HW3

Simulated Annealing for the Travelling Salesman Problem

陳凱騫

2024-12-17

Table of contents

Abstract

This paper applies the Simulated Annealing (SA) algorithm to the **Travelling Salesman Problem (TSP)**. The goal is to find the optimal route that maximizes rewards between cities. The algorithm's effectiveness is evaluated through multiple runs, analyzing convergence, best rewards, and variability across runs.

1. Introduction

The **Travelling Salesman Problem (TSP)** is a classic combinatorial optimization problem. In this version, the objective is to maximize a predefined **reward** between cities. Simulated Annealing (SA) is a probabilistic algorithm that uses a cooling schedule to balance exploration and exploitation.

We:

1. Implement the SA algorithm for a TSP with randomly generated rewards.
2. Analyze the results, including convergence of rewards and reward variability across multiple runs.

3. Visualize the optimal route.

2. Methodology

2.1 Problem Setup

We define:

- $U_{0,k}$: Rewards from the starting $city_0$ to $city_k$.
- $U_{i,j}$: Rewards between $cities_i$ and $city_j$ (for$i \neq j$).
- The objective is to find a route that **maximizes the total reward**.

2.2 Simulated Annealing Algorithm

The SA algorithm performs the following:

1. **Initialize**: Start with a random route and an initial temperature T.
2. **Iterate**:
    - Generate neighboring solutions by swapping two cities.
    - Accept or reject the new solution based on the reward difference and temperature.
3. **Cooling**: Gradually reduce the temperature$T$ using a cooling rate α.

2.3 R Implementation

The following R code implements the SA algorithm:

```r
# Generate Random Rewards
n_cities <- 10
U_ij <- matrix(runif(n_cities * n_cities, 1, 10), nrow = n_cities, ncol = n_cities)
diag(U_ij) <- 0 # No reward for staying in the same city
U_0k <- runif(n_cities, 1, 10) # Rewards from city 0 to other cities

# Function to Calculate Total Reward
calculate_reward <- function(route, U_ij, U_0k) {
  total_reward <- U_0k[route[1]]
  for (i in 1:(length(route) - 1)) {
    total_reward <- total_reward + U_ij[route[i], route[i + 1]]
  }
  return(total_reward)
}

# Simulated Annealing Function
simulated_annealing <- function(U_ij, U_0k, max_iter = 5000, T_init = 100, cooling_rate = 0.99) {
  T <- T_init
  current_route <- sample(1:n_cities)
  current_reward <- calculate_reward(current_route, U_ij, U_0k)
  best_route <- current_route
  best_reward <- current_reward
  reward_progress <- numeric(max_iter)

  for (iter in 1:max_iter) {
```

```
  new_route <- current_route
  swap_idx <- sample(1:n_cities, 2)
  new_route[swap_idx] <- new_route[rev(swap_idx)]
  new_reward <- calculate_reward(new_route, U_ij, U_0k)

  if (new_reward > current_reward || runif(1) < exp((new_reward - current_reward) / T)) {
   current_route <- new_route
   current_reward <- new_reward
   if (new_reward > best_reward) {
    best_reward <- new_reward
    best_route <- new_route
   }
  }
  reward_progress[iter] <- best_reward
  T <- T * cooling_rate
 }

 return(list(best_route = best_route, best_reward = best_reward, reward_progress = reward_progress))
}
```

3. Results and Analysis

3.1 Compare Rewards Across Multiple Runs

To analyze the variability of rewards, the SA algorithm is executed **50 times**, and the best rewards are compared across runs.

```
# Run Simulated Annealing Multiple Times
num_runs <- 50
results <- numeric(num_runs) # Store rewards
routes <- vector("list", num_runs) # Store routes

for (i in 1:num_runs) {
 result <- simulated_annealing(U_ij, U_0k)
 results[i] <- result$best_reward
 routes[[i]] <- result$best_route
}

best_run_index <- which.max(results)   # Index of the maximum reward
best_route <- routes[[best_run_index]]  # Retrieve the route for the max reward
# Print Summary of Results
# Print Results

cat("Best Route Corresponding to Maximum Reward:", best_route, "\n")
```

Best Route Corresponding to Maximum Reward: 7 10 5 1 2 4 3 8 9 6

```
cat("Best Reward Across Runs:", max(results), "\n")
```

Best Reward Across Runs: 89.62038

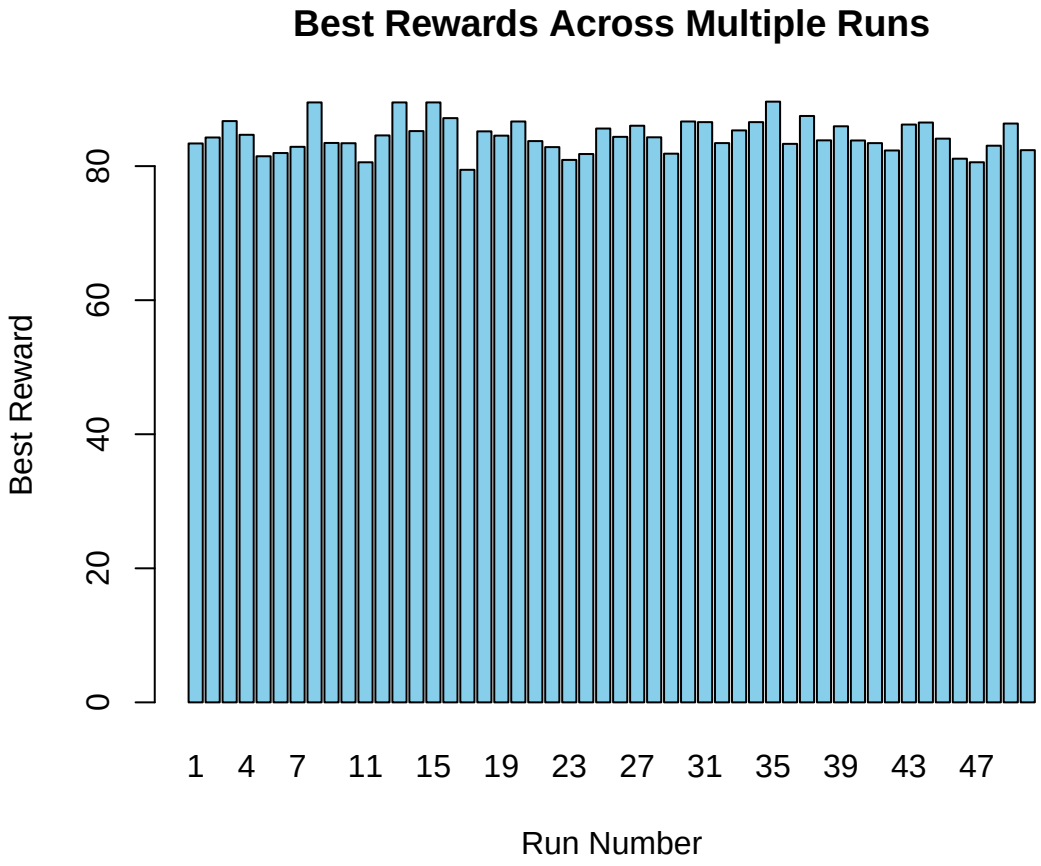
```
cat("Average Reward Across Runs:", mean(results), "\n")
```

Average Reward Across Runs: 84.48678

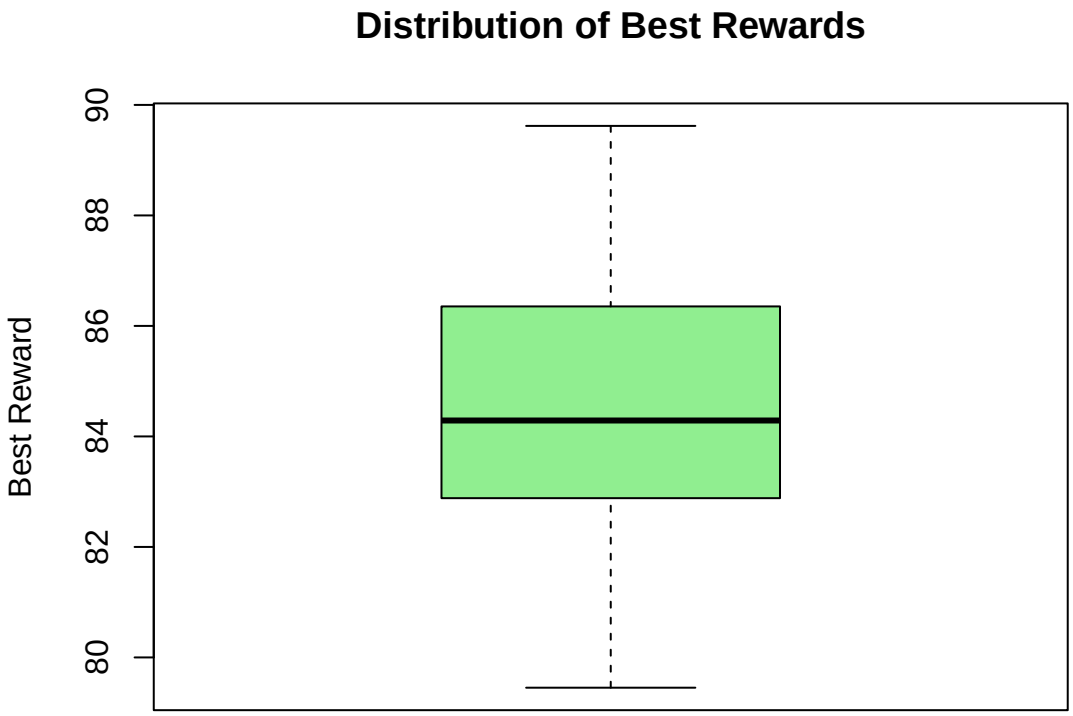
Visualization: Reward Comparison

The bar plot and boxplot below illustrate the distribution and variability of rewards across multiple runs.

```
# Bar Plot of Rewards
barplot(results, col = "skyblue", names.arg = 1:num_runs,
        xlab = "Run Number", ylab = "Best Reward",
        main = "Best Rewards Across Multiple Runs")
```



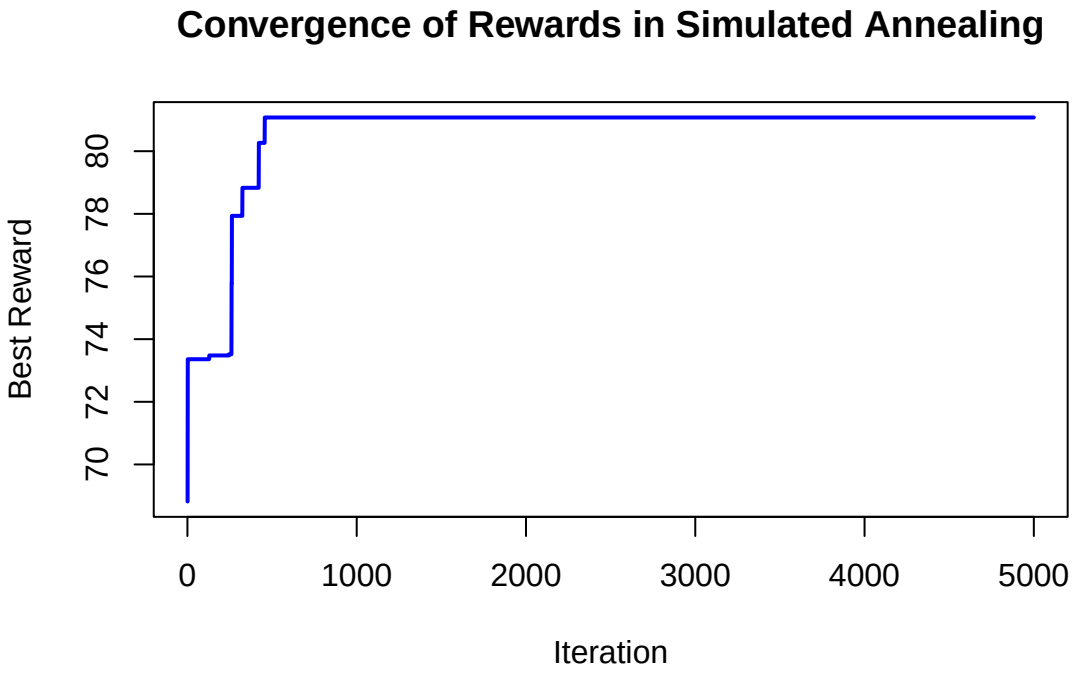

```
# Boxplot of Rewards
boxplot(results, col = "lightgreen",
        ylab = "Best Reward",
        main = "Distribution of Best Rewards")
```



3.2 Convergence of Rewards

The following plot shows the convergence of the reward during a single run of SA.

```
result <- simulated_annealing(U_ij, U_0k)
plot(result$reward_progress, type = "l", col = "blue", lwd = 2,
    xlab = "Iteration", ylab = "Best Reward",
    main = "Convergence of Rewards in Simulated Annealing")
```



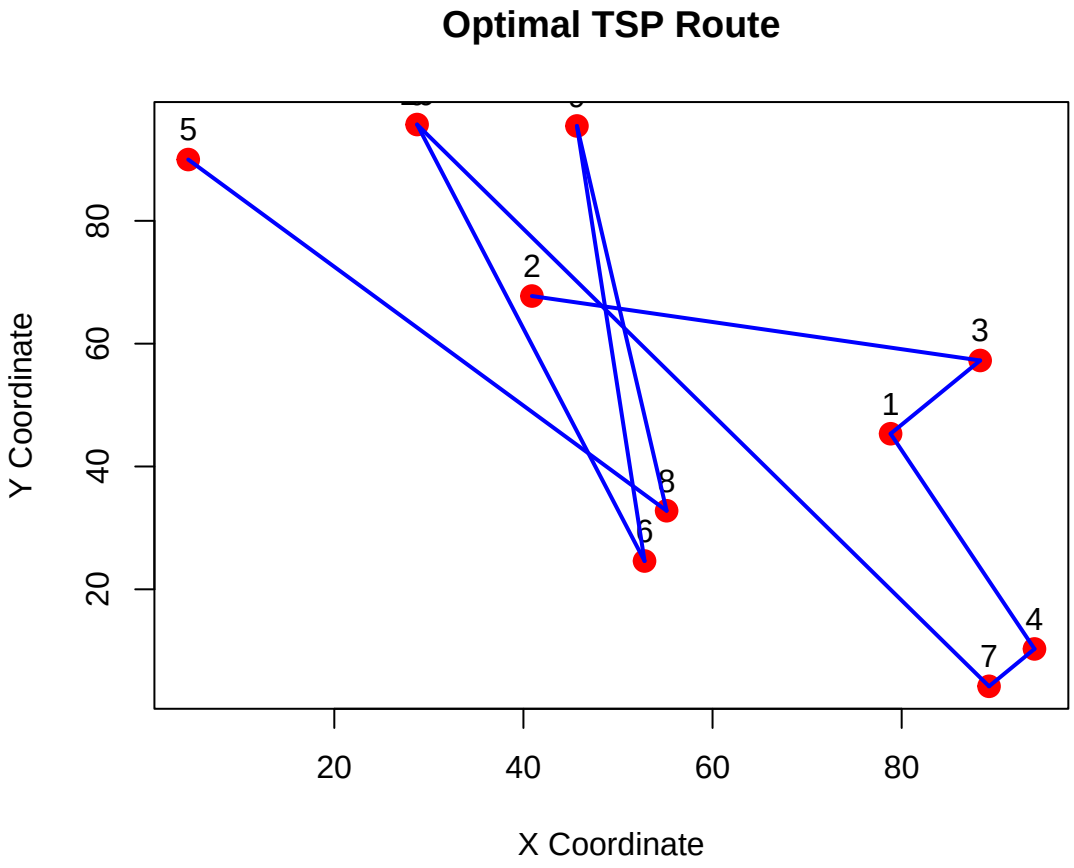
3.3 Visualization of the Optimal Route

The optimal route identified during a single SA run is plotted below.

```
set.seed(123)
city_coords <- data.frame(x = runif(n_cities, 0, 100), y = runif(n_cities, 0, 100))

plot_route <- function(route, coords) {
  complete_route <- c(0, route, 0)
  plot(coords$x, coords$y, pch = 19, col = "red", cex = 1.5,
       xlab = "X Coordinate", ylab = "Y Coordinate", main = "Optimal TSP Route")
  text(coords$x, coords$y, labels = 0:n_cities, pos = 3)
  for (i in 1:(length(complete_route) - 1)) {
    segments(coords$x[complete_route[i] + 1], coords$y[complete_route[i] + 1],
             coords$x[complete_route[i + 1] + 1], coords$y[complete_route[i + 1] + 1],
             col = "blue", lwd = 2)
  }
}

plot_route(result$best_route, city_coords)
```



4. Conclusion

The Simulated Annealing algorithm effectively solved the Traveling Salesman Problem by maximizing rewards. The results show:

1. **Convergence**: The SA algorithm consistently converged to high rewards.

2. **Variability**: Across 50 runs, rewards varied slightly, but most results were close to the optimal.
3. **Optimal Route**: The visualized route confirms the algorithm's ability to find an efficient path.

Future work could include comparisons with other metaheuristic algorithms (e.g., Genetic Algorithm or Tabu Search) and scaling the approach for larger datasets.

References

1. Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science.
2. Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2006). The Traveling Salesman Problem: A Computational Study. Princeton University Press.

What's Included:

1. **Multiple Runs**:
   - Rewards across 50 runs are compared using **bar plots** and **boxplots**.
2. **Convergence Analysis**:
   - A plot showing the improvement of rewards over iterations.
3. **Optimal Route**:
   - A graphical representation of the best route.