Repairman Trailer Relocation Optimization Problem

Name: Ratan Chowdary Paruchuri **Student ID:** 20011581

Introduction: The Repairman Trailer Relocation Optimization Problem is a crucial challenge faced by repair and maintenance service providers operating in multiple locations. By optimizing the movement of equipment trailers between job sites, these providers can enhance operational efficiency, reduce costs, and save time. This project aims to develop an optimal strategy for trailer relocation, benefiting repair and maintenance service providers such as HVAC technicians, plumbers, electricians, and general contractors.

The optimization of trailer relocation offers significant advantages for these providers. It leads to reduced fuel consumption, decreased vehicle wear and tear, and improved overall efficiency. By optimizing trailer movement, providers can minimize downtime and increase the number of jobs completed per day, resulting in improved customer service and satisfaction.

Background/Related Work: Previous research has explored the Repairman Trailer Relocation Optimization Problem and similar routing problems using mathematical modeling techniques like Markov Decision Processes, dynamic programming, and linear programming. Optimization software, including MATLAB, has been employed to develop algorithms for trailer relocation optimization in logistics, transportation, and supply chain management industries.

However, existing studies have mainly focused on general routing problems and haven't adequately addressed the unique constraints and requirements of the Repairman Trailer Relocation Optimization Problem. Factors such as the variable cost of moving the trailer between sites and the specific location-based costs of using the trailer need to be considered. Additionally, the decision to relocate the trailer is dependent on the repairman's next job site, requiring dynamic decision-making processes. Hence, there is a need for a tailored solution that accounts for the specific constraints and requirements of the problem.

This project aims to bridge that gap by developing algorithms specifically designed to address the Repairman Trailer Relocation Optimization Problem. These algorithms will provide repair and maintenance service providers with a more effective and efficient solution, optimizing trailer relocation decisions and minimizing operational costs

Problem Statement:

1. Job locations: The locations of the four sites, with site 1 representing the home office and sites 2, 3, and 4 representing remote job sites.

2. Equipment trailer cost: The cost of relocating the equipment trailer between the job sites, which is defined as $d(j, k) = 300$ for $k \neq j$.

3. Trailer usage cost: The cost of using the trailer for each job, which is defined as 100 if the work is at site $k > 1$ and trailer is at site $j \neq k$ with $j > 1$, 50 if $j = k$ and $j > 1$, and 200 if the work is at $k > 1$ and the trailer is at $j = 1$ (home office).

4. Assume the discount factor of 0.95.

5. Transition matrix: The transition matrix between job locations, which provides the probabilities of moving between the different job sites.

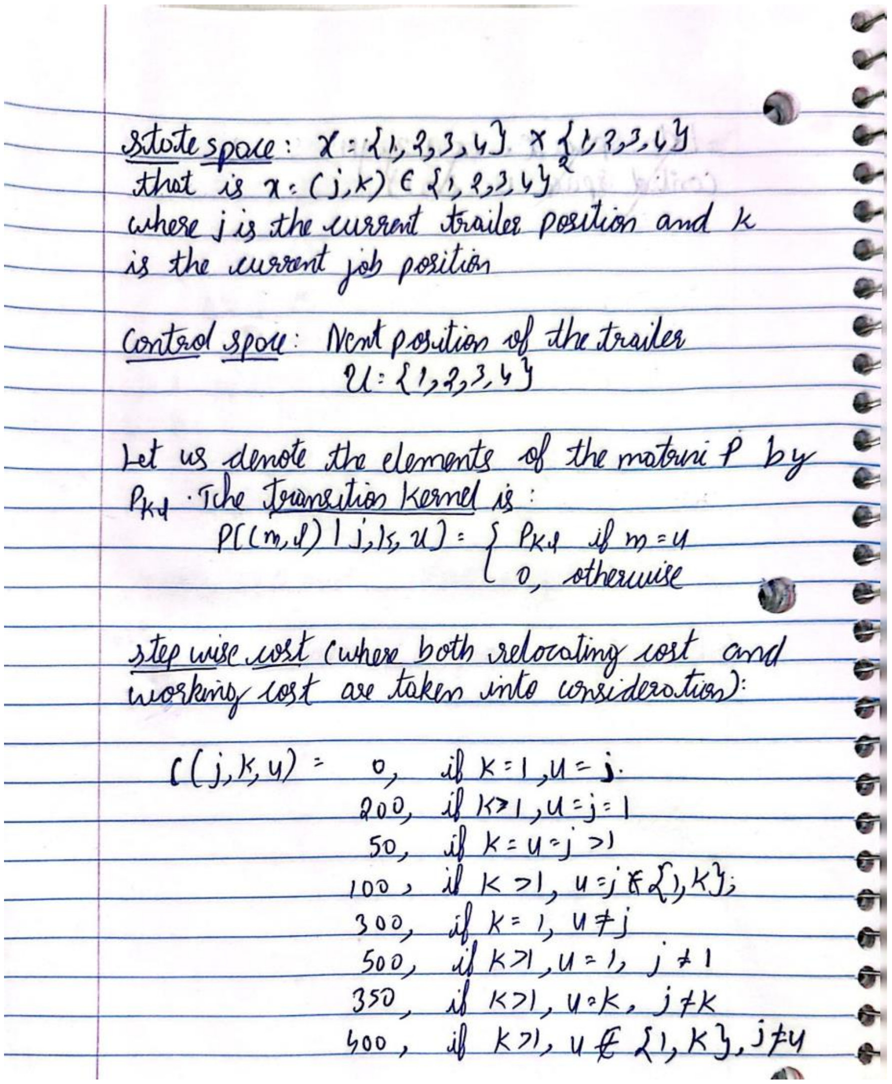
0.1	0.3	0.3	0.3
0.0	0.5	0.5	0.0
0.0	0.0	0.8	0.2
0.4	0.0	0.0	0.6

Given the above data,

- Formulate the dynamic programming equation for optimal value function
- Obtain optimal policy and optimal value function by value iteration and calculate lower and upper bounds
- Obtain optimal policy and optimal value function by policy iteration.
- Obtain optimal policy and optimal value function by linear programming.

Dynamic Programming Equation for Optimal Value Function:

To formulate dynamic programming equations, we require state space, control space, transition probabilities and reward/cost function. We will formulate the equation mathematically,



State space: $\mathcal{X} : \{1,2,3,4\} \times \{1,2,3,4\}$
that is $x : (j,k) \in \{1,2,3,4\}^2$
where j is the current trailer position and k is the current job position

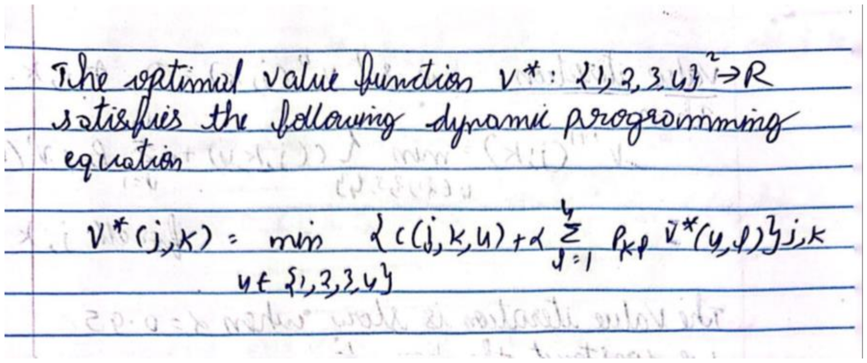
Control space: Next position of the trailer
$\mathcal{U} : \{1,2,3,4\}$

Let us denote the elements of the matrix P by P_{kl}. The Transition Kernel is:

$$P[(m,l) \mid j,k,u] = \begin{cases} P_{kl} & \text{if } m = u \\ 0, & \text{otherwise} \end{cases}$$

step wise cost (where both relocating cost and working cost are taken into consideration):

$$c(j,k,u) = \begin{array}{ll} 0, & \text{if } k = 1, u = j. \\ 200, & \text{if } k > 1, u = j = 1 \\ 50, & \text{if } k = u = j > 1 \\ 100, & \text{if } k > 1, u = j \notin \{1,k\}; \\ 300, & \text{if } k = 1, u \neq j \\ 500, & \text{if } k > 1, u = 1, j \neq 1 \\ 350, & \text{if } k > 1, u = k, j \neq k \\ 400, & \text{if } k > 1, u \notin \{1,k\}, j \neq u \end{array}$$



The optimal value function $V^* : \{1,2,3,4\}^2 \to R$ satisfies the following dynamic programming equation

$$V^*(j,k) = \min_{u \in \{1,2,3,4\}} \{ c(j,k,u) + \alpha \sum_{l=1}^{4} P_{kl} V^*(u,l) \} j,k$$

Value Iteration:

Value Iteration is used to find the optimal value function and policy for Markov Decision Processes (MDPs). It is a dynamic programming algorithm that iteratively improves the value function estimate until convergence.

Steps involved in value iteration:

1. Initialization: Initialize the value function V(s) arbitrarily for all states s in the MDP.
2. Iterative Update: Iterate until the value function converges a. For each state, calculate the value of each possible action a by considering the immediate reward and the expected value of the next state:

   $V(s) \leftarrow \max [R(s, a) + \gamma * \Sigma P(s' \mid s, a) * V(s')]$

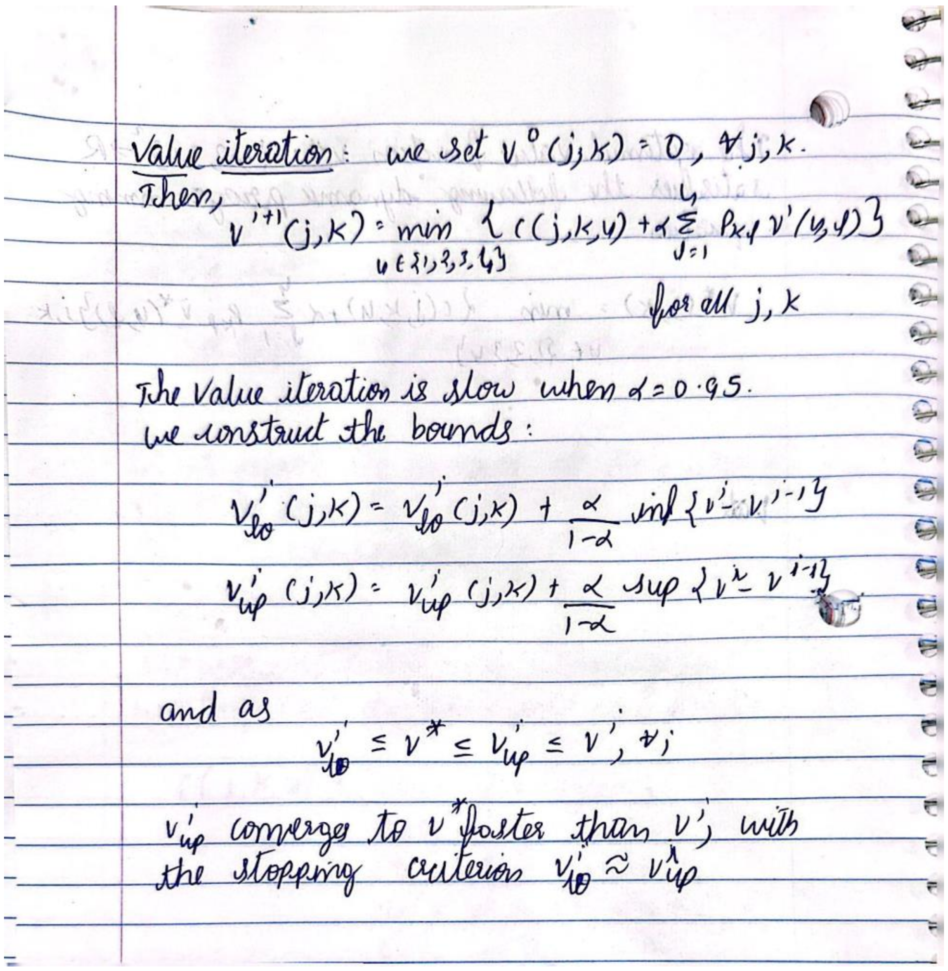
   where R (s, a) is the immediate reward, P (s' | s, a) is the transition probability, V(s') is the value of the next state, and γ is the discount factor.
3. Convergence Check: Check if the maximum change in the value function across all states is below a predefined threshold. If the change is below the threshold, the algorithm has converged.
4. Policy Extraction: Once the value function has converged, the optimal policy can be extracted by choosing the action with the highest expected value for each state:

   $\pi^*(s) \leftarrow \operatorname{argmax} [R(s, a) + \gamma * \Sigma P(s' \mid s, a) * V(s')]$
5. Repeat Steps 2-4 until the policy stabilizes and no further updates are needed.

Upper and lower bounds:

The value iteration is slow when alpha is 0.95. To speed up convergence, we construct lower and upper bounds. The upper and lower bounds maintain a certain monotonicity property during the iteration. Specifically, $v_{lo}(j, k) \leq v^*(j, k) \leq v_{up}(j, k)$, where v* represents the true optimal value function

Let's derive mathematical equations for value iteration, lower and upper bounds



Value iteration: we set $v^0(j,k) = 0, \forall j,k$.
Then,

$$v^{i+1}(j,k) = \min_{u \in \{1,2,3,4\}} \left\{ c(j,k,u) + \alpha \sum_{y=1}^{4} P_{xy} v^i(y,u) \right\}$$

for all j,k

The value iteration is slow when $\alpha = 0.95$.
we construct the bounds:

$$v^i_{lo}(j,k) = v^i_{lo}(j,k) + \frac{\alpha}{1-\alpha} \inf\{v^i - v^{i-1}\}$$

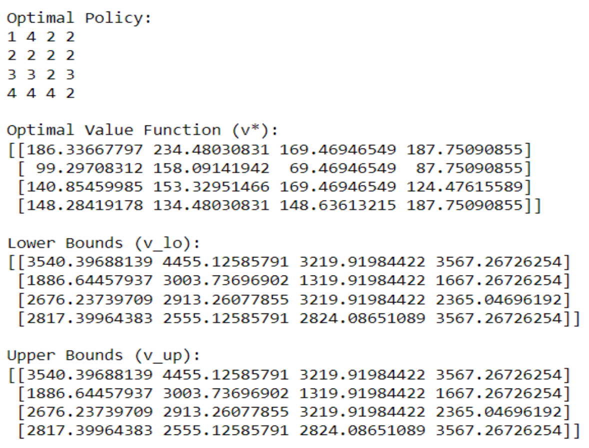
$$v^i_{up}(j,k) = v^i_{up}(j,k) + \frac{\alpha}{1-\alpha} \sup\{v^i - v^{i-1}\}$$

and as

$$v^i_{lo} \le v^* \le v^i_{up} \le v^i, \forall i$$

v^i_{up} converge to v^* faster than v^i, with the stopping criterion $v^i_{lo} \approx v^i_{up}$

Result:



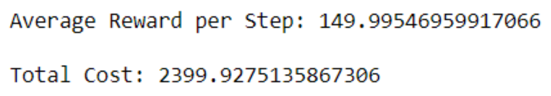
```
Optimal Policy:
1 4 2 2
2 2 2 2
3 3 2 3
4 4 4 2

Optimal Value Function (v*):
[[186.33667797 234.48030831 169.46946549 187.75090855]
 [ 99.29708312 158.09141942  69.46946549  87.75090855]
 [140.85459985 153.32951466 169.46946549 124.47615589]
 [148.28419178 134.48030831 148.63613215 187.75090855]]

Lower Bounds (v_lo):
[[3540.39688139 4455.12585791 3219.91984422 3567.26726254]
 [1886.64457937 3003.73696902 1319.91984422 1667.26726254]
 [2676.23739709 2913.26077855 3219.91984422 2365.04696192]
 [2817.39964383 2555.12585791 2824.08651089 3567.26726254]]

Upper Bounds (v_up):
[[3540.39688139 4455.12585791 3219.91984422 3567.26726254]
 [1886.64457937 3003.73696902 1319.91984422 1667.26726254]
 [2676.23739709 2913.26077855 3219.91984422 2365.04696192]
 [2817.39964383 2555.12585791 2824.08651089 3567.26726254]]
```

We use Average reward per step and Total cost as evaluation metrics for value iteration. Before evaluation, let us talk about it a little bit,

Average Reward per Step: Calculate the average reward earned per step when following the policy. This metric provides insights into the effectiveness of the policy in maximizing rewards.

Total Cost: Calculate the total cost of following the policy over a sequence of states. This can help evaluate the efficiency of the policy in terms of minimizing costs.

Results of the evaluation metrics:



```
Average Reward per Step: 149.99546959917066

Total Cost: 2399.9275135867306
```

Policy Iteration:

Policy iteration is an iterative algorithm used to find an optimal policy in a Markov decision process (MDP). It involves alternating between policy evaluation and policy improvement steps to converge on the optimal policy.

Steps involved in Policy Iteration:

1. Initialization: Start with an arbitrary initial policy π^0
2. Policy Evaluation: Evaluate the state-value function V_π^k for the current policy π_k. Iterate until the value function converges:

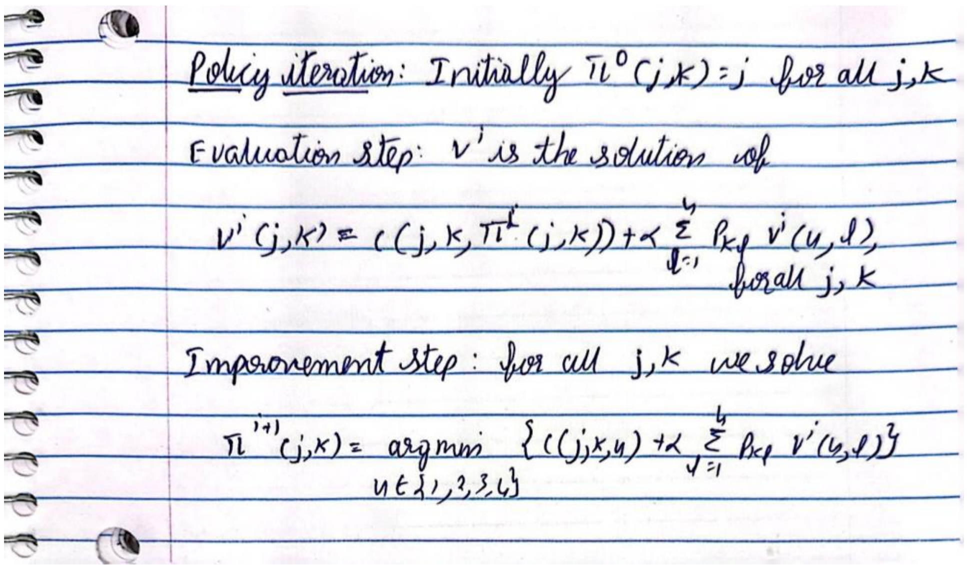
   $V_\pi^{k+1}(s) = \Sigma\, P(s, a, s') * [R(s, a, s') + \gamma * V_\pi^k(s')]$

3. Policy Improvement: Update the policy π_k to be a greedy policy with respect to V_π_k:

   $\pi^{k+1}(s) = \text{argmax}_a\, \Sigma\, P(s, a, s') * [R(s, a, s') + \gamma * V_\pi^k(s')]$

4. Convergence Check: Check if the policy π_k+1 is the same as π_k. If the policies are equal, the algorithm has converged, and the optimal policy π* has been found. Otherwise, go back to step 2 and repeat the process.

Let's derive mathematical equations for policy iteration,



Policy iteration: Initially $\pi^0(j,k)=j$ for all j,k

Evaluation step: v^i is the solution of

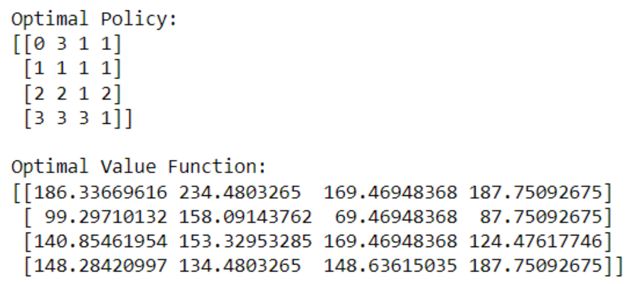
$$v^i(j,k) = c(j,k,\pi^i(j,k)) + \alpha \sum_{l=1}^{4} P_{kl}\, v^i(u,l),$$

for all j,k

Improvement step: for all j,k we solve

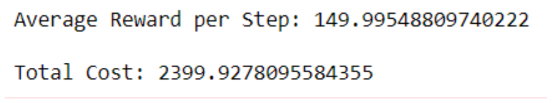
$$\pi^{i+1}(j,k) = \underset{u\in\{1,2,3,4\}}{\arg\min} \left\{ c(j,k,u) + \alpha \sum_{l=1}^{4} P_{kl}\, v^i(u,l) \right\}$$

Result:



```
Optimal Policy:
[[0 3 1 1]
 [1 1 1 1]
 [2 2 1 2]
 [3 3 3 1]]

Optimal Value Function:
[[186.33669616 234.4803265  169.46948368 187.75092675]
 [ 99.29710132 158.09143762  69.46948368  87.75092675]
 [140.85461954 153.32953285 169.46948368 124.47617746]
 [148.28420997 134.4803265  148.63615035 187.75092675]]
```

We use Average reward per step and Total cost as evaluation metrics for policy iteration. Below are the results,



```
Average Reward per Step: 149.99548809740222

Total Cost: 2399.9278095584355
```

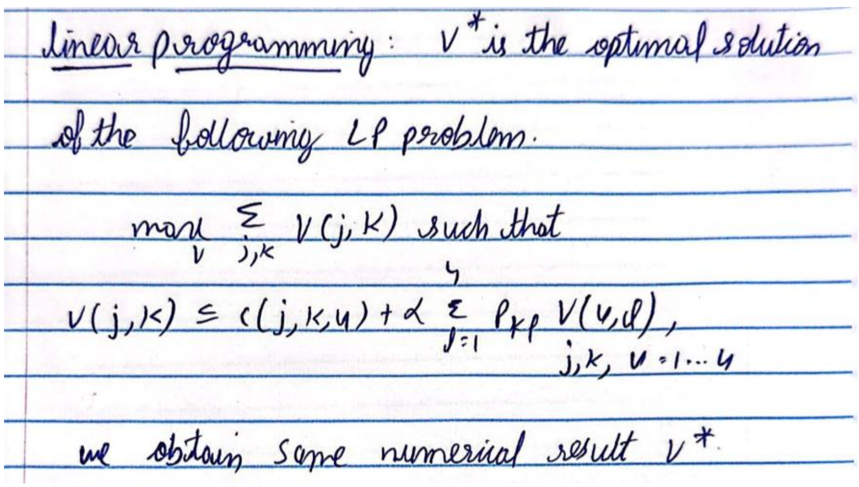
Linear Programming:

Linear programming is used to solve optimization problems where both the objective function and constraints are linear. It is particularly useful when there are multiple constraints and a linear objective function, as it allows finding the optimal solution that maximizes or minimizes the objective while satisfying all the constraints.

Steps involved in Linear programing:

1. Problem Formulation: Define decision variables, write the objective function to be maximized or minimized, and express constraints as a system of linear inequalities or equalities.
2. Convert to Standard Form: Modify the problem to ensure minimization objective, convert inequalities to ≤ form, and introduce slack or artificial variables if needed.
3. Write the Linear Program: Represent the problem in matrix form with coefficient vectors and matrices.
4. Construct the Initial Simplex Tableau: Set up the initial tableau with the objective function row, constraint rows, and right-hand side vector. Include slack or artificial variable columns if necessary.
5. Perform the Simplex Method: Select pivot columns and rows, apply row operations to make the pivot element equal to 1 and zeros in the pivot column, and update the tableau. Repeat until an optimal solution is reached.

Let's derive mathematical equations for Linear programming,



<u>Linear programming</u>: V^* is the optimal solution of the following LP problem.

$$\max_{V} \sum_{j,K} V(j,K) \text{ such that}$$

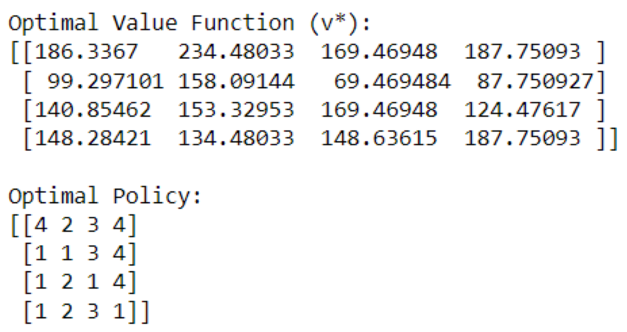
$$V(j,K) \le c(j,K,u) + \alpha \sum_{J=1}^{4} P_{KP} V(u,l), \quad j,K,\ u = 1 \dots 4$$

we obtain some numerical result V^*.

Pulp Module:

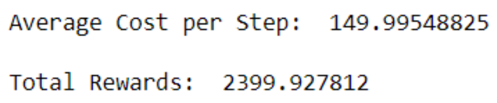
To solve the problem programmatically using linear programming, I have used Pulp module. The Pulp module is a popular open-source linear programming (LP) modeling package in Python. With Pulp, we can easily formulate LP models, specify objective functions and constraints, and solve them using various LP solvers.

Result:



```
Optimal Value Function (v*):
[[186.3367   234.48033  169.46948  187.75093 ]
 [ 99.297101 158.09144   69.469484  87.750927]
 [140.85462  153.32953  169.46948  124.47617 ]
 [148.28421  134.48033  148.63615  187.75093 ]]

Optimal Policy:
[[4 2 3 4]
 [1 1 3 4]
 [1 2 1 4]
 [1 2 3 1]]
```

We use Average reward per step and Total cost as evaluation metrics for linear programming. Below are the results,



```
Average Cost per Step:  149.99548825

Total Rewards:  2399.927812
```

Conclusion:

In conclusion, the Repairman Trailer Relocation Optimization Problem was successfully addressed using three different methods: value iteration, policy iteration, and linear programming. The obtained optimal policies and value functions showcase the effectiveness of these approaches in optimizing trailer relocation decisions and minimizing operational costs. The average cost per step of approximately 149.995 and the total rewards/cost of around 2399.928 demonstrate the efficiency and savings achieved through the optimized trailer relocation strategy. These results provide valuable insights for repair and maintenance service providers, empowering them to make informed decisions and operate more efficiently while delivering superior service to their customers.