```
import numpy as np

NUM_STATES = 22
MAX_CARS = 11
RENTAL_COST = 3
EMPLOYEE_TRANSFER = 1
EXTRA_PARKING_COST = 5
DISCOUNT_FACTOR = 0.85


def reward(state, action):
  cars_moved = min(action, state)
  cars_left = state - cars_moved
  cars_right = action - cars_moved

  reward = 0

  if cars_moved > 0:
    reward -= RENTAL_COST * EMPLOYEE_TRANSFER

  reward += RENTAL_COST * (cars_moved - EMPLOYEE_TRANSFER)

  if cars_left > MAX_CARS:
    reward -= EXTRA_PARKING_COST
  if cars_right > MAX_CARS:
    reward -= EXTRA_PARKING_COST

  return reward


def transition_probability(state, action):
  epsilon = 1e-9
  num_actions = min(state, NUM_STATES - state)
  if num_actions == 0:
    num_actions = 1
  probability = 1 / (num_actions + epsilon)
  P = np.zeros((NUM_STATES, NUM_STATES))
  for next_state in range(NUM_STATES):
    if abs(next_state - state) <= action:
      P[state, next_state] = probability
  return P

def policy_evaluation(policy, value_function, iterations=100, threshold=1e-8):
  for _ in range(iterations):
    new_value_function = np.zeros_like(value_function)
    for state in range(NUM_STATES):
      action = policy[state]
      transition_probs = transition_probability(state, action)
      rewards = reward(state, action)
      new_value_function[state] = np.sum(transition_probs * (rewards + DISCOUNT_FACTOR * value_function))
    change = np.abs(new_value_function - value_function).max()
    value_function = new_value_function
    if change < threshold:
      break
  return value_function

def policy_improvement(value_function):
  policy = np.zeros(NUM_STATES, dtype=int)
  for state in range(NUM_STATES):
    best_action = 0
    best_value = -float('inf')
    for action in range(min(state, NUM_STATES - state) + 1):
      transition_probs = transition_probability(state, action)
      rewards = reward(state, action)
      expected_value = np.sum(transition_probs * (rewards + DISCOUNT_FACTOR * value_function))
      if expected_value > best_value:
        best_value = expected_value
        best_action = action
    policy[state] = best_action
  return policy


def policy_iteration():
  policy = np.ones(NUM_STATES, dtype=int) * EMPLOYEE_TRANSFER
  value_function = np.zeros(NUM_STATES)
  while True:
    new_value_function = policy_evaluation(policy, value_function)
    new_policy = policy_improvement(new_value_function)
    if np.array_equal(policy, new_policy):
      return new_policy, new_value_function
    policy = new_policy
```

```
    value_function = new_value_function

optimal_policy, optimal_value_function = policy_iteration()

print("Optimal Policy:")
for state, action in enumerate(optimal_policy):
  print(f"State {state} :  Move {action} cars")

print("\nOptimal Value Function:")
print(optimal_value_function)
```

```
Optimal Policy:
State 0 :  Move 0 cars
State 1 :  Move 1 cars
State 2 :  Move 2 cars
State 3 :  Move 3 cars
State 4 :  Move 4 cars
State 5 :  Move 5 cars
State 6 :  Move 6 cars
State 7 :  Move 7 cars
State 8 :  Move 8 cars
State 9 :  Move 9 cars
State 10 :  Move 10 cars
State 11 :  Move 10 cars
State 12 :  Move 10 cars
State 13 :  Move 9 cars
State 14 :  Move 8 cars
State 15 :  Move 7 cars
State 16 :  Move 6 cars
State 17 :  Move 5 cars
State 18 :  Move 4 cars
State 19 :  Move 3 cars
State 20 :  Move 2 cars
State 21 :  Move 1 cars

Optimal Value Function:
[-1.99999999e+01  7.21632608e+74  7.20594241e+74  7.19479984e+74
  7.18596385e+74  7.18195861e+74  7.15275016e+74  7.15402283e+74
  7.15458708e+74  7.15842558e+74  7.16313855e+74  6.84019513e+74
  7.16313856e+74  7.15842560e+74  7.15458711e+74  7.15402287e+74
  7.15275021e+74  7.18195868e+74  7.18596393e+74  7.19479993e+74
  7.20594252e+74  7.21632619e+74]
```