function of the demand at that location and the price the agent sets. This method will be used instead

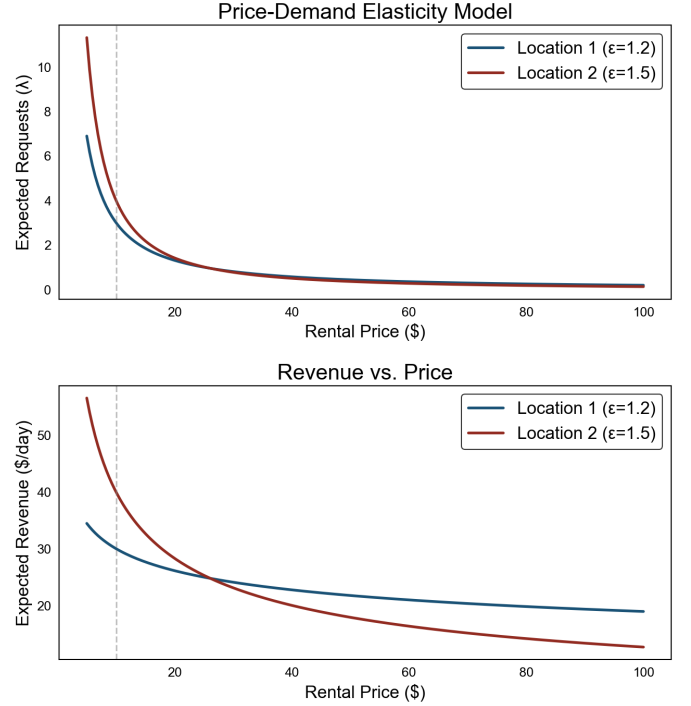
Section 4.2: Price-Demand Model ($\epsilon=1.2$, $\epsilon=1.5$)

Fig. 5. Price-demand elasticity model for both rental locations under the constant elasticity formulation $\lambda(p) = \lambda_{base} \left(\frac{p_{base}}{p} \right)^\epsilon$. The upper panel shows expected rental requests as a function of price, and the lower panel shows the corresponding expected daily revenue. Location 2 ($\epsilon = 1.5$) exhibits greater demand sensitivity to price than Location 1 ($\epsilon = 1.2$), resulting in a lower revenue-maximizing price point.

of the original Poisson model used in the original Jack's Car Rental problem.

Demand at each location is modeled as price-sensitive by scaling the base Poisson request rate according to a constant elasticity function:

$$\lambda(p) = \lambda_{base} \left(\frac{p_{base}}{p} \right)^\epsilon$$

$p_{base} = \$10$ corresponds to the original fixed rental price and epsilon represents the price elasticity of demand. The price elasticity of demand will be determined by the location. Location one will have an $\epsilon = 1.2$ and location two will have an $\epsilon = 1.5$.

Under this formulation, the expected number of rental requests decreases with price, while per-rental revenue increases, creating a revenue-maximization tradeoff that the agent must learn to balance alongside its inventory management decisions. As this formulation of the Jack's Car Rental problem is complex it would be expected that the agent takes a longer time to converge to an optimal

Section 4.2: Value Function and Transfer Policy

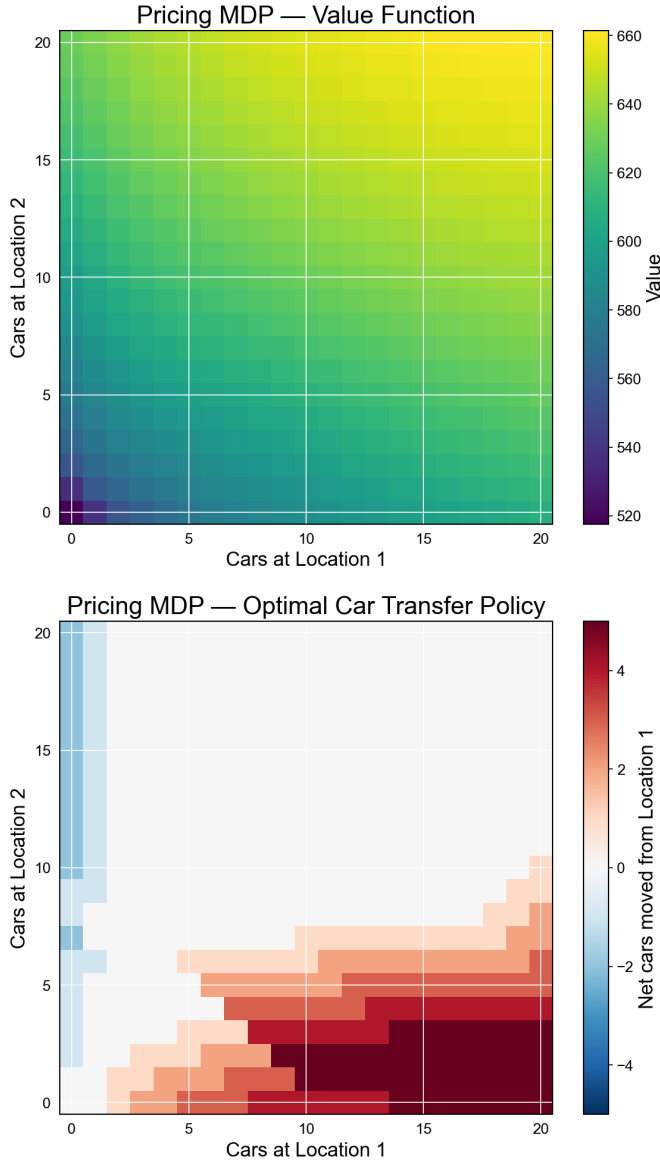


Fig. 6. Value function and optimal car transfer policy for the pricing MDP extension. The upper panel shows $V(s)$ over the 21×21 state space under the joint inventory-and-pricing formulation. The lower panel displays the corresponding optimal transfer policy $\pi^*(s)$. Relative to the base problem, elevated state values reflect the additional revenue potential of dynamic pricing, while the transfer policy retains the diagonal structure driven by the asymmetry in demand elasticity between locations ($\epsilon_1 = 1.2$, $\epsilon_2 = 1.5$).

policy, in comparison to the convergence of the original Jack's Car Rental problem and the extension in Exercise 4.7.

The value function and optimal transfer policy for the pricing MDP, shown in Figure 6, resemble those of the base problem but exhibit uniformly higher state values across the entire state space. This elevation reflects the additional revenue available to

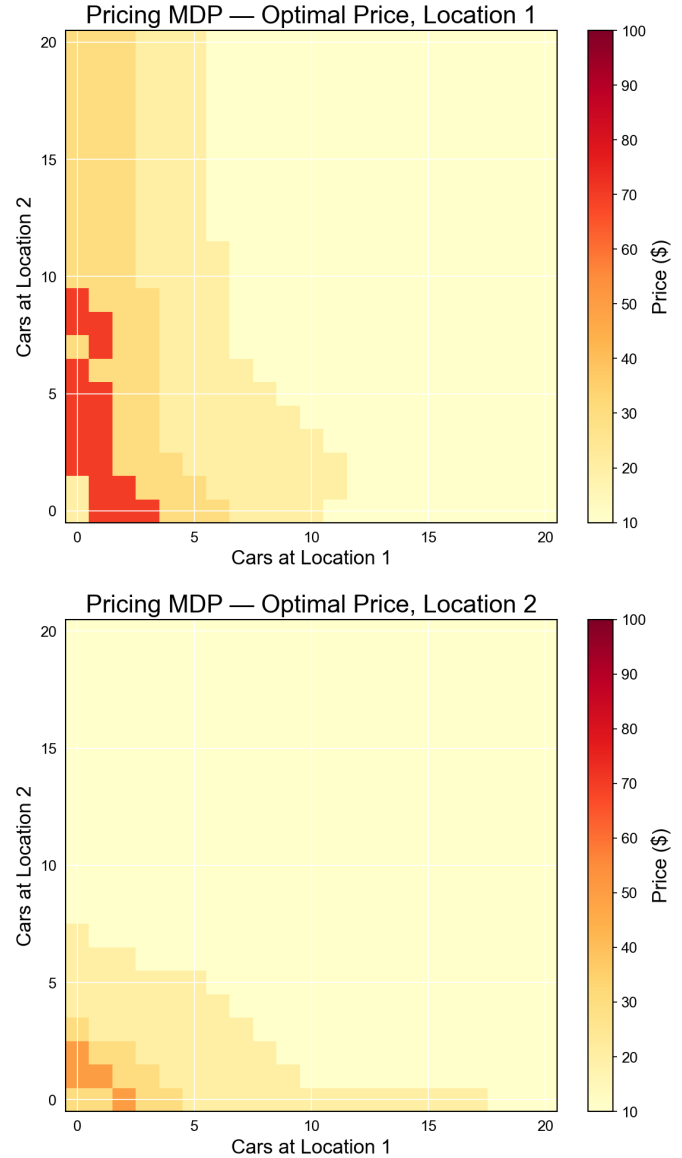
Section 4.2: Optimal Pricing Policies ($\epsilon_1=1.2$, $\epsilon_2=1.5$)

Fig. 7. Optimal pricing policies for Location 1 ($\epsilon = 1.2$) and Location 2 ($\epsilon = 1.5$) under the pricing MDP formulation. Each cell indicates the revenue-maximizing price selected by the agent for the corresponding state (i, j) . Low-inventory states favor reduced prices to maximize rental volume, while high-inventory states support elevated prices to extract greater per-rental revenue. The more elastic demand at Location 2 produces a more aggressive price reduction at low inventory levels relative to Location 1.

an agent capable of setting prices above the fixed \$10 baseline when the demand from customers supports it. The optimal pricing policies for each location, shown in Figure 7, reveal that the agent selects higher prices in low-inventory states, where there is more to gain from a single car, and increases prices in high-inventory states to maximize the number of rentals, therefore increasing total profits.

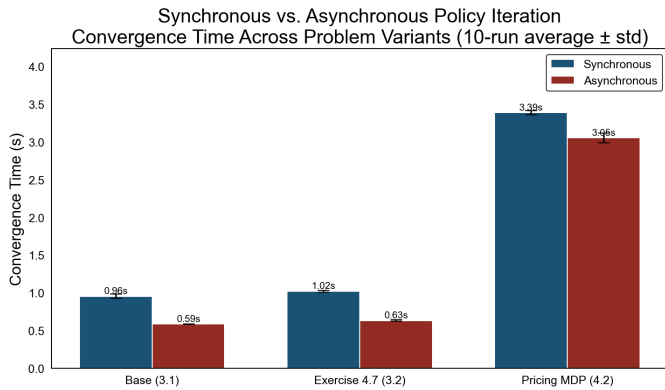


Fig. 8. Mean convergence time of synchronous and asynchronous policy iteration across all three problem formulations: the base problem (Section 3.1), the Exercise 4.7 cost structure (Section 3.2), and the pricing MDP (Section 4.2). Results are averaged over 10 independent runs; error bars indicate one standard deviation. Asynchronous evaluation converges faster than synchronous evaluation across all formulations, with the advantage most pronounced in the computationally demanding pricing MDP.

This behavior is more pronounced at Location 1, where the lower elasticity ($\epsilon = 1.2$) produces a steeper demand response to price increases, causing the agent to reduce prices more aggressively than at Location 1 ($\epsilon = 1.2$) under equivalent inventory conditions. The convergence times across all three problem formulations, shown in Figure 8, confirm that the pricing MDP is substantially more computationally demanding than either the base problem or the Exercise 4.7 extension, requiring over four seconds under synchronous evaluation compared to under two seconds for the simpler formulations. Asynchronous evaluation reduces convergence time across all three formulations, with the absolute time savings largest in the pricing MDP, consistent with the efficiency advantage observed in Section 4.1.

5 CONCLUSION

The experiments presented in this paper demonstrate the adaptability and scalability limitations of dynamic programming as applied to Jack’s Car Rental problem. The base formulation and Exercise 4.7 extension confirm that policy iteration reliably converges to interpretable optimal policies, with non-linearities in the cost structure producing measurable shifts in agent behavior. The irregular policy boundary observed under the Exercise 4.7 cost structure illustrates a broader principle: when reward functions cannot be expressed as smooth functions of the state, the resulting optimal policies may exhibit complex, non-monotonic structure

that would be difficult to anticipate analytically but is nonetheless captured correctly by dynamic programming.

The scalability analysis reveals that asynchronous policy evaluation offers a consistent efficiency advantage over synchronous evaluation as the state space grows, with the performance gap widening at larger state space sizes. However, neither method remains tractable at scales representative of real-world fleet management operations. The pricing MDP extension demonstrates that augmenting the action space with dynamic pricing decisions substantially increases computational cost while producing policies consistent with economic intuition, with agents selecting lower prices under low-inventory conditions to maximize rental volume and higher prices when inventory is abundant. The asymmetry in demand elasticity between the two locations is reflected directly in the learned pricing policies, confirming that the agent correctly identifies location-specific revenue-maximizing behavior.

Taken together, these results suggest that dynamic programming methods are well-suited to small, well-defined MDPs but require approximation or decomposition strategies to remain viable as problem complexity scales. Future work could investigate the application of approximate dynamic programming or deep reinforcement learning methods to the pricing MDP formulation, where the joint inventory-and-pricing state space may be better handled through function approximation than through exhaustive tabular evaluation.

APPENDIX A

RUNNING THE ATTACHED PROG2.IPYNB FILE

In order to run the experiments outlined in this paper. Open the .ipynb file in a IDE of your choice. Examples of IDEs that can be used are Google Colab, VSCode, or PyCharm.

Once the file is open go through each cell in the notebook and run the cell. Run cells in order for best results. Please be advised that cells that run simulations may require several minutes to complete and generate data.

REFERENCES

- [1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.
- [2] Hertz Global Holdings, “Our Company,” *Hertz Newsroom*. [Online]. Available: <https://newsroom.hertz.com/our-company/>. [Accessed: Feb. 20, 2026].