CLOUD COMPUTING

A REPORT ON THE PROJECT ENTITLED

Energy-SLA-aware genetic algorithm for edge–cloud integrated computation offloading in vehicular networks

SANDEEP KUMAR - 221CS149

AARUSH KASHYAP - 221CS201

RAHUL BHIMAKARI -221CS143

SHASHANK PRABHAKAR - 221CS246

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL

TABLE OF CONTENT

Understanding

CORE CONCEPT

Vehicular ad hoc Networks (VANETs)

- **Transforming mobility** by enabling vehicle-to-everything (V2X) communications, including vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), vehicle-to-roadside (V2R), vehicle-to-sensor (V2S), and vehicle-to-pedestrian (V2P) connections. This interconnected ecosystem is crucial for supporting various applications such as traffic congestion management, road safety, autonomous driving, infotainment, and emergency services, which demand high responsiveness and precise data transmission to improve road user experience and safety.

- The **computational demands of VANET** applications are high due to the need for real-time data processing and low-latency communications, yet vehicles typically have limited onboard resources. These constraints necessitate innovative approaches to handle intensive data processing without compromising response times, as delays could lead to compromised safety and reduced efficiency in critical applications.

- **Edge computing** and **Cloud computing** integration have emerged as a solution to offload resource-heavy tasks from vehicles to nearby edge servers and more powerful cloud infrastructure, thus meeting the Quality of Service (QoS) demands of critical applications. This integration allows for localized processing at the edge and broader computational support from the cloud, facilitating scalable and efficient processing solutions for large-scale VANET implementations.

- The focus of this research is on developing a novel offloading algorithm that not only reduces **energy consumption** but also meets stringent Service Level Agreement (SLA) constraints, such as **latency** and **processing time**. Such an algorithm is essential to maintain optimal performance and reliability in vehicular networks, which are often characterized by highly dynamic conditions and fluctuating workloads.

- In **Vehicular Ad Hoc Networks** (VANETs), computation offloading is a critical concept aimed at improving the processing capabilities and efficiency of connected vehicles. In a VANET, vehicles generate and share a vast amount of data in real-time, often for applications requiring high processing power, such as collision detection, traffic monitoring, and route optimization. However, individual vehicles may have limited processing resources and cannot efficiently handle complex computational tasks on their own. Offloading addresses this by leveraging external resources, typically through edge computing and cloud computing.

Key Concepts of Offloading in VANETs:

- **Edge Offloading:** Edge servers are strategically placed close to vehicles, often at roadside units (RSUs) or within local cellular infrastructure. This proximity allows tasks to be processed with minimal latency, making edge offloading highly suitable for time-sensitive applications that require quick response times, such as emergency braking alerts or traffic light optimization. Edge servers can quickly process and return data to vehicles, enhancing real-time decision-making while reducing the load on vehicles' onboard systems.

- **Cloud Offloading:** Cloud servers, located further away from vehicles, provide a more centralized and robust computing resource, with high processing power and storage capacity. Although cloud offloading introduces higher latency due to distance, it is ideal for tasks that require intensive computation but are less time-sensitive. These could include analyzing historical traffic patterns, route optimization over a broader region, or aggregating data for fleet management. Cloud offloading thus enables vehicles to access advanced services that are beyond the capacity of edge servers alone.

- **Hybrid Offloading Strategy:** In many VANET setups, a hybrid approach is utilized, combining both edge and cloud resources to maximize efficiency. Tasks are allocated based on their processing requirements and time sensitivity—time-critical tasks are offloaded to nearby edge servers, while more resource-intensive but less urgent tasks are processed in the cloud. This approach balances latency and processing power, optimizing resource utilization across the network.

Types Of Offloading in VANETs:

1. Full Offloading

Full offloading involves transferring the entire computational task from the local device to a remote server for processing. This approach is used when the local device lacks sufficient resources, such as processing power or memory, to execute the task efficiently. Full offloading maximizes energy savings on the local device and takes full advantage of the server's high processing capabilities, resulting in faster execution. It is particularly useful for complex applications like data-intensive simulations or large-scale machine learning training. However, a significant drawback of full offloading is its reliance on a stable and high-speed network connection. In the absence of reliable connectivity, the system may experience latency, which can impact the overall performance and user experience.

2. Partial Offloading

Partial offloading splits a task into smaller components or subtasks, which can then be processed both locally and on a remote server simultaneously. This method is particularly effective for larger, complex tasks that consist of multiple parallelizable subtasks. For example, in data processing applications, certain subtasks can be executed locally, while others that require more computational power are transferred to a remote server. This approach maximizes resource utilization and allows for better load balancing, optimizing both performance and energy consumption. However, partial offloading requires a sophisticated system for task partitioning and synchronization to ensure that subtasks are efficiently managed and combined. The complexity of dividing tasks and managing interdependencies can be a challenge, especially for applications with highly interconnected processes.

3. Binary Offloading

Binary offloading refers to an approach where a task is either executed entirely on the local device or offloaded entirely to a remote server, without any intermediate splitting. The decision on where the task is processed depends on current conditions, such as device resource availability and network quality. Binary offloading provides a straightforward method to optimize performance and power consumption by leveraging the strengths of both the local and remote environments. This type of offloading simplifies decision-making, as it only requires evaluating the conditions to choose between local execution and remote execution. While binary offloading can enhance performance when used effectively, its all-or-nothing nature means that it may not provide the same level of optimization as more granular approaches like partial offloading

Benefits of Offloading in VANETs:

1. Reduced Latency

Edge offloading ensures that data processing occurs closer to the source, resulting in significantly lower response times. This is crucial for real-time safety applications such as collision avoidance systems, lane-keeping assistance, and autonomous emergency braking, where even milliseconds can make a difference in preventing accidents. Additionally, reduced latency improves the experience in non-safety applications, such as augmented reality navigation or infotainment systems, enhancing user satisfaction and operational efficiency.

2. Enhanced Resource Utilization

Offloading computations across edge and cloud servers helps distribute the workload efficiently, which optimizes the usage of available resources. This prevents the central cloud server from becoming a bottleneck and ensures that edge servers handle local data processing tasks while more resource-intensive computations are sent to the cloud. This division of labor reduces the risk of server overload, minimizes delays, and helps maintain balanced system performance even during peak usage times.

3. Energy Efficiency

Outsourcing computationally heavy tasks to edge or cloud servers reduces the power demand on vehicle onboard systems, leading to extended battery life or reduced fuel consumption. This is particularly important for electric vehicles, where efficient energy use translates to a longer driving range. By minimizing the processing load on the vehicle's CPU, thermal management is improved, reducing the risk of overheating and wear on internal components. Energy efficiency also contributes to the overall sustainability of connected vehicle networks.

4. Improved Scalability

The combination of edge and cloud computing resources supports seamless scaling as the number of connected vehicles in a network increases. As more vehicles join the network and transmit data, edge servers handle localized processing, while the cloud manages broader, data-intensive tasks. This distributed approach ensures that the system can scale up without significant modifications, accommodating growing demand in urban and highway settings without compromising performance or service quality.

5. SLA Compliance

Meeting Service Level Agreements (SLAs) for critical vehicular applications is essential for maintaining trust and reliability in connected vehicle networks. With a well-implemented offloading strategy, VANETs can prioritize and fulfill SLAs by dynamically allocating tasks to the appropriate resources, ensuring that critical applications such as emergency alerts, real-time diagnostics, and route optimization adhere to their required response times and performance metrics. This compliance helps maintain service reliability, essential for public safety and user satisfaction.

6. Improved Data Processing Capabilities

Offloading complex data processing tasks to the cloud or edge servers empowers vehicles to run sophisticated applications that would otherwise be limited by onboard computational capacity. Examples include AI-driven predictive analytics for traffic flow management, detailed video analysis for road condition assessment, and advanced driver assistance systems (ADAS) that rely on deep learning algorithms. This capability enables vehicles to make smarter, data-informed decisions, enhancing both safety and the driving experience.

PROBLEM TO ADDRESS

Main Problem:

- The increasing reliance on offloading in **Vehicular Ad Hoc Networks** (VANETs) presents a dual challenge: it escalates energy consumption while heightening the risk of violating **Service Level Agreements (SLAs**). The essential goal of offloading is to minimize the energy required to meet the demands of various computational requests from vehicles, but this objective is complicated by several interrelated issues.

Key Challenges:

- **High Energy Consumption** - Offloading tasks to edge and cloud servers results in substantial energy consumption, leading to higher operational costs and detrimental environmental impacts. Continuous offloading contributes to an increased carbon footprint, necessitating the development of more energy-efficient strategies to mitigate both economic and ecological concerns. The transition to sustainable energy sources in server operations is also a critical consideration for reducing the overall environmental impact.

- **Latency and SLA Violations**- Latency poses a significant challenge, particularly for time-sensitive applications such as accident alerts and real-time traffic updates. Delays can compromise both safety and operational efficiency, while the trade-off between energy efficiency and SLA compliance remains inadequately addressed. Traditional algorithms often either violate SLAs due to excessive delays or consume excessive energy to meet performance demands, underscoring the need for advanced optimization methods that can ensure timely processing without incurring prohibitive energy costs.

- **Limited Resources in Vehicles**- Vehicles are typically constrained in their computational capabilities, making it difficult to process complex tasks locally. This limitation results in inefficient task management and an increased reliance on offloading, which can further exacerbate energy consumption and SLA adherence issues. Developing lightweight algorithms for local processing is essential to alleviate these challenges and improve the efficiency of task management in vehicles. Furthermore, enhancing the onboard processing capabilities can facilitate better real-time decision-making and reduce latency in critical applications.

CHALLENGES

- **Scalability and Real-Time Processing**:
  As vehicular networks expand, supporting a growing number of connected vehicles and infrastructure elements, the need for efficient, real-time processing becomes critical. Applications such as autonomous driving, accident prevention, and augmented reality require rapid data exchange and processing to ensure timely responses, where even slight delays can compromise safety. However, scaling these networks without facing processing bottlenecks is challenging due to the high data throughput and low-latency requirements. Consequently, highly efficient task offloading mechanisms are essential to maintain system responsiveness, optimize resource utilization, and prevent network congestion as VANETs grow in size and complexity. Developing algorithms and frameworks that ensure scalability while upholding stringent Quality of Service (QoS) parameters is, therefore, a significant challenge in this domain.

- **Environmental Sustainability**:
  The expansion of vehicular networks and the integration of edge and cloud computing significantly impact energy consumption, with global data centers projected to account for 4.5% of total energy use by 2025. As these networks and computational infrastructures grow, so does their environmental footprint, raising concerns about the sustainability of large-scale data processing. Addressing energy optimization in VANETs is therefore vital, not only to meet economic goals by reducing operational costs but also to support environmental objectives by minimizing greenhouse gas emissions and resource depletion. Strategies for energy-efficient processing, effective offloading algorithms, and sustainable data management are critical to ensuring that the deployment of VANETs aligns with global sustainability targets. This requires a careful balance between maintaining the high performance demanded by vehicular applications and reducing the environmental impact associated with such intensive computational
  activities.

- **Data Privacy and Security:**
  Ensuring robust data privacy and security in vehicular ad hoc networks (VANETs) poses significant challenges as these networks handle sensitive information such as location data, vehicle diagnostics, and user behavior. The interconnected nature of VANETs, combined with edge and cloud offloading, creates a complex ecosystem where data is transmitted between various nodes, increasing vulnerability to cyber-attacks and unauthorized access. Potential risks include data interception, identity spoofing, and denial-of-service attacks, all of which can have severe consequences for both the vehicle and driver safety. Implementing comprehensive security protocols, such as end-to-end encryption, robust authentication mechanisms, and real-time intrusion detection systems, is essential to safeguard data integrity and confidentiality. However, these security measures often come at the cost of additional computational overhead, which can affect the system's performance and real-time processing capabilities. Therefore, balancing the implementation of stringent security measures with the need for low latency and high-speed data processing is a complex but crucial aspect of VANET deployment. The development of lightweight, scalable security frameworks that can integrate seamlessly with offloading mechanisms while preserving QoS and maintaining user trust is a pressing challenge in the evolution of connected vehicular technology.

High-Level Working Of Our Genetic Algorithm

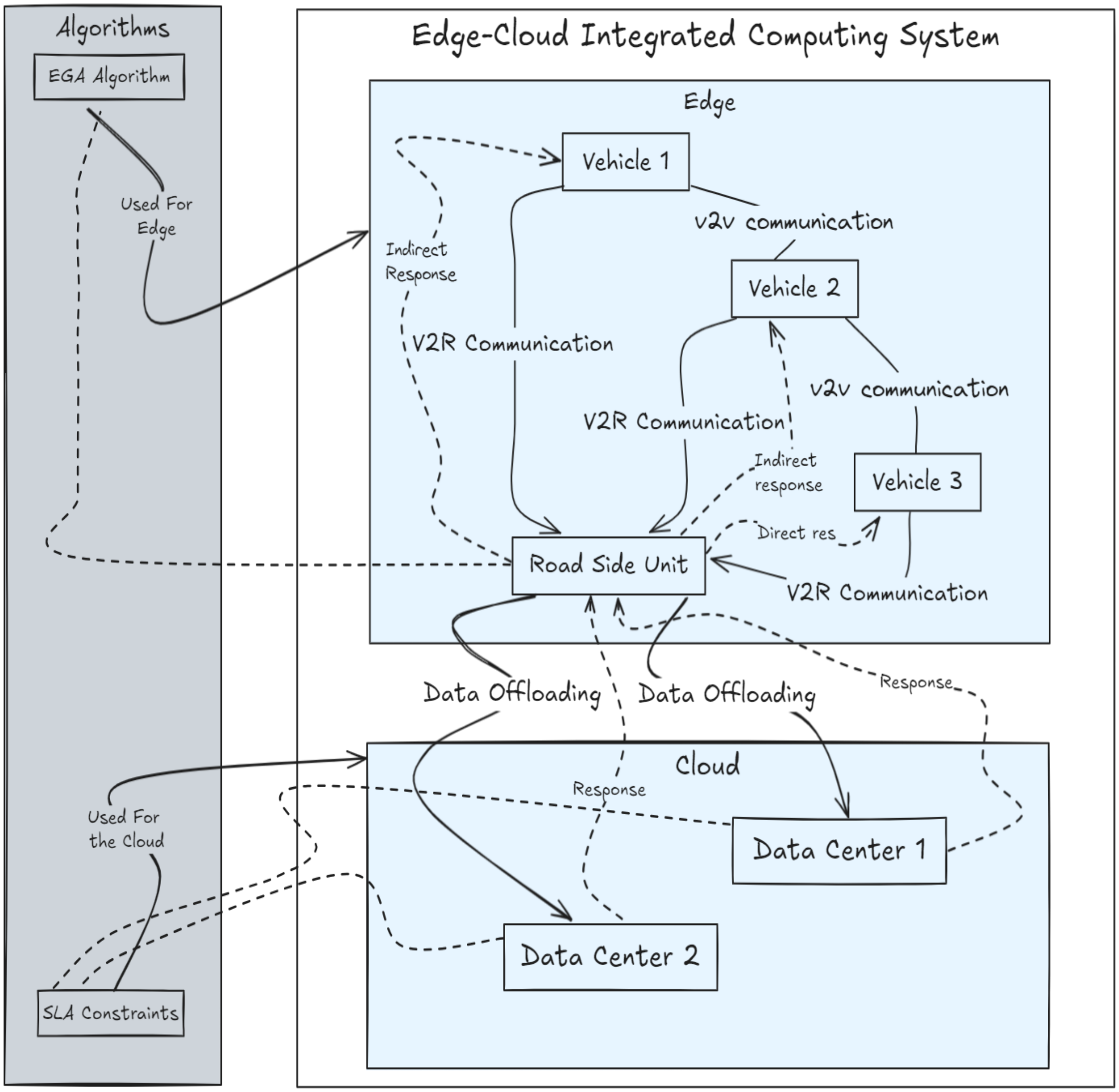


Fig 1. Diagram of computational offloading in VANETs

Workflow

Step-by-Step workflow for the python implementation of EGA:

The workflow for implementing an **Evolutionary Genetic Algorithm (EGA)** involves several key steps to iteratively optimize solutions. It begins with **initializing a population** of potential solutions, encoded as chromosomes. Each chromosome is evaluated for its **fitness**, indicating how well it addresses the target problem.

Next, **selection** occurs, where fitter solutions are chosen as parents to create the next generation. These parents undergo **crossover** to produce offspring and may also experience **mutation** to introduce random changes, enhancing diversity.

This process repeats over multiple generations, with each new generation evaluated for fitness. The algorithm concludes when a solution meets a predefined fitness threshold or after a specified number of generations, resulting in an optimized solution that approximates the best answer to the problem.

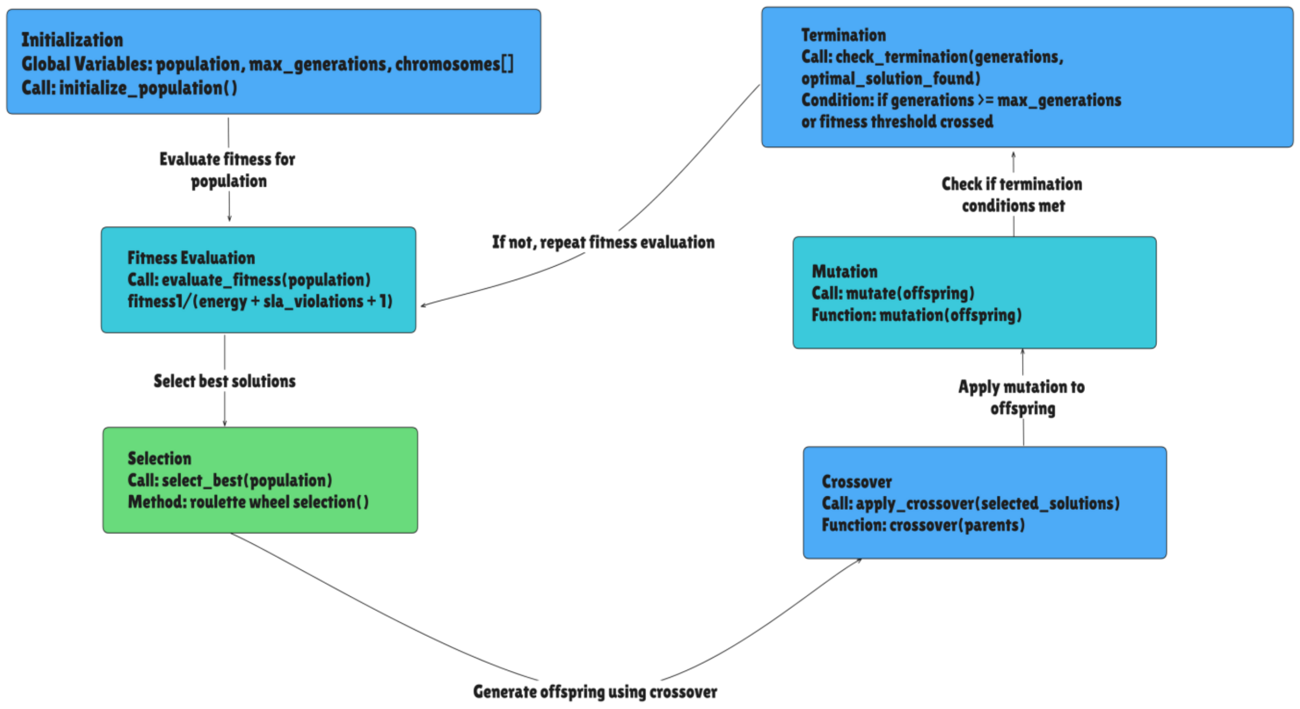

Initialization
Global Variables: population, max_generations, chromosomes[]
Call: initialize_population()
Evaluate fitness for population
Fitness Evaluation
Call: evaluate_fitness(population)
fitness1/(energy + sla_violations + 1)
Select best solutions
Selection
Call: select_best(population)
Method: roulette wheel selection()
Generate offspring using crossover
Crossover
Call: apply_crossover(selected_solutions)
Function: crossover(parents)
Apply mutation to offspring
Mutation
Call: mutate(offspring)
Function: mutation(offspring)
Check if termination conditions met
Termination
Call: check_termination(generations, optimal_solution_found)
Condition: if generations >= max_generations or fitness threshold crossed
If not, repeat fitness evaluation

Fig 2. Flowchart of our algorithm

Overview:

The genetic algorithm iteratively improves task allocation by simulating a natural selection process. In each generation, it evaluates multiple potential solutions, or "chromosomes," based on a fitness function that considers both energy consumption and SLA compliance. Through operations like selection, crossover, and mutation, the algorithm combines and modifies chromosomes to evolve better solutions over time. By balancing task distribution between edge and cloud servers, the algorithm aims to minimize latency and energy costs while ensuring that tasks meet the strict processing time constraints essential for vehicular network applications. This approach is particularly advantageous in dynamic environments, as it adapts to fluctuating network conditions and varying computational demands.

WHAT THE CODE DOES

Initialization:

Generate Parameters: Sets up simulation parameters, including server details and request information.

Offloading Solution Initialization: Creates an initial set of possible solutions that define where each computation request (task) will be processed (either on an edge server or a cloud server).

Genetic Algorithm Steps:

1. **Evaluation:** Each solution is evaluated based on how well it meets the processing time requirements, determining a fitness score for each solution.
2. **Selection:** The best solutions are chosen based on fitness scores to contribute to the next generation.
3. **Crossover:** New solutions are created by combining parts of selected solutions.
4. **Mutation:** Random changes are introduced in solutions to maintain diversity and explore a broader range of potential solutions.

Termination and Final Evaluation:

The algorithm iterates over multiple generations, refining the solutions each time. After the final generation, it evaluates the solutions one last time to pick the best one based on the highest fitness score, outputting it as the final solution.

Code Flow:

Main Function:

1. Sets up the simulation parameters and initializes the Generations folder structure. Performs initial offloading solution setup and enters a loop for each generation.

Generation Loop:

1. For each generation, the code:
2. Evaluates fitness scores.
3. Selects the best solutions for the next generation.
4. Applies crossover to produce new solutions.
5. Applies mutation to maintain diversity.

Output Files:

1. Each generation's data is stored in its own folder (e.g., gen1, gen2, …).
2. The final.txt file at the end of all generations contains the best-performing solution across all generations, based on the fitness evaluation.

How It Works:

The genetic algorithm follows these principles:

Fitness-Based Selection: Solutions that perform better (i.e., have lower energy use and meet SLAs more consistently) are more likely to be retained for the next generation.

Crossover and Mutation:

Helps explore new possible solutions, mimicking natural selection, and evolution.

Termination:

The algorithm stops after a set number of generations, and the best solution found is selected as the final output. In essence, the code refines possible solutions over multiple iterations (generations), leveraging both edge and cloud resources, to find an optimal solution for computation offloading in a vehicular network.

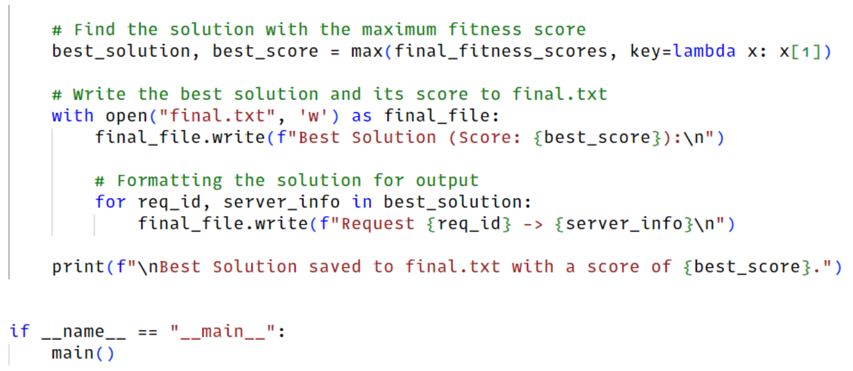
NOTE: WE ONLY USE THE PROCESSING TIME FOR EVALUATING THE SOLUTIONS

CODE

```python
import os
import random
import shutil

# Global variables
popsize = None
num_requests = None
num_edge_servers = None
num_cloud_servers = None
crossover_rate = None
mutation_rate = None
chromosome_length = None
max_num_generations = None
requests = []
edge_servers = []
cloud_servers = []

def cleanup():
    # Remove the Generations folder if it exists
    if os.path.exists("Generations"):
        shutil.rmtree("Generations")

    # Remove parameters.txt if it exists
    if os.path.exists("parameters.txt"):
        os.remove("parameters.txt")

    # Remove final.txt if it exists
    if os.path.exists("final.txt"):
        os.remove("final.txt")


def step_0_generate_parameters():
    """Generates and saves initial parameters to parameters.txt based on specified constraints."""

    global popsize, num_requests, num_edge_servers, num_cloud_servers
    global crossover_rate, mutation_rate, chromosome_length, max_num_generations
    global requests, edge_servers, cloud_servers

    # Define parameter values
    popsize = random.randint(10, 30)  # Small population size for smooth and quick execution
    num_requests = 10  # Fixed low number of requests
    num_edge_servers = random.randint(3, 7)  # Number of edge servers
    num_cloud_servers = random.randint(3, 7)  # Number of cloud servers
    crossover_rate = 0.5  # Fixed crossover rate
    mutation_rate = 0.1   # Fixed mutation rate
    chromosome_length = num_requests  # Each chromosome represents an offloading solution for each request
    max_num_generations = 100  # Maximum number of generations for the algorithm

    # Generate Edge and Cloud Servers with processing speeds
    edge_servers = []
    for i in range(1, num_edge_servers + 1):
        speed = random.randint(100, 500)  # Edge servers have lower processing speeds
        edge_servers.append({"id": i, "processing_speed": speed, "area_id": i})

    cloud_servers = []
    for i in range(1, num_cloud_servers + 1):
        speed = random.randint(600, 1500)  # Cloud servers have higher processing speeds
        cloud_servers.append({"id": i, "processing_speed": speed})

    # Generate Requests with an instruction length and assign area_id
    requests = []
    for i in range(num_requests):
        area_id = random.randint(1, num_edge_servers)  # Request area_id within number of RSUs
        instruction_length = random.randint(500, 5000)  # Request instruction length in MI
        max_processing_time = random.randint(1, 10)  # Max processing time in seconds (assumed 1/10 of instruction length)

        requests.append({
            "id": i + 1,
            "area_id": area_id,
            "instruction_length": instruction_length,
            "max_processing_time": max_processing_time
        })
    # Save all parameters to the parameters.txt file
    with open("parameters.txt", "w") as file:
        file.write(f"popsize: {popsize}\n")
        file.write(f"num_requests: {num_requests}\n")
        file.write(f"num_edge_servers: {num_edge_servers}\n")
        file.write(f"num_cloud_servers: {num_cloud_servers}\n")
        file.write(f"crossover_rate: {crossover_rate}\n")
        file.write(f"mutation_rate: {mutation_rate}\n")
        file.write(f"chromosome_length: {chromosome_length}\n")
        file.write(f"max_num_generations: {max_num_generations}\n\n")

        file.write("Edge Servers:\n")
        for es in edge_servers:
            file.write(f"  Edge Server {es['id']}: Processing Speed = {es['processing_speed']} MIPS, Area ID = {es
            ['area_id']}\n")

        file.write("\nCloud Servers:\n")
        for cs in cloud_servers:
            file.write(f"  Cloud Server {cs['id']}: Processing Speed = {cs['processing_speed']} MIPS\n")

        file.write("\nRequests:\n")
        for req in requests:
            file.write(f"  Request {req['id']}: Instruction Length = {req['instruction_length']} MI, "
                       f"Max Processing Time = {req['max_processing_time']} seconds, Area ID = {req['area_id']}\n")

    print("Parameters generated and saved to parameters.txt")


def step_1_initialization(generation_folder):
    """Initializes a subset of offloading solutions and saves to step1.txt in the specified generation folder."""
    offloading_solutions = []
    for l in range(popsize):  # For each solution in the population
        solution = []
        for req in requests:  # For each request in this solution
            # Decide between edge server and cloud server
            server_type = random.choice(["edge", "cloud"])
            if server_type == "edge":
                # Get edge servers in the area specified by the request
                area_id = req["area_id"]
                available_edge_servers = [es for es in edge_servers if es["area_id"] == area_id]

                # Choose an edge server from the available ones
                if available_edge_servers:
                    server = random.choice(available_edge_servers)
                    server_info = f"Edge Server {server['id']}"
                else:
                    # If no edge server is available, fallback to cloud server
                    server = random.choice(cloud_servers)
                    server_info = f"Cloud Server {server['id']}"
            else:
                server = random.choice(cloud_servers)
                server_info = f"Cloud Server {server['id']}"

            solution.append((req["id"], server_info))

        offloading_solutions.append(solution)

     # Save offloading solutions to step1.txt within the specified generation folder
     with open(os.path.join(generation_folder, "step1.txt"), "w") as file:
         file.write("Initialization of Offloading Solutions:\n")
         for idx, solution in enumerate(offloading_solutions, 1):
             file.write(f"Solution {idx}:\n")
             for req_id, server_info in solution:
                 file.write(f"  Request {req_id} -> {server_info}\n")
             file.write("\n")

     print(f"Step 1: Initialization complete. Saved to {os.path.join(generation_folder, 'step1.txt')}")

     return offloading_solutions  # Return the initialized solutions for evaluation

def evaluate_solutions(offloading_solutions):
    """Evaluates all solutions and calculates their fitness scores."""
    violations_list = []

    # Calculate violations for each solution
    for solution in offloading_solutions:
        total_violation = 0
        for req_id, server_info in solution:
            # Get the request details
            request = next(req for req in requests if req['id'] == int(req_id))

            # Determine the server type and processing speed
            if "Edge Server" in server_info:
                server = next(es for es in edge_servers if es['id'] == int(server_info.split()[-1]))
                processing_speed = server['processing_speed']
            else:
                server = next(cs for cs in cloud_servers if cs['id'] == int(server_info.split()[-1]))
                processing_speed = server['processing_speed']

            # Calculate processing time
            processing_time = request['instruction_length'] / processing_speed

            # Calculate processing time violation
            violation = max(0, processing_time - request['max_processing_time'])
            total_violation += violation

        violations_list.append(total_violation)

    # Find the maximum violation among all solutions
    max_violation = max(violations_list) if violations_list else 0

    # Calculate fitness scores for each solution and return them
    fitness_scores = []
    for total_violation in violations_list:
        if max_violation > 0:
            fitness_score = 1 / (total_violation / max_violation + 1)
        else:
            fitness_score = 1.0  # No violations
        fitness_scores.append(fitness_score)

    return fitness_scores  # Return fitness scores for selection

def step_2_evaluate_solutions(offloading_solutions, gen_folder):
    """Evaluates all solutions and saves the results to step2.txt."""
    fitness_scores = evaluate_solutions(offloading_solutions)

    # Save fitness scores to step2.txt
    with open(os.path.join(gen_folder, "step2.txt"), "w") as file:
        file.write("Fitness Scores:\n")
        for idx, score in enumerate(fitness_scores, 1):
            file.write(f"Solution {idx}: Fitness Score = {score}\n")

    print(f"Step 2: Evaluation complete. Fitness scores saved to {os.path.join(gen_folder, 'step2.txt')}.")
    return fitness_scores  # Return fitness scores
def step_3_selection(offloading_solutions, fitness_scores, gen_folder):
    """Selects fittest solutions using Roulette Wheel Selection (RWS) method."""
    total_fitness = sum(fitness_scores)

    # Calculate fitness probabilities and cumulative probabilities
    fitness_probabilities = [fitness / total_fitness for fitness in fitness_scores]
    cumulative_probabilities = [sum(fitness_probabilities[:i + 1]) for i in range(len(fitness_probabilities))]

    # Generate random numbers for selection
    random_numbers = [random.random() for _ in range(len(offloading_solutions))]

    # Select solutions based on random numbers
    selected_solutions = []
    for rand in random_numbers:
        for idx, cumulative_probability in enumerate(cumulative_probabilities):
            if rand <= cumulative_probability:
                selected_solutions.append((idx + 1, offloading_solutions[idx]))
                break

    # Save selection results to step3.txt within the specified generation folder
    with open(os.path.join(gen_folder, "step3.txt"), "w") as file:
        file.write("Selection Results:\n")
        file.write("Solution  Fitness Probability  Cumulative Probability  Random Number  Selected Solution\n")
        for i in range(len(offloading_solutions)):
            file.write(f"{i + 1:<9}\t{fitness_probabilities[i]:<20.4f}\t{cumulative_probabilities[i]:<25.4f}"
                       f"{random_numbers[i]:<15.4f}\t{selected_solutions[i][0]}\n")

    print(f"Step 3: Selection complete. Results saved to {os.path.join(gen_folder, 'step3.txt')}.")
    return [solution[1] for solution in selected_solutions]  # Return only the selected solutions
```

```python
def step_4_crossover(selected_solutions, gen_folder):
    """Performs single-point crossover on selected solutions and saves the results to step4.txt in the specified
    generation folder."""
    offspring_solutions = selected_solutions.copy()  # Copy of selected solutions for generating offspring

    # Step 1: Select solutions for crossover
    crossover_candidates = []
    random_numbers = [random.random() for _ in range(len(selected_solutions))]
    for i, rand in enumerate(random_numbers):
        if rand < crossover_rate:
            crossover_candidates.append(i)  # Select solution based on random number and crossover rate

    # Step 2: Randomly pair solutions for crossover
    random.shuffle(crossover_candidates)
    pairs = [(crossover_candidates[i], crossover_candidates[i + 1])
             for i in range(0, len(crossover_candidates) - 1, 2)]

    # Step 3: Perform single-point crossover for each pair
    for idx1, idx2 in pairs:
        parent1 = offspring_solutions[idx1]
        parent2 = offspring_solutions[idx2]

        # Generate a random cutoff point for single-point crossover
        cutoff = random.randint(1, chromosome_length - 1)

        # Create offspring by swapping allocations after the cutoff point
        offspring1 = parent1[:cutoff] + parent2[cutoff:]
        offspring2 = parent2[:cutoff] + parent1[cutoff:]

        # Step 4: Calculate fitness scores for parents and offspring
        parent1_violation = calculate_total_violation(parent1)
        parent2_violation = calculate_total_violation(parent2)
        offspring1_violation = calculate_total_violation(offspring1)
        offspring2_violation = calculate_total_violation(offspring2)

        # Maximum violation among all 4 solutions for relative fitness calculation
        max_violation = max(parent1_violation, parent2_violation, offspring1_violation, offspring2_violation)

        # Fitness scores with conditional check for max violation
        parent1_fitness = 1 if max_violation == 0 else 1 / (parent1_violation / max_violation + 1)
        parent2_fitness = 1 if max_violation == 0 else 1 / (parent2_violation / max_violation + 1)
        offspring1_fitness = 1 if max_violation == 0 else 1 / (offspring1_violation / max_violation + 1)
        offspring2_fitness = 1 if max_violation == 0 else 1 / (offspring2_violation / max_violation + 1)

        # Step 5: Replace parents with the two fittest solutions from parents and offspring
        fittest_two = sorted(
            [(parent1, parent1_fitness), (parent2, parent2_fitness),
             (offspring1, offspring1_fitness), (offspring2, offspring2_fitness)],
            key=lambda x: x[1], reverse=True
        )[:2]

        offspring_solutions[idx1], offspring_solutions[idx2] = fittest_two[0][0], fittest_two[1][0]
    # Save results to step4.txt in the specified generation folder
    with open(os.path.join(gen_folder, "step4.txt"), "w") as file:
        file.write("Offloading Solutions after Crossover:\n")
        for idx, solution in enumerate(offspring_solutions, 1):
            file.write(f"Solution {idx}:\n")
            for req_id, server_info in solution:
                # Update to format "Request i -> Server"
                file.write(f" Request {req_id} -> {server_info}\n")
            file.write("\n")

    print(f"Step 4: Crossover complete. Results saved to {os.path.join(gen_folder, 'step4.txt')}.")
    return offspring_solutions

def calculate_total_violation(solution):
    """Calculates the total violation for a given solution."""
    total_violation = 0
    for req_id, server_info in solution:
        # Use req_id directly since it's already an integer
        request = requests[int(req_id) - 1]  # Access the request directly using the integer

        if "Edge Server" in server_info:
            server = next(es for es in edge_servers if es['id'] == int(server_info.split()[-1]))
        else:
            server = next(cs for cs in cloud_servers if cs['id'] == int(server_info.split()[-1]))

        processing_time = request['instruction_length'] / server['processing_speed']
        violation = max(0, processing_time - request['max_processing_time'])
        total_violation += violation
    return total_violation
```

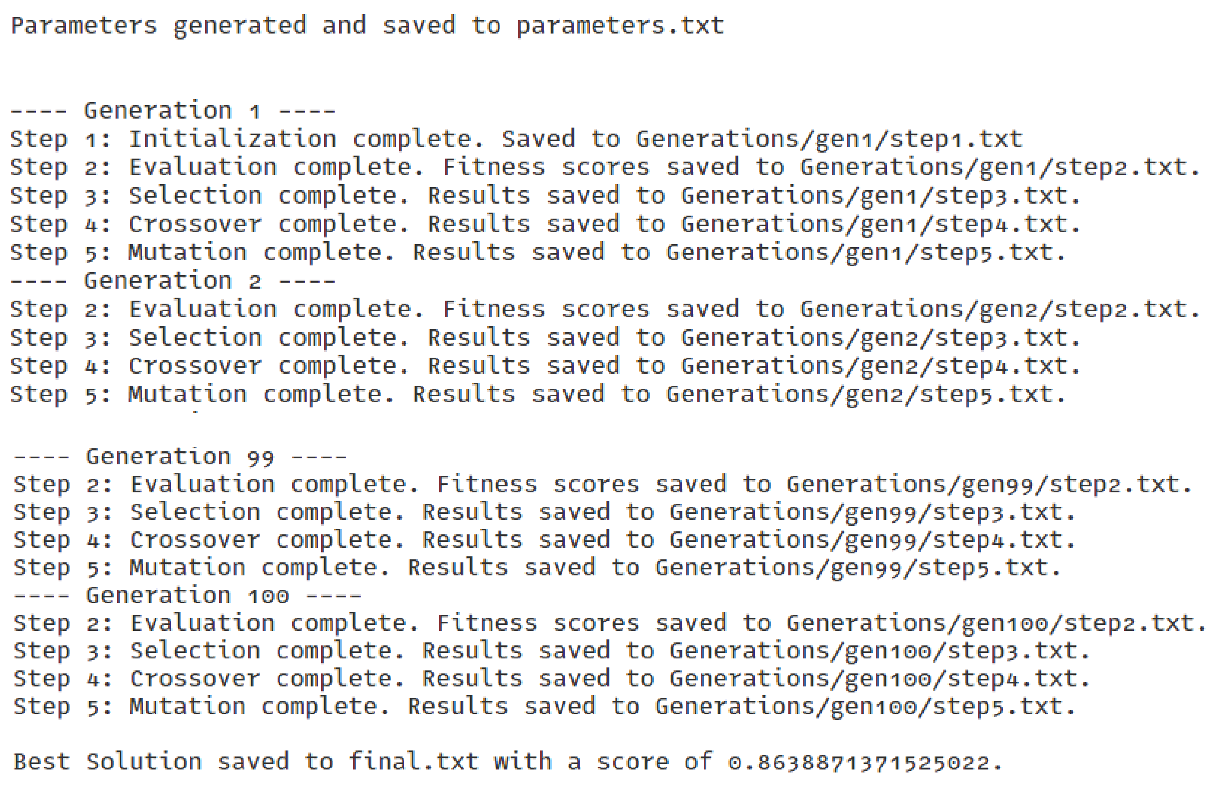
```python
def step_5_mutation(offspring_solutions, gen_folder):
    """Performs mutation on the offloading solutions and saves the results to step5.txt."""
    mutated_solutions = [solution.copy() for solution in offspring_solutions]  # Create a copy for mutation

    # Calculate the total number of allocations and mutations
    nallocations = chromosome_length * popsize
    nmut = int(nallocations * mutation_rate)  # Calculate the number of mutations

    for _ in range(nmut):
        rnd = random.randint(1, nallocations)  # Generate a random number to select a request for mutation
        rem = rnd % chromosome_length  # Determine which request to mutate

        # Determine which solution the random number corresponds to
        solution_index = (rnd - rem) // chromosome_length

        # Check that indices are within range
        if solution_index >= len(mutated_solutions) or rem >= len(mutated_solutions[solution_index]):
            continue  # Skip this iteration if indices are out of bounds

        # Identify the current request to reallocate
        current_request_id = rem + 1
        current_server_info = mutated_solutions[solution_index][rem][1]  # Access the server info correctly
        current_server_type = "edge" if "Edge Server" in current_server_info else "cloud"

        if current_server_type == "edge":
            # If the current server is an edge server, choose a different edge server or a cloud server
            new_server = random.choice(cloud_servers)  # Choose a cloud server
            mutated_solutions[solution_index][rem] = (f"{current_request_id}", f"Cloud Server {new_server['id']}")
        else:
            # Retrieve the area ID for the current request using the current_request_id
            current_request_area_id = None
            for request in requests:  # Assuming 'requests' is a list or dictionary containing your requests
                if request['id'] == current_request_id:  # Check if the current request matches
                    current_request_area_id = request['area_id']  # Get the area ID
                    break  # Exit the loop once the request is found

            # Proceed to choose an edge server based on the area ID
            if len(edge_servers) > 0 and random.random() < 0.5:  # Randomly choose to switch to an edge server
                # Filter edge servers to find those in the same area as the current request
                valid_edge_servers = [server for server in edge_servers if server['area_id'] == current_request_area_id]

                if valid_edge_servers:  # Check if there are any valid edge servers
                    new_server = random.choice(valid_edge_servers)  # Randomly select a valid edge server
                    mutated_solutions[solution_index][rem] = (f"{current_request_id}", f"Edge Server {new_server['id']}")
            else:
                # Otherwise, choose a different cloud server
                available_cloud_servers = [cs for cs in cloud_servers if cs['id'] != int(current_server_info.split()[-1])]
                if available_cloud_servers:
                    new_server = random.choice(available_cloud_servers)
                    mutated_solutions[solution_index][rem] = (f"{current_request_id}", f"Cloud Server {new_server['id']}")

    # Save mutated solutions to step5.txt in the specified generation folder
    with open(os.path.join(gen_folder, "step5.txt"), "w") as file:
        file.write("Mutated Offloading Solutions:\n")
        for idx, solution in enumerate(mutated_solutions, 1):
            file.write(f"Solution {idx}:\n")
            for req_id, server_info in solution:
                file.write(f" Request {req_id} -> {server_info}\n")
            file.write("\n")

    print(f"Step 5: Mutation complete. Results saved to {os.path.join(gen_folder, 'step5.txt')}.")
    return mutated_solutions  # Return the mutated solutions for further processing
def final_evaluation(offloading_solutions):
    """Evaluates all solutions and calculates their fitness scores."""
    violations_list = []

    # Calculate violations for each solution
    for solution in offloading_solutions:
        total_violation = 0

        for req_id, server_info in solution:
            # Get the request details
            request = next(req for req in requests if req['id'] == int(req_id))

            # Determine the server type and processing speed
            if "Edge Server" in server_info:
                server = next(es for es in edge_servers if es['id'] == int(server_info.split()[-1]))
                processing_speed = server['processing_speed']
            else:
                server = next(cs for cs in cloud_servers if cs['id'] == int(server_info.split()[-1]))
                processing_speed = server['processing_speed']
            # Calculate processing time
            processing_time = request['instruction_length'] / processing_speed

            # Calculate processing time violation
            violation = max(0, processing_time - request['max_processing_time'])
            total_violation += violation

        violations_list.append(total_violation)

    # Find the maximum violation among all solutions
    max_violation = max(violations_list) if violations_list else 0

    # Calculate fitness scores for each solution and return them as tuples
    fitness_scores = []
    for i, total_violation in enumerate(violations_list):
        if max_violation > 0:
            fitness_score = 1 / (total_violation / max_violation + 1)
        else:
            fitness_score = 1.0  # No violations
        fitness_scores.append((offloading_solutions[i], fitness_score))  # Append tuple (solution, fitness_score)

    return fitness_scores  # Return tuples of solutions and their fitness scores

def main():

    cleanup()

    print("\n")
    # Step 0: Generate parameters
    step_0_generate_parameters()
    print("\n")

    # Create main "Generations" folder and first generation folder "gen1"
    main_folder = "Generations"
    os.makedirs(main_folder, exist_ok=True)
    gen1_folder = os.path.join(main_folder, "gen1")
    os.makedirs(gen1_folder, exist_ok=True)

    print("---- Generation 1 ----")

    # Step 1: Initialization for generation 1
4   offloading_solutions_gen1 = step_1_initialization(gen1_folder)

    # Step 2: Evaluate the solutions for generation 1
    fitness_scores_gen1 = step_2_evaluate_solutions(offloading_solutions_gen1, gen1_folder)

    # Perform selection for generation 1
    selected_solutions_gen1 = step_3_selection(offloading_solutions_gen1, fitness_scores_gen1, gen1_folder)

    # Save crossover results for generation 1
    offspring_solutions_gen1 = step_4_crossover(selected_solutions_gen1, gen1_folder)

    # Perform mutation on the offspring solutions for generation 1
    mutated_solutions_gen1 = step_5_mutation(offspring_solutions_gen1, gen1_folder)

    # Create folders for generations 2 to 10 and perform evaluation
    mutated_solutions = mutated_solutions_gen1  # Initialize for the loop
    for gen in range(2, max_num_generations + 1):
        gen_folder = os.path.join(main_folder, f"gen{gen}")
        os.makedirs(gen_folder, exist_ok=True)

        print(f"---- Generation {gen} ----")  # Print the generation header

        # Use the mutated solutions from the previous generation
        offloading_solutions = mutated_solutions

        # Evaluate the solutions for the current generation
        fitness_scores = step_2_evaluate_solutions(offloading_solutions, gen_folder)

        # Perform selection for the current generation
        selected_solutions = step_3_selection(offloading_solutions, fitness_scores, gen_folder)

        # Step 4: Perform crossover on selected solutions and save results
        offspring_solutions = step_4_crossover(selected_solutions, gen_folder)

        # In the loop for subsequent generations, perform mutation on the offspring solutions
        mutated_solutions = step_5_mutation(offspring_solutions, gen_folder)

    final_fitness_scores=final_evaluation(mutated_solutions)
```



```python
    # Find the solution with the maximum fitness score
    best_solution, best_score = max(final_fitness_scores, key=lambda x: x[1])

    # Write the best solution and its score to final.txt
    with open("final.txt", 'w') as final_file:
        final_file.write(f"Best Solution (Score: {best_score}):\n")

        # Formatting the solution for output
        for req_id, server_info in best_solution:
            final_file.write(f"Request {req_id} -> {server_info}\n")

    print(f"\nBest Solution saved to final.txt with a score of {best_score}.")


if __name__ == "__main__":
    main()
```

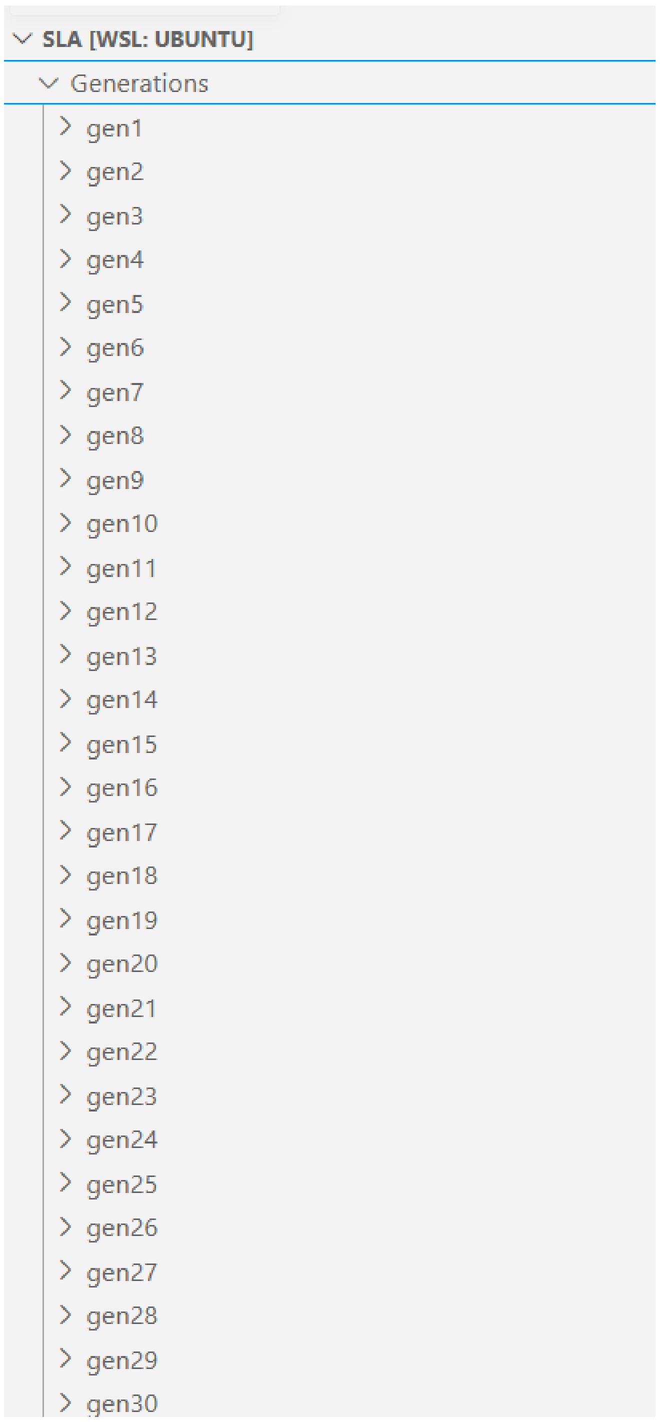
Outputs

Terminal Output:

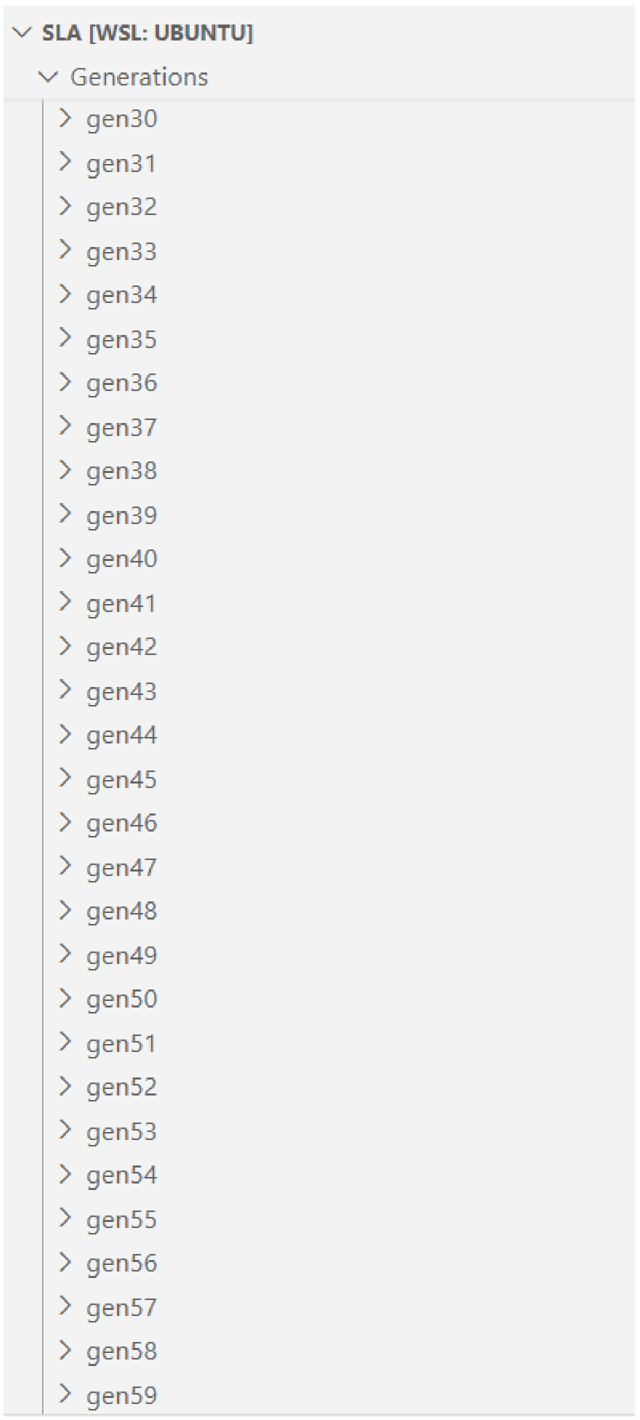


```
Parameters generated and saved to parameters.txt


---- Generation 1 ----
Step 1: Initialization complete. Saved to Generations/gen1/step1.txt
Step 2: Evaluation complete. Fitness scores saved to Generations/gen1/step2.txt.
Step 3: Selection complete. Results saved to Generations/gen1/step3.txt.
Step 4: Crossover complete. Results saved to Generations/gen1/step4.txt.
Step 5: Mutation complete. Results saved to Generations/gen1/step5.txt.
---- Generation 2 ----
Step 2: Evaluation complete. Fitness scores saved to Generations/gen2/step2.txt.
Step 3: Selection complete. Results saved to Generations/gen2/step3.txt.
Step 4: Crossover complete. Results saved to Generations/gen2/step4.txt.
Step 5: Mutation complete. Results saved to Generations/gen2/step5.txt.

---- Generation 99 ----
Step 2: Evaluation complete. Fitness scores saved to Generations/gen99/step2.txt.
Step 3: Selection complete. Results saved to Generations/gen99/step3.txt.
Step 4: Crossover complete. Results saved to Generations/gen99/step4.txt.
Step 5: Mutation complete. Results saved to Generations/gen99/step5.txt.
---- Generation 100 ----
Step 2: Evaluation complete. Fitness scores saved to Generations/gen100/step2.txt.
Step 3: Selection complete. Results saved to Generations/gen100/step3.txt.
Step 4: Crossover complete. Results saved to Generations/gen100/step4.txt.
Step 5: Mutation complete. Results saved to Generations/gen100/step5.txt.

Best Solution saved to final.txt with a score of 0.8638871371525022.
```

File Structure:

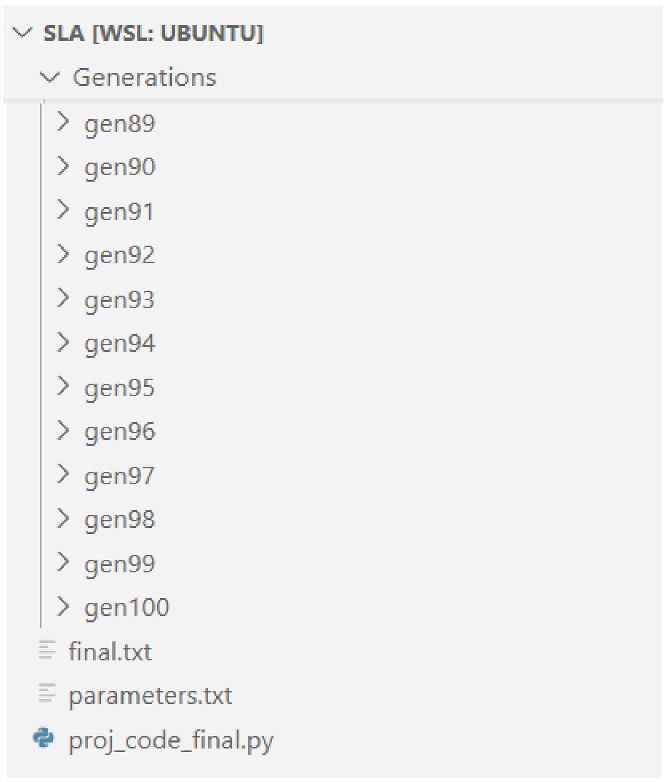


SLA [WSL: UBUNTU]
Generations
gen1
gen2
gen3
gen4
gen5
gen6
gen7
gen8
gen9
gen10
gen11
gen12
gen13
gen14
gen15
gen16
gen17
gen18
gen19
gen20
gen21
gen22
gen23
gen24
gen25
gen26
gen27
gen28
gen29
gen30



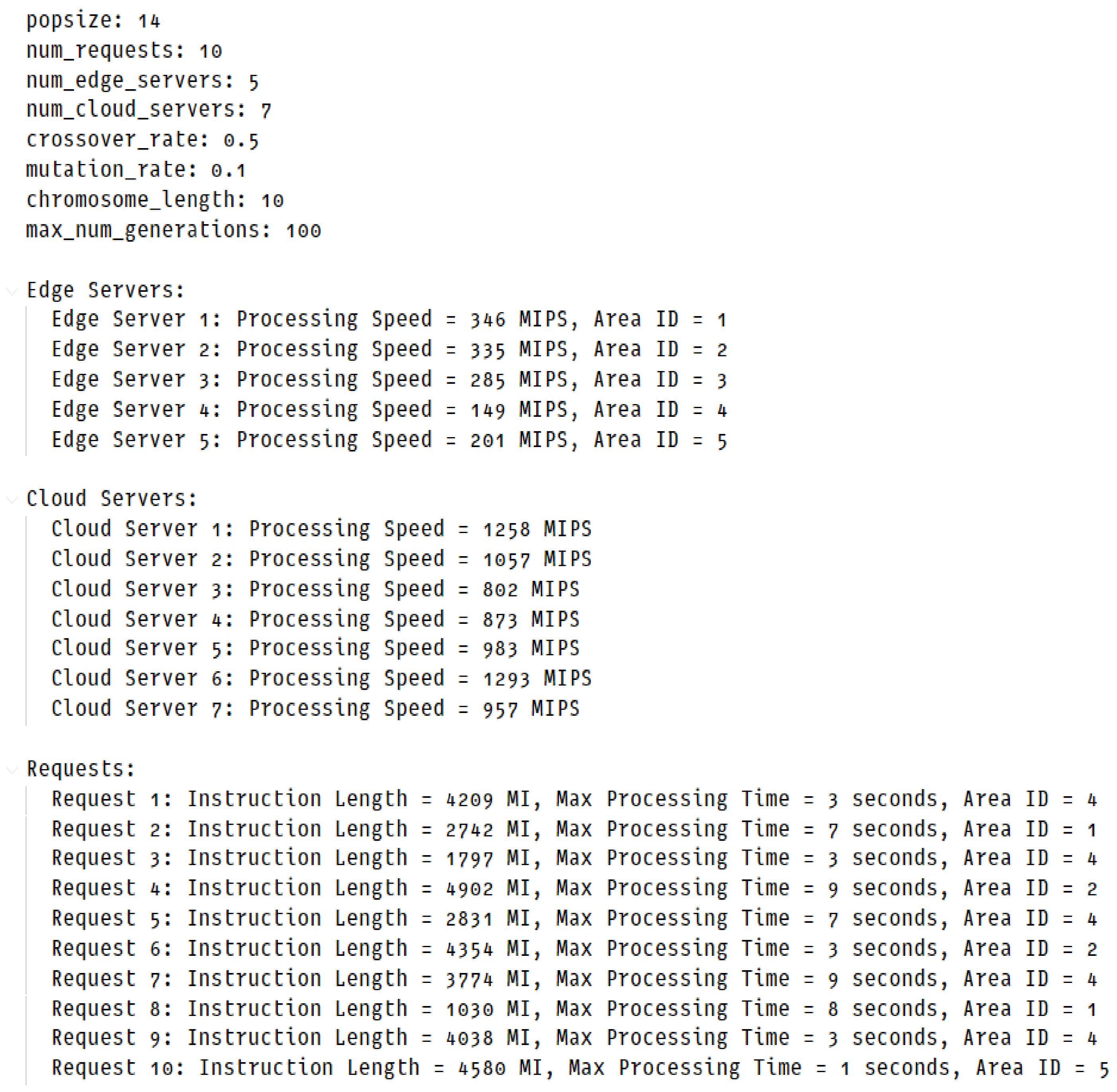
SLA [WSL: UBUNTU]
Generations
gen30
gen31
gen32
gen33
gen34
gen35
gen36
gen37
gen38
gen39
gen40
gen41
gen42
gen43
gen44
gen45
gen46
gen47
gen48
gen49
gen50
gen51
gen52
gen53
gen54
gen55
gen56
gen57
gen58
gen59

SLA [WSL: UBUNTU]
Generations
gen60
gen61
gen62
gen63
gen64
gen65
gen66
gen67
gen68
gen69
gen70
gen71
gen72
gen73
gen74
gen75
gen76
gen77
gen78
gen79
gen80
gen81
gen82
gen83
gen84
gen85
gen86
gen87
gen88
gen89



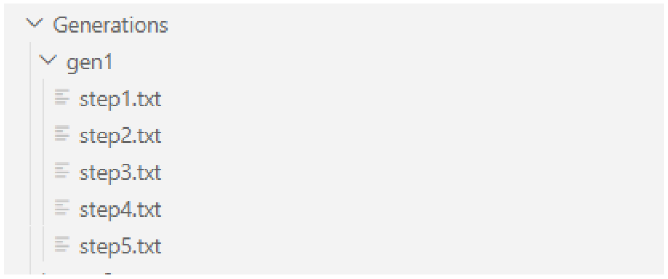
SLA [WSL: UBUNTU]
Generations
gen89
gen90
gen91
gen92
gen93
gen94
gen95
gen96
gen97
gen98
gen99
gen100
final.txt
parameters.txt
proj_code_final.py

Parameters.txt



```
popsize: 14
num_requests: 10
num_edge_servers: 5
num_cloud_servers: 7
crossover_rate: 0.5
mutation_rate: 0.1
chromosome_length: 10
max_num_generations: 100

Edge Servers:
  Edge Server 1: Processing Speed = 346 MIPS, Area ID = 1
  Edge Server 2: Processing Speed = 335 MIPS, Area ID = 2
  Edge Server 3: Processing Speed = 285 MIPS, Area ID = 3
  Edge Server 4: Processing Speed = 149 MIPS, Area ID = 4
  Edge Server 5: Processing Speed = 201 MIPS, Area ID = 5

Cloud Servers:
  Cloud Server 1: Processing Speed = 1258 MIPS
  Cloud Server 2: Processing Speed = 1057 MIPS
  Cloud Server 3: Processing Speed = 802 MIPS
  Cloud Server 4: Processing Speed = 873 MIPS
  Cloud Server 5: Processing Speed = 983 MIPS
  Cloud Server 6: Processing Speed = 1293 MIPS
  Cloud Server 7: Processing Speed = 957 MIPS

Requests:
  Request 1: Instruction Length = 4209 MI, Max Processing Time = 3 seconds, Area ID = 4
  Request 2: Instruction Length = 2742 MI, Max Processing Time = 7 seconds, Area ID = 1
  Request 3: Instruction Length = 1797 MI, Max Processing Time = 3 seconds, Area ID = 4
  Request 4: Instruction Length = 4902 MI, Max Processing Time = 9 seconds, Area ID = 2
  Request 5: Instruction Length = 2831 MI, Max Processing Time = 7 seconds, Area ID = 4
  Request 6: Instruction Length = 4354 MI, Max Processing Time = 3 seconds, Area ID = 2
  Request 7: Instruction Length = 3774 MI, Max Processing Time = 9 seconds, Area ID = 4
  Request 8: Instruction Length = 1030 MI, Max Processing Time = 8 seconds, Area ID = 1
  Request 9: Instruction Length = 4038 MI, Max Processing Time = 3 seconds, Area ID = 4
  Request 10: Instruction Length = 4580 MI, Max Processing Time = 1 seconds, Area ID = 5
```

Generations:

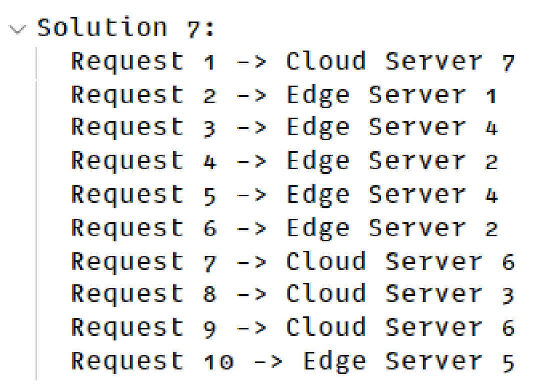


```
Generations
  gen1
    step1.txt
    step2.txt
    step3.txt
    step4.txt
    step5.txt
```

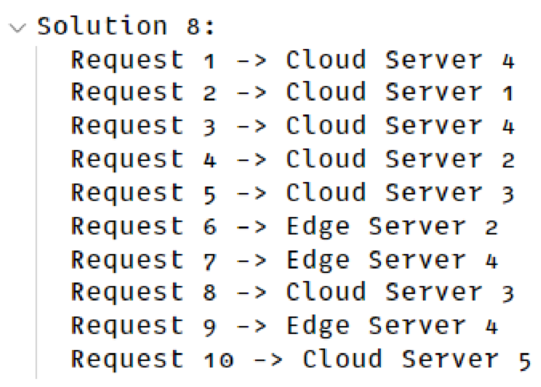
Gen1: step1.txt

```
Initialization of Offloading Solutions:
Solution 1:
  Request 1 -> Edge Server 4
  Request 2 -> Cloud Server 1
  Request 3 -> Cloud Server 7
  Request 4 -> Edge Server 2
  Request 5 -> Cloud Server 2
  Request 6 -> Edge Server 2
  Request 7 -> Cloud Server 5
  Request 8 -> Cloud Server 2
  Request 9 -> Edge Server 4
  Request 10 -> Cloud Server 2

Solution 2:
  Request 1 -> Cloud Server 2
  Request 2 -> Edge Server 1
  Request 3 -> Edge Server 4
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Edge Server 2
  Request 7 -> Edge Server 4
  Request 8 -> Edge Server 1
  Request 9 -> Cloud Server 5
  Request 10 -> Cloud Server 2

Solution 3:
  Request 1 -> Cloud Server 2
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 2
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 6
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Cloud Server 1
  Request 10 -> Cloud Server 4
```

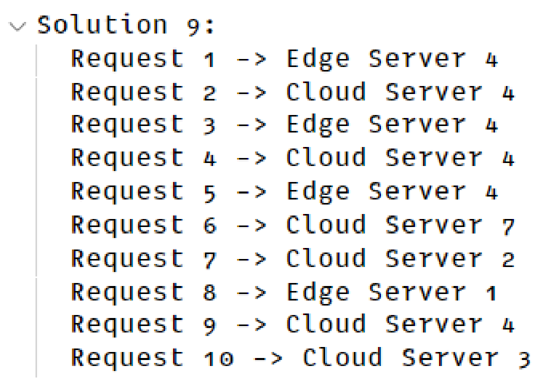
```
Solution 4:
  Request 1 -> Cloud Server 6
  Request 2 -> Edge Server 1
  Request 3 -> Edge Server 4
  Request 4 -> Cloud Server 5
  Request 5 -> Cloud Server 2
  Request 6 -> Cloud Server 4
  Request 7 -> Cloud Server 1
  Request 8 -> Cloud Server 5
  Request 9 -> Edge Server 4
  Request 10 -> Edge Server 5

Solution 5:
  Request 1 -> Edge Server 4
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 4
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 5
  Request 7 -> Cloud Server 1
  Request 8 -> Edge Server 1
  Request 9 -> Edge Server 4
  Request 10 -> Cloud Server 6

Solution 6:
  Request 1 -> Edge Server 4
  Request 2 -> Edge Server 1
  Request 3 -> Edge Server 4
  Request 4 -> Cloud Server 7
  Request 5 -> Edge Server 4
  Request 6 -> Edge Server 2
  Request 7 -> Edge Server 4
  Request 8 -> Edge Server 1
  Request 9 -> Cloud Server 2
  Request 10 -> Edge Server 5
```



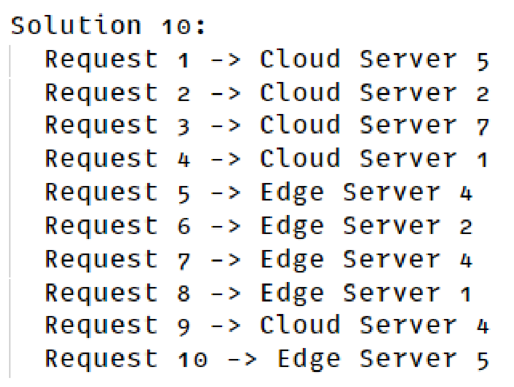
```
Solution 7:
  Request 1 -> Cloud Server 7
  Request 2 -> Edge Server 1
  Request 3 -> Edge Server 4
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Edge Server 2
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 3
  Request 9 -> Cloud Server 6
  Request 10 -> Edge Server 5
```



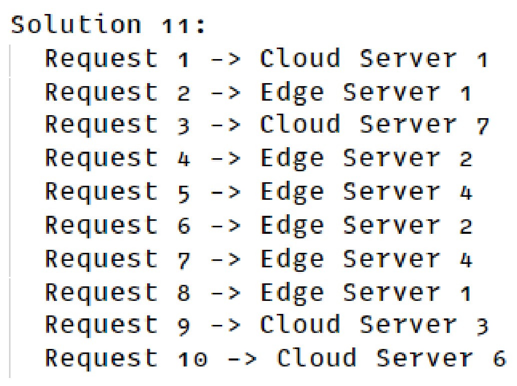
```
Solution 8:
  Request 1 -> Cloud Server 4
  Request 2 -> Cloud Server 1
  Request 3 -> Cloud Server 4
  Request 4 -> Cloud Server 2
  Request 5 -> Cloud Server 3
  Request 6 -> Edge Server 2
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 3
  Request 9 -> Edge Server 4
  Request 10 -> Cloud Server 5
```



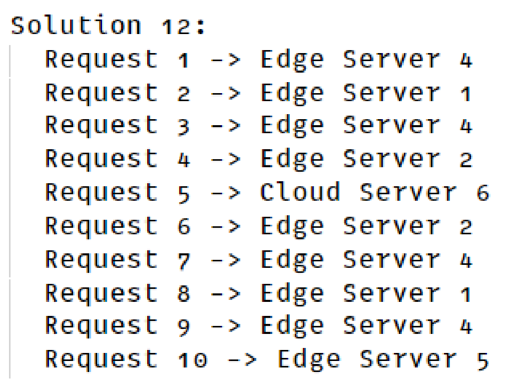
```
Solution 9:
  Request 1 -> Edge Server 4
  Request 2 -> Cloud Server 4
  Request 3 -> Edge Server 4
  Request 4 -> Cloud Server 4
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 7
  Request 7 -> Cloud Server 2
  Request 8 -> Edge Server 1
  Request 9 -> Cloud Server 4
  Request 10 -> Cloud Server 3
```



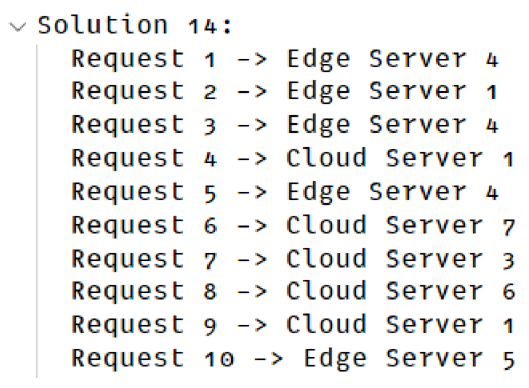
```
Solution 10:
  Request 1 -> Cloud Server 5
  Request 2 -> Cloud Server 2
  Request 3 -> Cloud Server 7
  Request 4 -> Cloud Server 1
  Request 5 -> Edge Server 4
  Request 6 -> Edge Server 2
  Request 7 -> Edge Server 4
  Request 8 -> Edge Server 1
  Request 9 -> Cloud Server 4
  Request 10 -> Edge Server 5
```



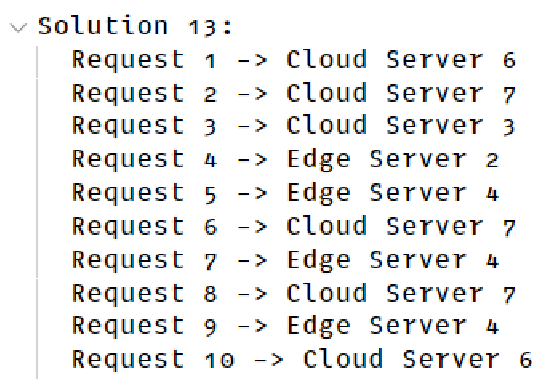
```
Solution 11:
  Request 1 -> Cloud Server 1
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 7
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Edge Server 2
  Request 7 -> Edge Server 4
  Request 8 -> Edge Server 1
  Request 9 -> Cloud Server 3
  Request 10 -> Cloud Server 6
```



```
Solution 12:
  Request 1 -> Edge Server 4
  Request 2 -> Edge Server 1
  Request 3 -> Edge Server 4
  Request 4 -> Edge Server 2
  Request 5 -> Cloud Server 6
  Request 6 -> Edge Server 2
  Request 7 -> Edge Server 4
  Request 8 -> Edge Server 1
  Request 9 -> Edge Server 4
  Request 10 -> Edge Server 5
```

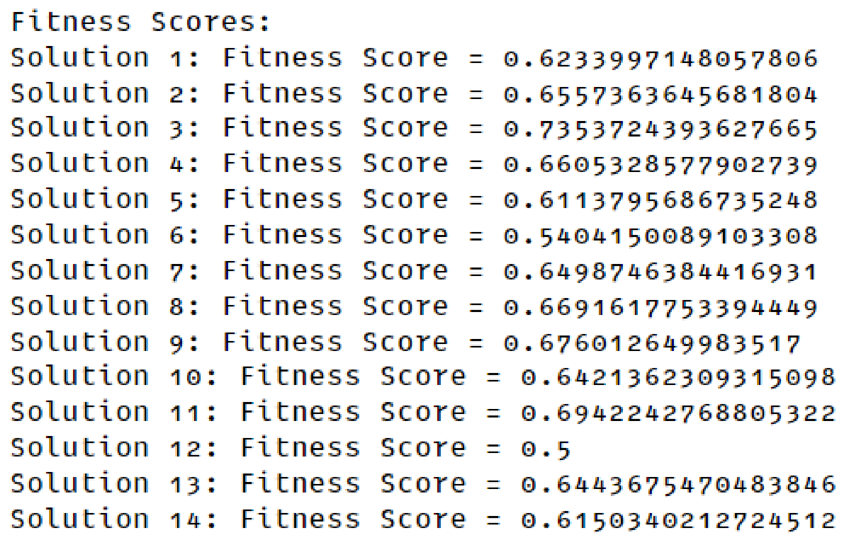


```
Solution 14:
  Request 1 -> Edge Server 4
  Request 2 -> Edge Server 1
  Request 3 -> Edge Server 4
  Request 4 -> Cloud Server 1
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 7
  Request 7 -> Cloud Server 3
  Request 8 -> Cloud Server 6
  Request 9 -> Cloud Server 1
  Request 10 -> Edge Server 5
```



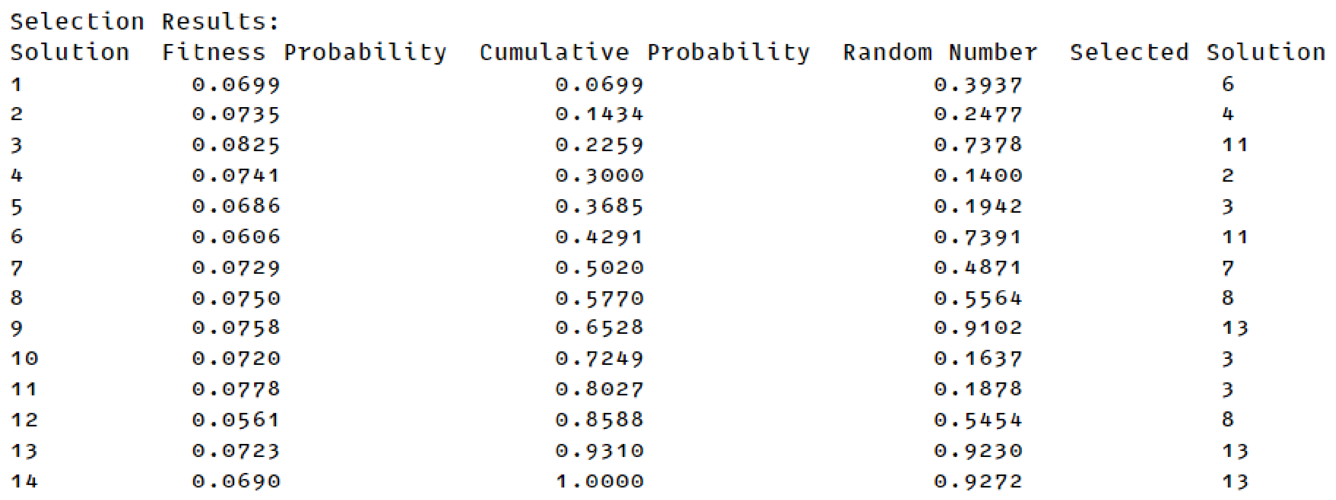
```
Solution 13:
  Request 1 -> Cloud Server 6
  Request 2 -> Cloud Server 7
  Request 3 -> Cloud Server 3
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 7
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Edge Server 4
  Request 10 -> Cloud Server 6
```

Gen1: Step2.txt



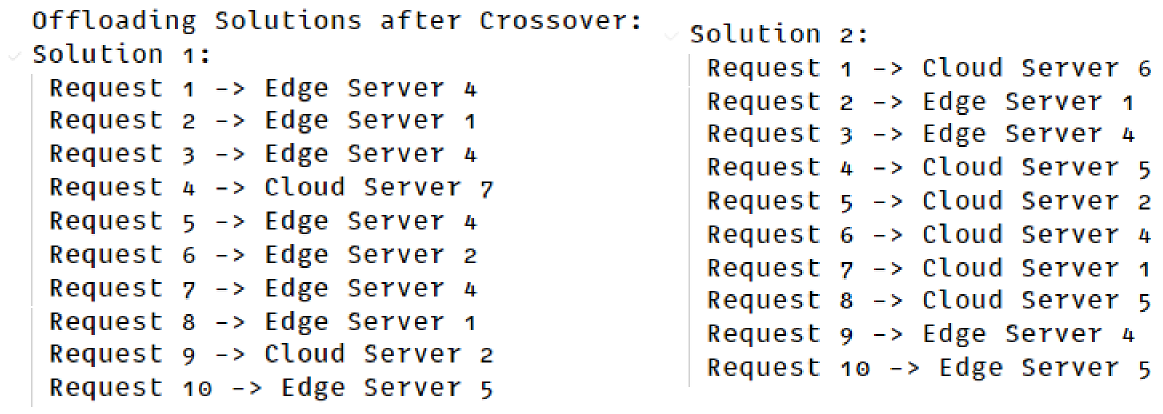
```
Fitness Scores:
Solution 1: Fitness Score = 0.6233997148057806
Solution 2: Fitness Score = 0.655736364568180 4
Solution 3: Fitness Score = 0.7353724393627665
Solution 4: Fitness Score = 0.6605328577902739
Solution 5: Fitness Score = 0.611379568673524 8
Solution 6: Fitness Score = 0.5404150089103308
Solution 7: Fitness Score = 0.6498746384416931
Solution 8: Fitness Score = 0.6691617753394449
Solution 9: Fitness Score = 0.67601264998351 7
Solution 10: Fitness Score = 0.6421362309315098
Solution 11: Fitness Score = 0.6942242768805322
Solution 12: Fitness Score = 0.5
Solution 13: Fitness Score = 0.6443675470483846
Solution 14: Fitness Score = 0.6150340212724512
```

Gen1: Step3.txt



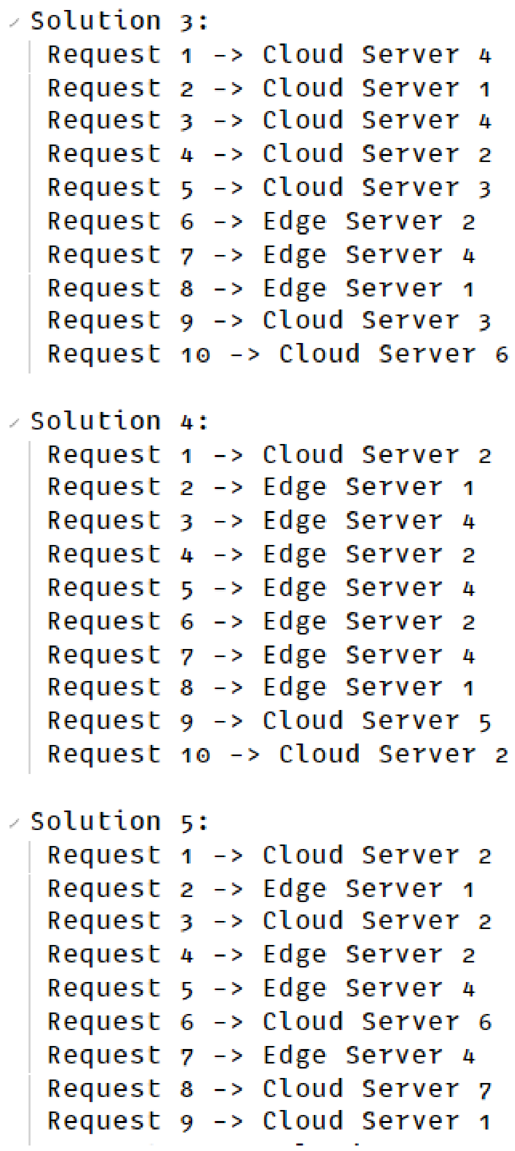
```
Selection Results:
Solution   Fitness Probability   Cumulative Probability   Random Number   Selected Solution
1            0.0699                   0.0699                   0.3937           6
2            0.0735                   0.1434                   0.2477           4
3            0.0825                   0.2259                   0.7378           11
4            0.0741                   0.3000                   0.1400           2
5            0.0686                   0.3685                   0.1942           3
6            0.0606                   0.4291                   0.7391           11
7            0.0729                   0.5020                   0.4871           7
8            0.0750                   0.5770                   0.5564           8
9            0.0758                   0.6528                   0.9102           13
10           0.0720                   0.7249                   0.1637           3
11           0.0778                   0.8027                   0.1878           3
12           0.0561                   0.8588                   0.5454           8
13           0.0723                   0.9310                   0.9230           13
14           0.0690                   1.0000                   0.9272           13
```

Gen1: Step4.txt

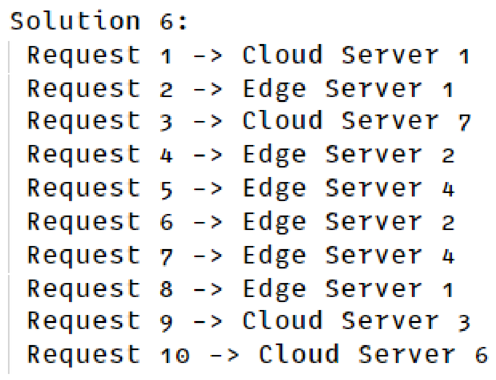


```
Offloading Solutions after Crossover:
Solution 1:
  Request 1 -> Edge Server 4
  Request 2 -> Edge Server 1
  Request 3 -> Edge Server 4
  Request 4 -> Cloud Server 7
  Request 5 -> Edge Server 4
  Request 6 -> Edge Server 2
  Request 7 -> Edge Server 4
  Request 8 -> Edge Server 1
  Request 9 -> Cloud Server 2
  Request 10 -> Edge Server 5
```

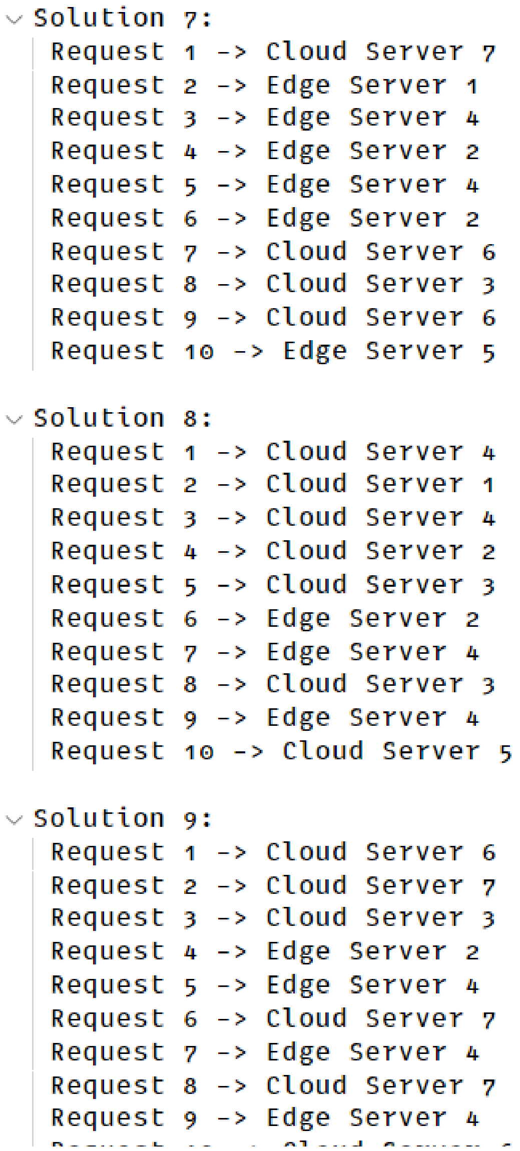
```
Solution 2:
  Request 1 -> Cloud Server 6
  Request 2 -> Edge Server 1
  Request 3 -> Edge Server 4
  Request 4 -> Cloud Server 5
  Request 5 -> Cloud Server 2
  Request 6 -> Cloud Server 4
  Request 7 -> Cloud Server 1
  Request 8 -> Cloud Server 5
  Request 9 -> Edge Server 4
  Request 10 -> Edge Server 5
```



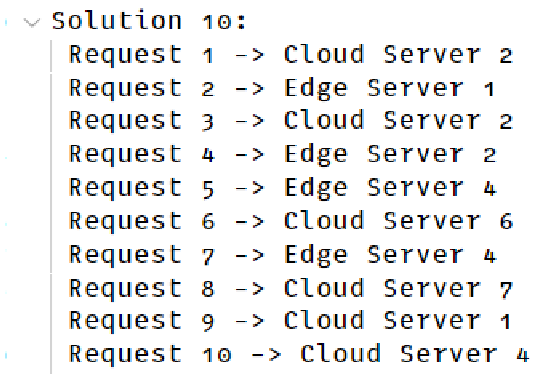
```
Solution 3:
  Request 1 -> Cloud Server 4
  Request 2 -> Cloud Server 1
  Request 3 -> Cloud Server 4
  Request 4 -> Cloud Server 2
  Request 5 -> Cloud Server 3
  Request 6 -> Edge Server 2
  Request 7 -> Edge Server 4
  Request 8 -> Edge Server 1
  Request 9 -> Cloud Server 3
  Request 10 -> Cloud Server 6

Solution 4:
  Request 1 -> Cloud Server 2
  Request 2 -> Edge Server 1
  Request 3 -> Edge Server 4
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Edge Server 2
  Request 7 -> Edge Server 4
  Request 8 -> Edge Server 1
  Request 9 -> Cloud Server 5
  Request 10 -> Cloud Server 2

Solution 5:
  Request 1 -> Cloud Server 2
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 2
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 6
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Cloud Server 1
```



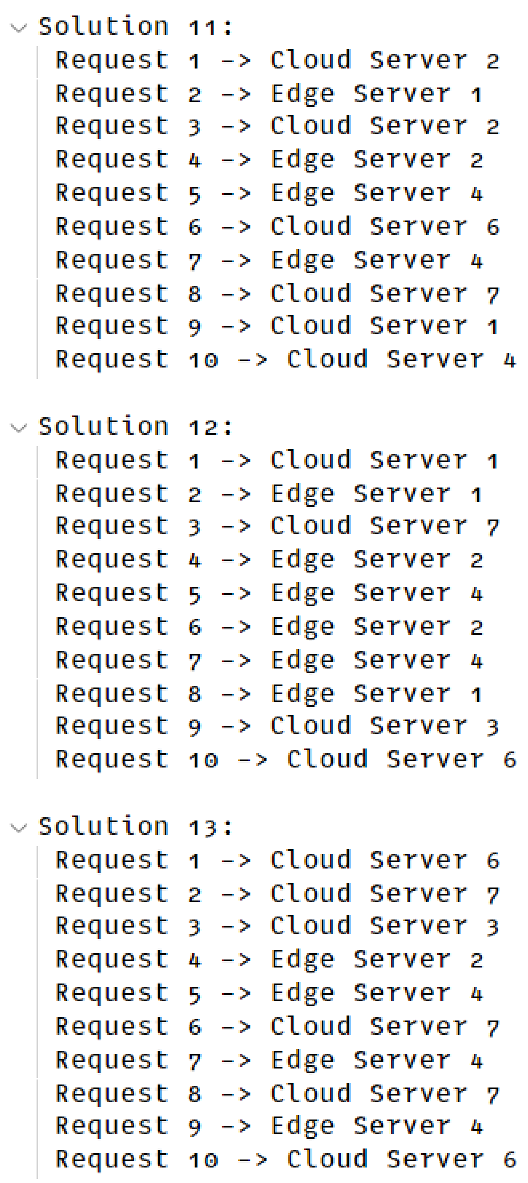
```
Solution 6:
  Request 1 -> Cloud Server 1
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 7
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Edge Server 2
  Request 7 -> Edge Server 4
  Request 8 -> Edge Server 1
  Request 9 -> Cloud Server 3
  Request 10 -> Cloud Server 6
```



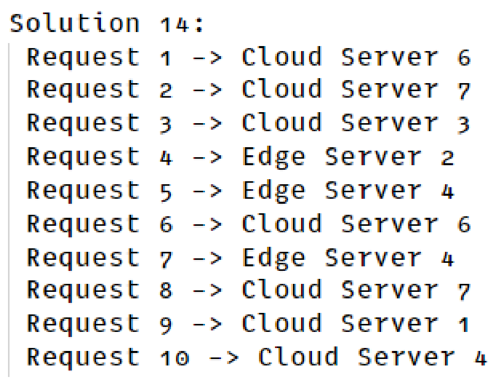
```
Solution 7:
  Request 1 -> Cloud Server 7
  Request 2 -> Edge Server 1
  Request 3 -> Edge Server 4
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Edge Server 2
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 3
  Request 9 -> Cloud Server 6
  Request 10 -> Edge Server 5

Solution 8:
  Request 1 -> Cloud Server 4
  Request 2 -> Cloud Server 1
  Request 3 -> Cloud Server 4
  Request 4 -> Cloud Server 2
  Request 5 -> Cloud Server 3
  Request 6 -> Edge Server 2
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 3
  Request 9 -> Edge Server 4
  Request 10 -> Cloud Server 5

Solution 9:
  Request 1 -> Cloud Server 6
  Request 2 -> Cloud Server 7
  Request 3 -> Cloud Server 3
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 7
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Edge Server 4
```



```
Solution 10:
  Request 1 -> Cloud Server 2
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 2
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 6
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Cloud Server 1
  Request 10 -> Cloud Server 4
```

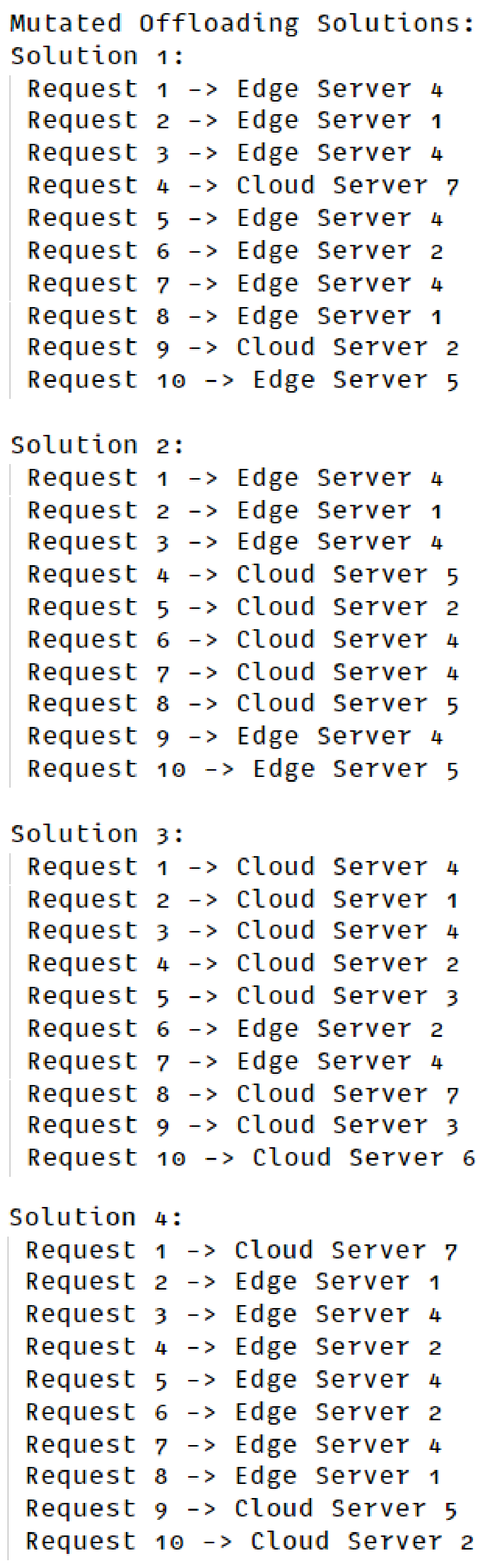


```
Solution 11:
  Request 1 -> Cloud Server 2
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 2
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 6
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Cloud Server 1
  Request 10 -> Cloud Server 4

Solution 12:
  Request 1 -> Cloud Server 1
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 7
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Edge Server 2
  Request 7 -> Edge Server 4
  Request 8 -> Edge Server 1
  Request 9 -> Cloud Server 3
  Request 10 -> Cloud Server 6

Solution 13:
  Request 1 -> Cloud Server 6
  Request 2 -> Cloud Server 7
  Request 3 -> Cloud Server 3
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 7
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Edge Server 4
  Request 10 -> Cloud Server 6
```

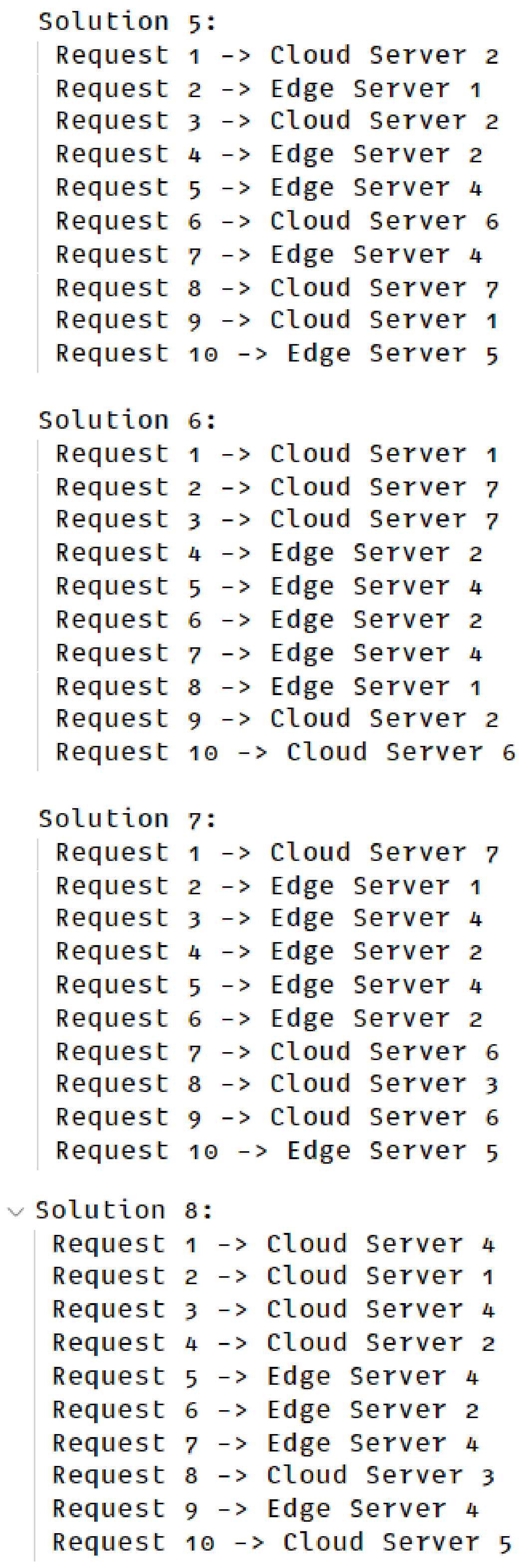


```
Solution 14:
  Request 1 -> Cloud Server 6
  Request 2 -> Cloud Server 7
  Request 3 -> Cloud Server 3
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 6
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Cloud Server 1
  Request 10 -> Cloud Server 4
```

Gen1: Step5.txt



```
Mutated Offloading Solutions:
Solution 1:
 Request 1 -> Edge Server 4
 Request 2 -> Edge Server 1
 Request 3 -> Edge Server 4
 Request 4 -> Cloud Server 7
 Request 5 -> Edge Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Edge Server 1
 Request 9 -> Cloud Server 2
 Request 10 -> Edge Server 5

Solution 2:
 Request 1 -> Edge Server 4
 Request 2 -> Edge Server 1
 Request 3 -> Edge Server 4
 Request 4 -> Cloud Server 5
 Request 5 -> Cloud Server 2
 Request 6 -> Cloud Server 4
 Request 7 -> Cloud Server 4
 Request 8 -> Cloud Server 5
 Request 9 -> Edge Server 4
 Request 10 -> Edge Server 5

Solution 3:
 Request 1 -> Cloud Server 4
 Request 2 -> Cloud Server 1
 Request 3 -> Cloud Server 4
 Request 4 -> Cloud Server 2
 Request 5 -> Cloud Server 3
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 3
 Request 10 -> Cloud Server 6

Solution 4:
 Request 1 -> Cloud Server 7
 Request 2 -> Edge Server 1
 Request 3 -> Edge Server 4
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Edge Server 1
 Request 9 -> Cloud Server 5
 Request 10 -> Cloud Server 2
```

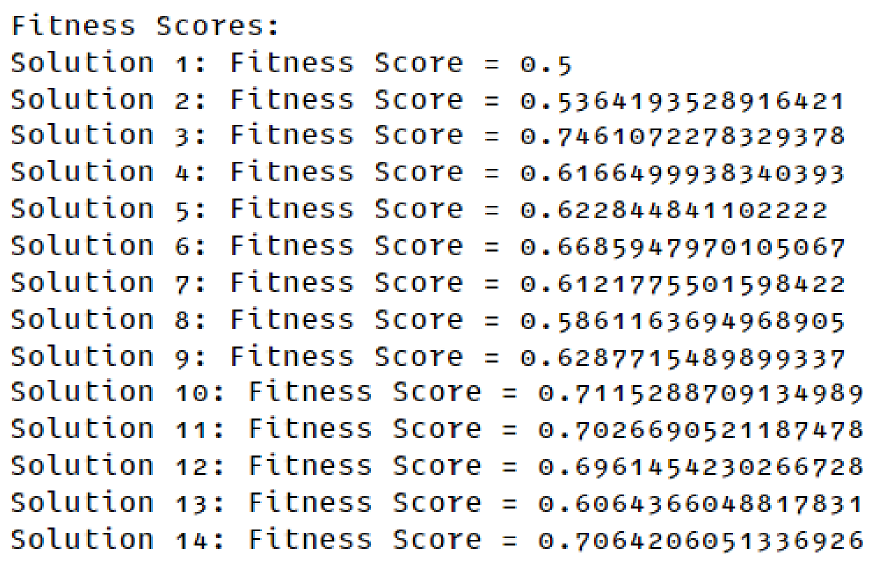


```
Solution 5:
 Request 1 -> Cloud Server 2
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Edge Server 5

Solution 6:
 Request 1 -> Cloud Server 1
 Request 2 -> Cloud Server 7
 Request 3 -> Cloud Server 7
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Edge Server 1
 Request 9 -> Cloud Server 2
 Request 10 -> Cloud Server 6

Solution 7:
 Request 1 -> Cloud Server 7
 Request 2 -> Edge Server 1
 Request 3 -> Edge Server 4
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Cloud Server 6
 Request 8 -> Cloud Server 3
 Request 9 -> Cloud Server 6
 Request 10 -> Edge Server 5

Solution 8:
 Request 1 -> Cloud Server 4
 Request 2 -> Cloud Server 1
 Request 3 -> Cloud Server 4
 Request 4 -> Cloud Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 3
 Request 9 -> Edge Server 4
 Request 10 -> Cloud Server 5
```

```
Solution 9:
  Request 1 -> Cloud Server 6
  Request 2 -> Cloud Server 7
  Request 3 -> Cloud Server 7
  Request 4 -> Cloud Server 6
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 7
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Edge Server 4
  Request 10 -> Cloud Server 6

Solution 10:
  Request 1 -> Cloud Server 2
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 2
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 6
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Cloud Server 1
  Request 10 -> Cloud Server 6

Solution 11:
  Request 1 -> Cloud Server 2
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 2
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 6
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Cloud Server 1
  Request 10 -> Cloud Server 4
```

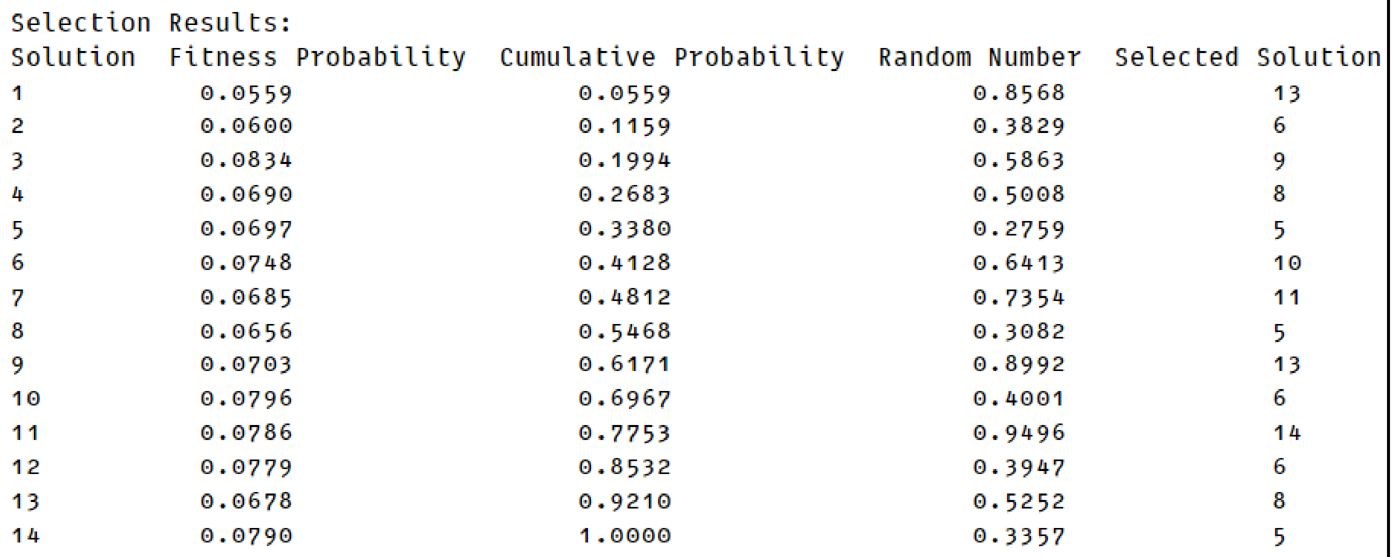
```
Solution 12:
  Request 1 -> Cloud Server 1
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 7
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 1
  Request 7 -> Edge Server 4
  Request 8 -> Edge Server 1
  Request 9 -> Cloud Server 3
  Request 10 -> Cloud Server 4

Solution 13:
  Request 1 -> Cloud Server 6
  Request 2 -> Cloud Server 7
  Request 3 -> Cloud Server 3
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 7
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Edge Server 4
  Request 10 -> Cloud Server 6

Solution 14:
  Request 1 -> Cloud Server 6
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 3
  Request 4 -> Edge Server 2
  Request 5 -> Edge Server 4
  Request 6 -> Cloud Server 6
  Request 7 -> Edge Server 4
  Request 8 -> Cloud Server 7
  Request 9 -> Cloud Server 1
  Request 10 -> Cloud Server 4
```

Gen2: Step2.txt



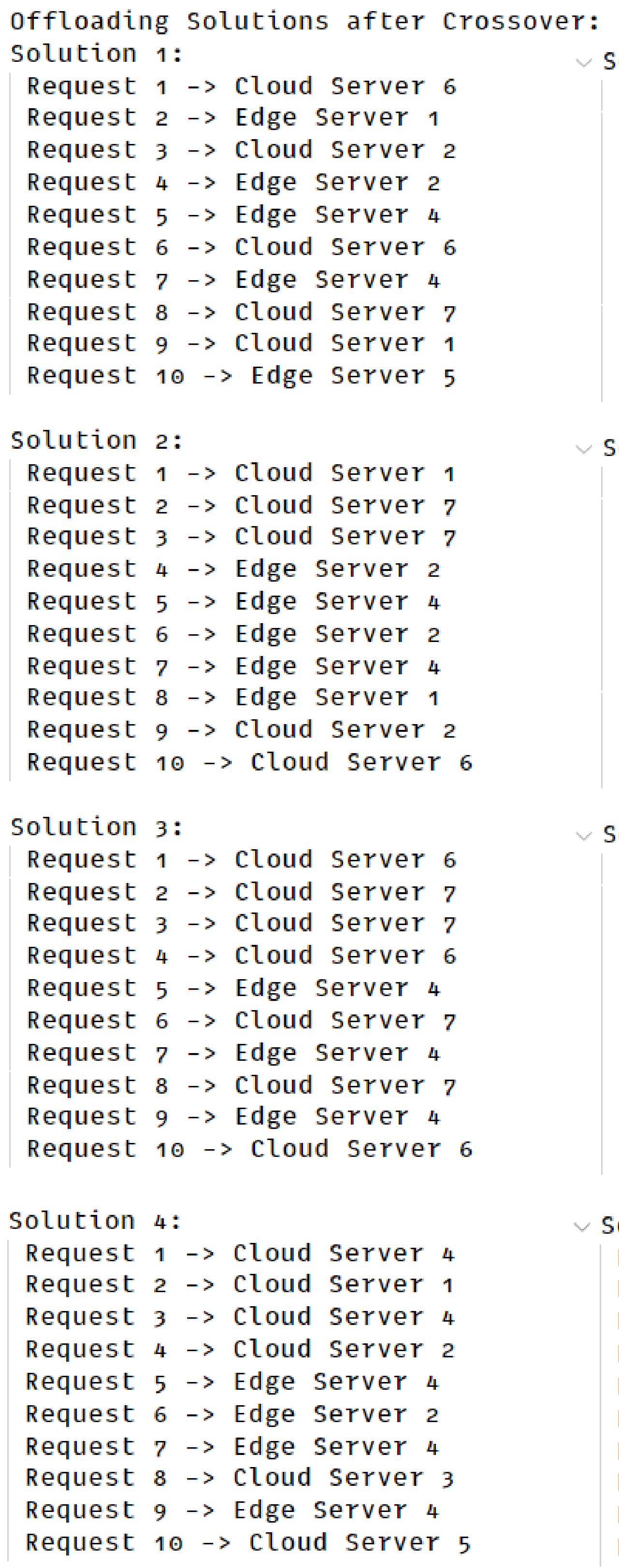
```
Fitness Scores:
Solution 1: Fitness Score = 0.5
Solution 2: Fitness Score = 0.5364193528916421
Solution 3: Fitness Score = 0.7461072278329378
Solution 4: Fitness Score = 0.6166499938340393
Solution 5: Fitness Score = 0.622844841102222
Solution 6: Fitness Score = 0.6685947970105067
Solution 7: Fitness Score = 0.6121775501598422
Solution 8: Fitness Score = 0.5861163694968905
Solution 9: Fitness Score = 0.6287715489899337
Solution 10: Fitness Score = 0.7115288709134989
Solution 11: Fitness Score = 0.7026690521187478
Solution 12: Fitness Score = 0.6961454230266728
Solution 13: Fitness Score = 0.6064366048817831
Solution 14: Fitness Score = 0.7064206051336926
```

Gen2: Step3.txt

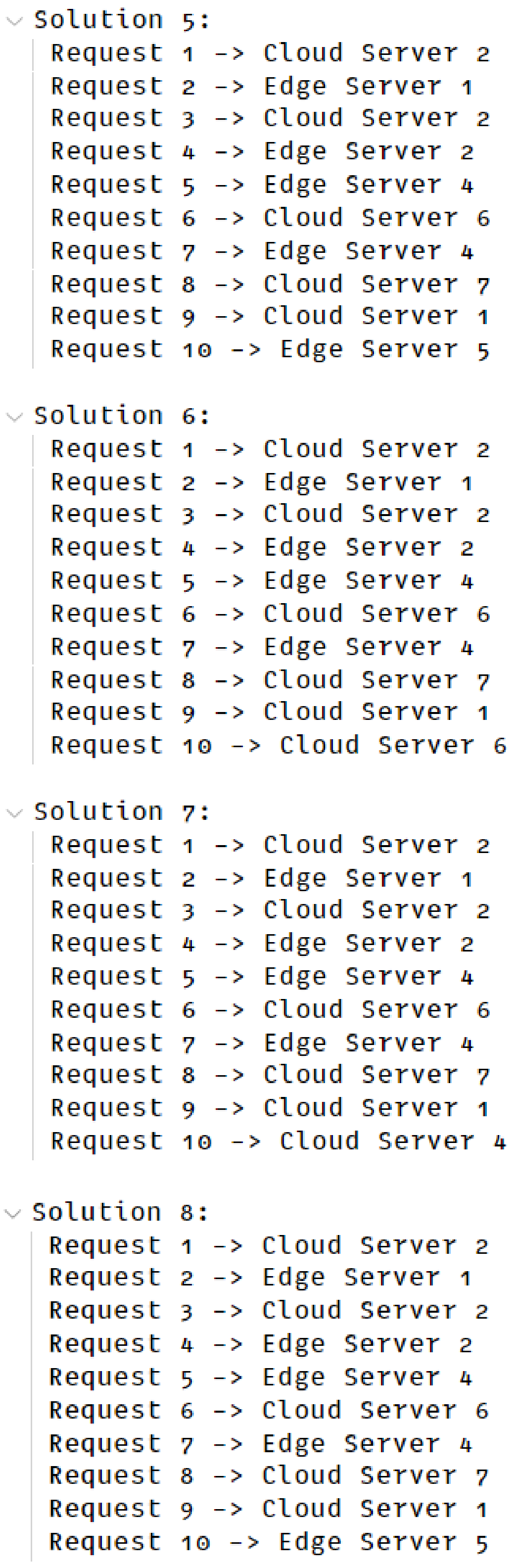


```
Selection Results:
Solution  Fitness Probability  Cumulative Probability  Random Number  Selected Solution
1           0.0559                   0.0559                0.8568            13
2           0.0600                   0.1159                0.3829            6
3           0.0834                   0.1994                0.5863            9
4           0.0690                   0.2683                0.5008            8
5           0.0697                   0.3380                0.2759            5
6           0.0748                   0.4128                0.6413            10
7           0.0685                   0.4812                0.7354            11
8           0.0656                   0.5468                0.3082            5
9           0.0703                   0.6171                0.8992            13
10          0.0796                   0.6967                0.4001            6
11          0.0786                   0.7753                0.9496            14
12          0.0779                   0.8532                0.3947            6
13          0.0678                   0.9210                0.5252            8
14          0.0790                   1.0000                0.3357            5
```

Gen2: Step4.txt



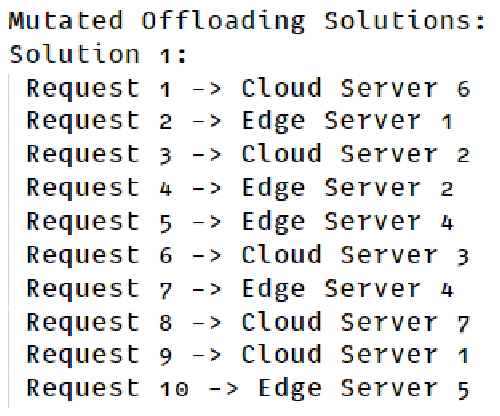
```
Offloading Solutions after Crossover:
Solution 1:
 Request 1 -> Cloud Server 6
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Edge Server 5

Solution 2:
 Request 1 -> Cloud Server 1
 Request 2 -> Cloud Server 7
 Request 3 -> Cloud Server 7
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Edge Server 1
 Request 9 -> Cloud Server 2
 Request 10 -> Cloud Server 6

Solution 3:
 Request 1 -> Cloud Server 6
 Request 2 -> Cloud Server 7
 Request 3 -> Cloud Server 7
 Request 4 -> Cloud Server 6
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 7
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Edge Server 4
 Request 10 -> Cloud Server 6

Solution 4:
 Request 1 -> Cloud Server 4
 Request 2 -> Cloud Server 1
 Request 3 -> Cloud Server 4
 Request 4 -> Cloud Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 3
 Request 9 -> Edge Server 4
 Request 10 -> Cloud Server 5
```



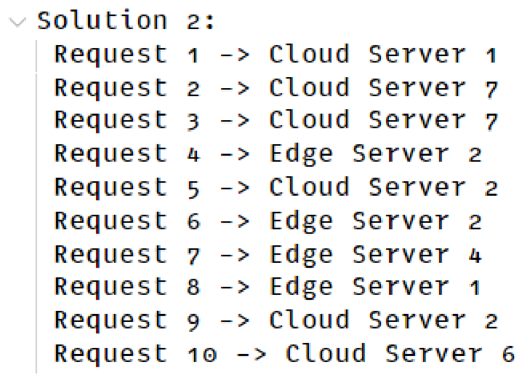
```
Solution 5:
 Request 1 -> Cloud Server 2
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Edge Server 5

Solution 6:
 Request 1 -> Cloud Server 2
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Cloud Server 6

Solution 7:
 Request 1 -> Cloud Server 2
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Cloud Server 4

Solution 8:
 Request 1 -> Cloud Server 2
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Edge Server 5
```

```
Solution 9:
 Request 1 -> Cloud Server 6
 Request 2 -> Cloud Server 7
 Request 3 -> Cloud Server 3
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Cloud Server 6

Solution 10:
 Request 1 -> Cloud Server 1
 Request 2 -> Cloud Server 7
 Request 3 -> Cloud Server 7
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Edge Server 1
 Request 9 -> Cloud Server 2
 Request 10 -> Cloud Server 6

Solution 11:
 Request 1 -> Cloud Server 6
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 3
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Cloud Server 4

Solution 12:
 Request 1 -> Cloud Server 1
 Request 2 -> Cloud Server 7
 Request 3 -> Cloud Server 7
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Edge Server 1
 Request 9 -> Cloud Server 2
 Request 10 -> Cloud Server 6

Solution 13:
 Request 1 -> Cloud Server 4
 Request 2 -> Cloud Server 1
 Request 3 -> Cloud Server 4
 Request 4 -> Cloud Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 3
 Request 9 -> Edge Server 4
 Request 10 -> Cloud Server 5

Solution 14:
 Request 1 -> Cloud Server 2
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Edge Server 5
```

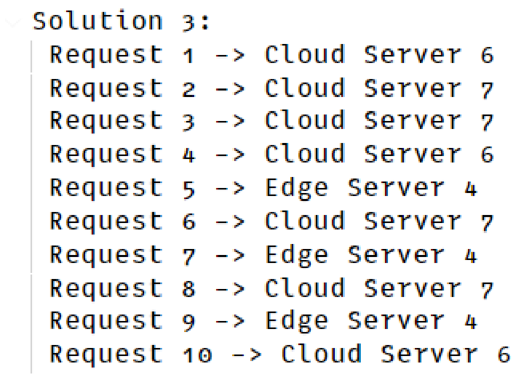
Gen2: Step5.txt



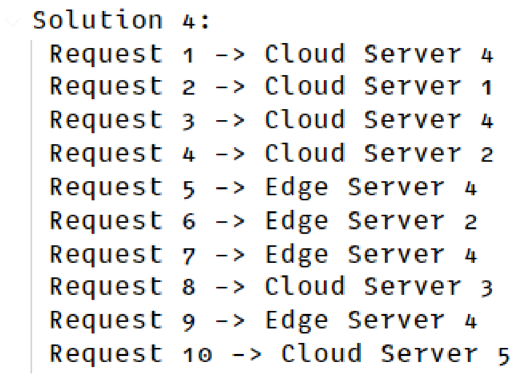
```
Mutated Offloading Solutions:
Solution 1:
 Request 1 -> Cloud Server 6
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 3
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Edge Server 5
```



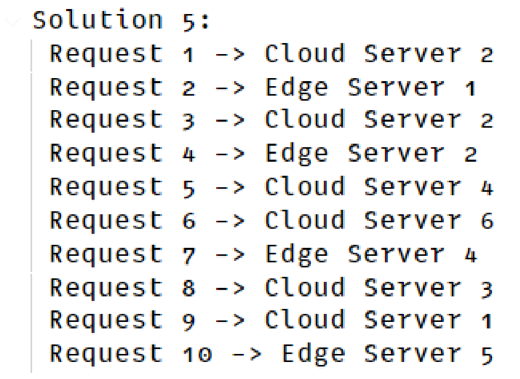
```
Solution 2:
 Request 1 -> Cloud Server 1
 Request 2 -> Cloud Server 7
 Request 3 -> Cloud Server 7
 Request 4 -> Edge Server 2
 Request 5 -> Cloud Server 2
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Edge Server 1
 Request 9 -> Cloud Server 2
 Request 10 -> Cloud Server 6
```



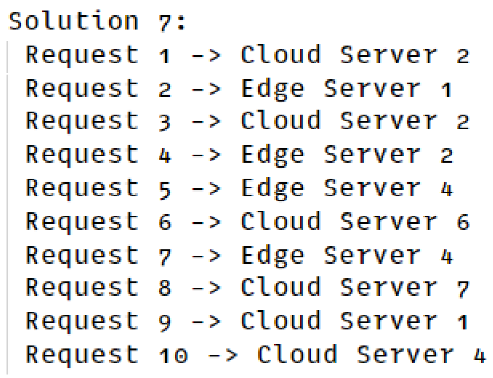
```
Solution 3:
 Request 1 -> Cloud Server 6
 Request 2 -> Cloud Server 7
 Request 3 -> Cloud Server 7
 Request 4 -> Cloud Server 6
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 7
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Edge Server 4
 Request 10 -> Cloud Server 6
```



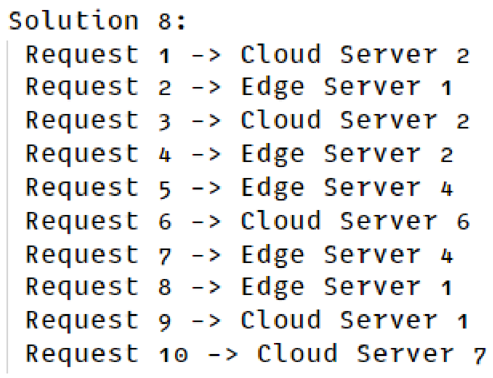
```
Solution 4:
 Request 1 -> Cloud Server 4
 Request 2 -> Cloud Server 1
 Request 3 -> Cloud Server 4
 Request 4 -> Cloud Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 3
 Request 9 -> Edge Server 4
 Request 10 -> Cloud Server 5
```



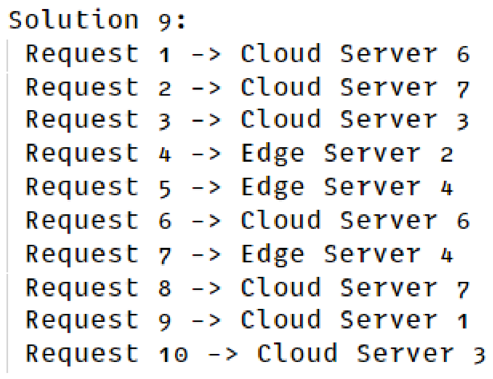
```
Solution 5:
 Request 1 -> Cloud Server 2
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Cloud Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 3
 Request 9 -> Cloud Server 1
 Request 10 -> Edge Server 5
```



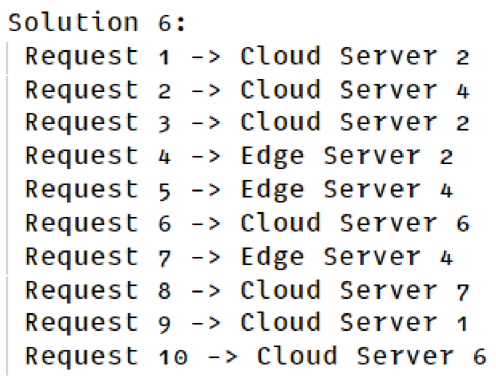
```
Solution 7:
 Request 1 -> Cloud Server 2
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Cloud Server 4
```



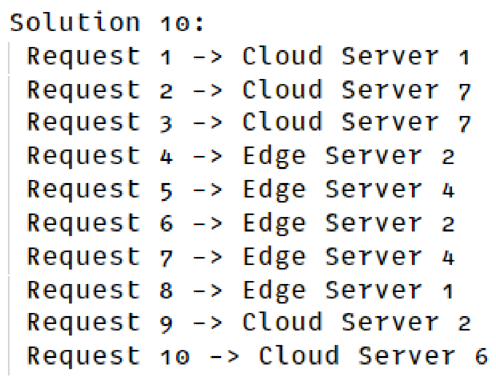
```
Solution 8:
 Request 1 -> Cloud Server 2
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Edge Server 1
 Request 9 -> Cloud Server 1
 Request 10 -> Cloud Server 7
```



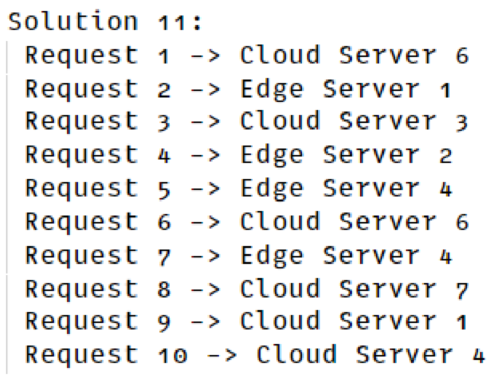
```
Solution 9:
 Request 1 -> Cloud Server 6
 Request 2 -> Cloud Server 7
 Request 3 -> Cloud Server 3
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Cloud Server 3
```



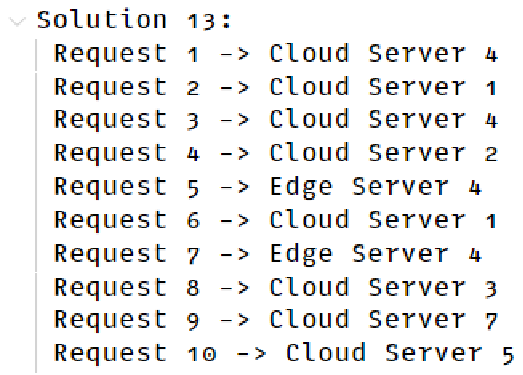
```
Solution 6:
 Request 1 -> Cloud Server 2
 Request 2 -> Cloud Server 4
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Cloud Server 6
```



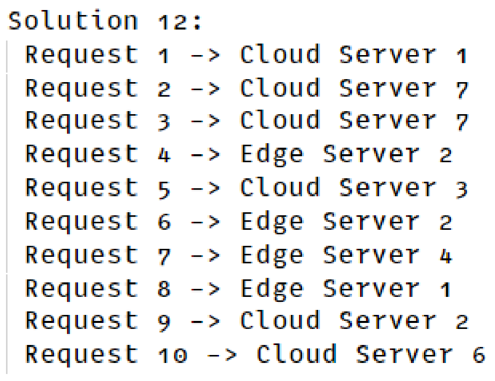
```
Solution 10:
 Request 1 -> Cloud Server 1
 Request 2 -> Cloud Server 7
 Request 3 -> Cloud Server 7
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Edge Server 1
 Request 9 -> Cloud Server 2
 Request 10 -> Cloud Server 6
```



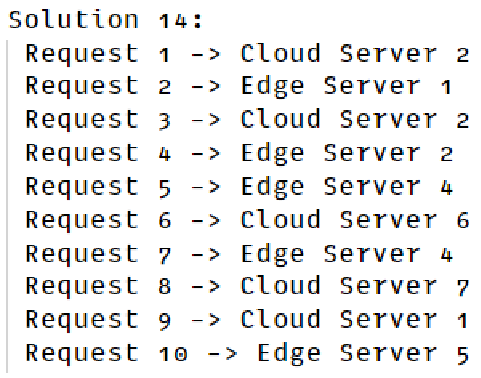
```
Solution 11:
 Request 1 -> Cloud Server 6
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 3
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Cloud Server 4
```



```
Solution 13:
 Request 1 -> Cloud Server 4
 Request 2 -> Cloud Server 1
 Request 3 -> Cloud Server 4
 Request 4 -> Cloud Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 1
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 3
 Request 9 -> Cloud Server 7
 Request 10 -> Cloud Server 5
```

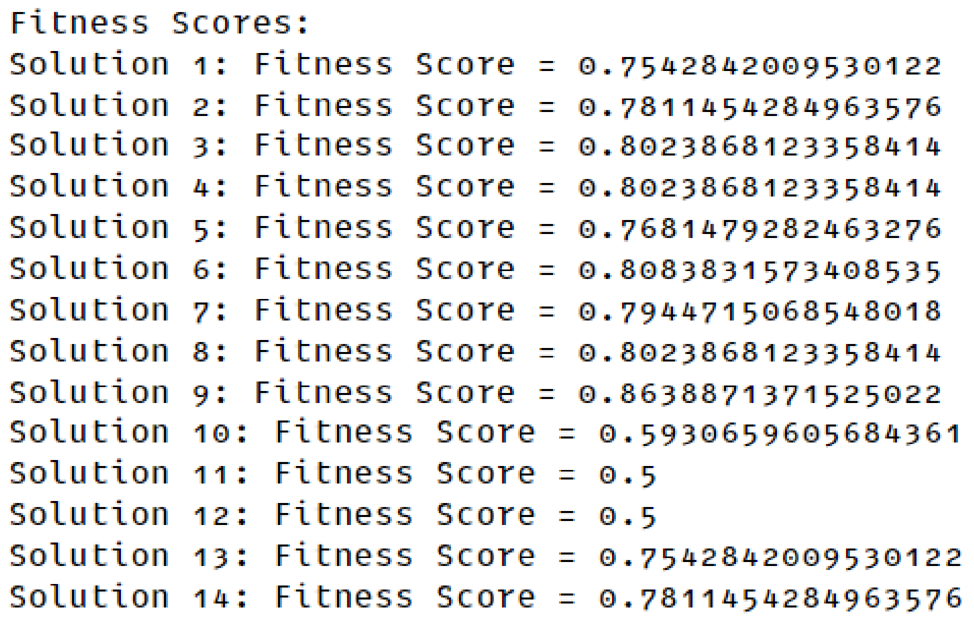


```
Solution 12:
 Request 1 -> Cloud Server 1
 Request 2 -> Cloud Server 7
 Request 3 -> Cloud Server 7
 Request 4 -> Edge Server 2
 Request 5 -> Cloud Server 3
 Request 6 -> Edge Server 2
 Request 7 -> Edge Server 4
 Request 8 -> Edge Server 1
 Request 9 -> Cloud Server 2
 Request 10 -> Cloud Server 6
```



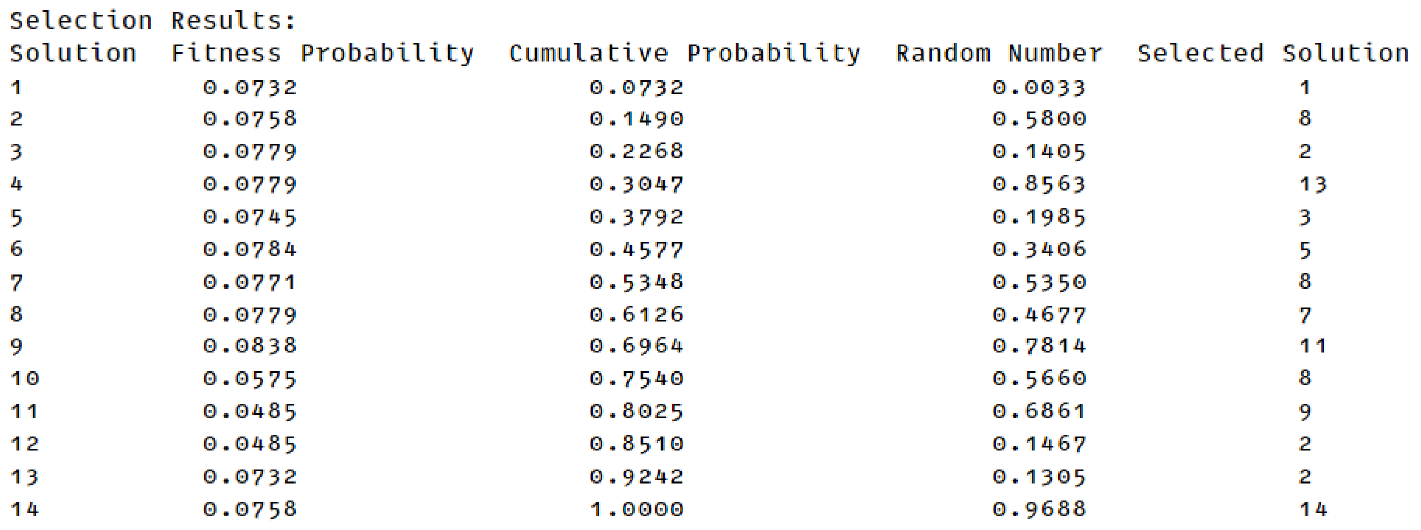
```
Solution 14:
 Request 1 -> Cloud Server 2
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 2
 Request 4 -> Edge Server 2
 Request 5 -> Edge Server 4
 Request 6 -> Cloud Server 6
 Request 7 -> Edge Server 4
 Request 8 -> Cloud Server 7
 Request 9 -> Cloud Server 1
 Request 10 -> Edge Server 5
```

Gen100: Step2.txt



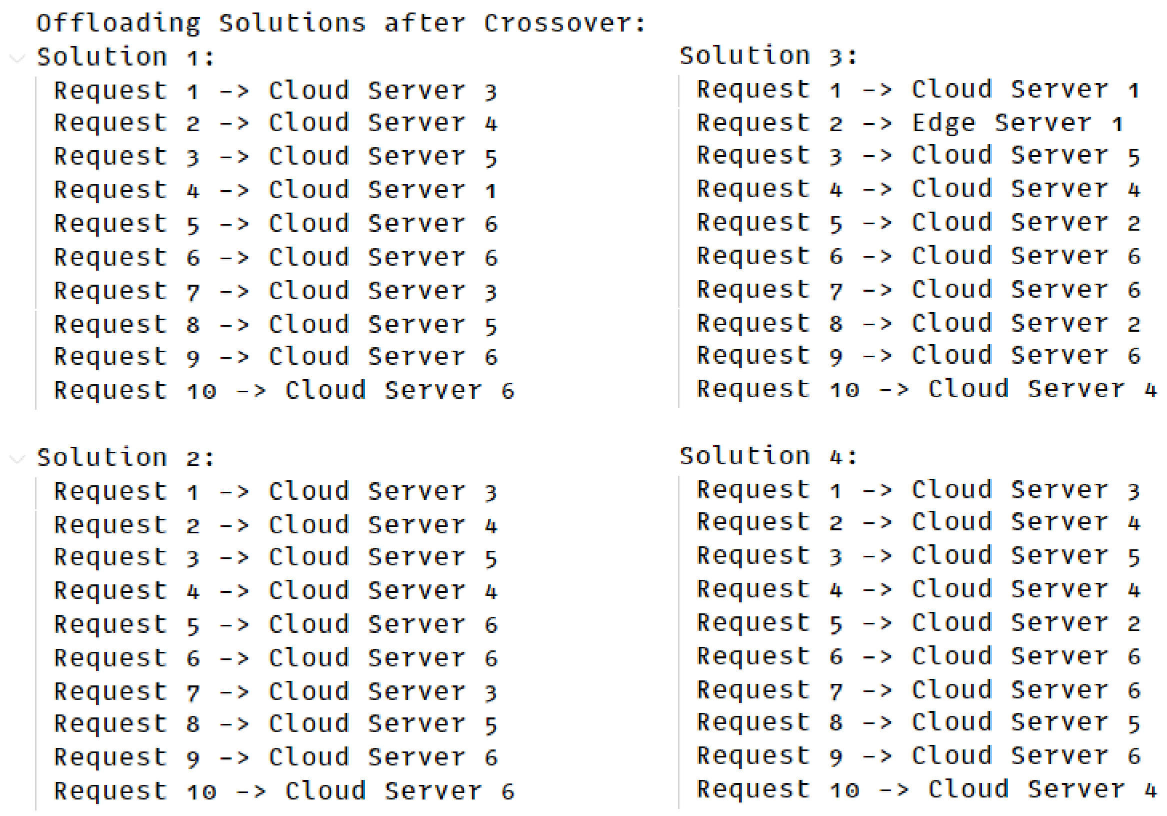
```
Fitness Scores:
Solution 1: Fitness Score = 0.7542842009530122
Solution 2: Fitness Score = 0.7811454284963576
Solution 3: Fitness Score = 0.802386812335841 4
Solution 4: Fitness Score = 0.8023868123358414
Solution 5: Fitness Score = 0.7681479282463276
Solution 6: Fitness Score = 0.8083831573408535
Solution 7: Fitness Score = 0.7944715068548018
Solution 8: Fitness Score = 0.8023868123358414
Solution 9: Fitness Score = 0.8638871371525022
Solution 10: Fitness Score = 0.5930659605684361
Solution 11: Fitness Score = 0.5
Solution 12: Fitness Score = 0.5
Solution 13: Fitness Score = 0.7542842009530122
Solution 14: Fitness Score = 0.7811454284963576
```

Gen100: Step3.txt



```
Selection Results:
Solution  Fitness Probability  Cumulative Probability  Random Number  Selected Solution
1           0.0732                   0.0732                 0.0033           1
2           0.0758                   0.1490                 0.5800           8
3           0.0779                   0.2268                 0.1405           2
4           0.0779                   0.3047                 0.8563           13
5           0.0745                   0.3792                 0.1985           3
6           0.0784                   0.4577                 0.3406           5
7           0.0771                   0.5348                 0.5350           8
8           0.0779                   0.6126                 0.4677           7
9           0.0838                   0.6964                 0.7814           11
10          0.0575                   0.7540                 0.5660           8
11          0.0485                   0.8025                 0.6861           9
12          0.0485                   0.8510                 0.1467           2
13          0.0732                   0.9242                 0.1305           2
14          0.0758                   1.0000                 0.9688           14
```

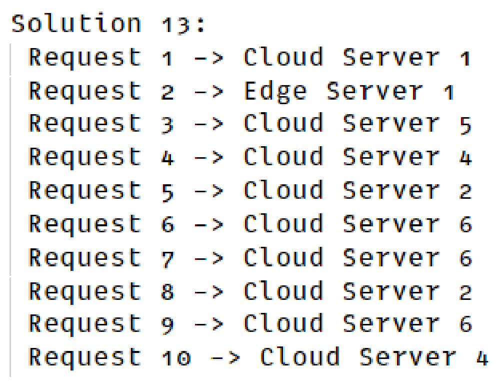
Gen100: Step4.txt

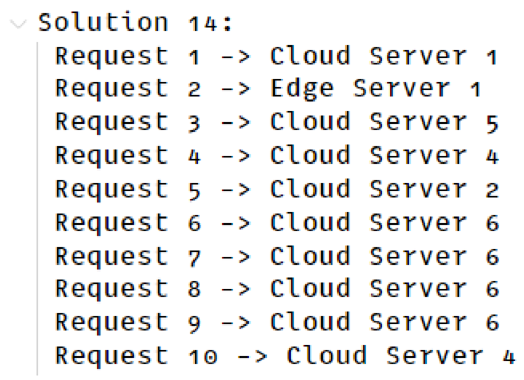


```
Offloading Solutions after Crossover:
Solution 1:
  Request 1 -> Cloud Server 3
  Request 2 -> Cloud Server 4
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 1
  Request 5 -> Cloud Server 6
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 3
  Request 8 -> Cloud Server 5
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 6

Solution 2:
  Request 1 -> Cloud Server 3
  Request 2 -> Cloud Server 4
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 6
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 3
  Request 8 -> Cloud Server 5
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 6

Solution 3:
  Request 1 -> Cloud Server 1
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 2
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 2
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 4

Solution 4:
  Request 1 -> Cloud Server 3
  Request 2 -> Cloud Server 4
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 2
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 5
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 4
```

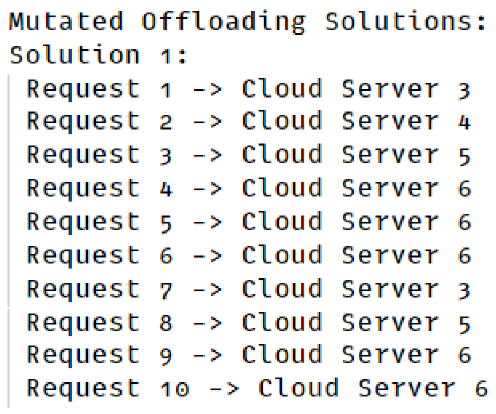
```
Solution 5:
  Request 1 -> Cloud Server 1
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 5
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 1
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 6

Solution 6:
  Request 1 -> Cloud Server 1
  Request 2 -> Cloud Server 5
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 2
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 6
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 3

Solution 7:
  Request 1 -> Cloud Server 3
  Request 2 -> Cloud Server 4
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 6
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 3
  Request 8 -> Cloud Server 5
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 6

Solution 8:
  Request 1 -> Cloud Server 1
  Request 2 -> Cloud Server 5
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 2
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 6
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 3
```

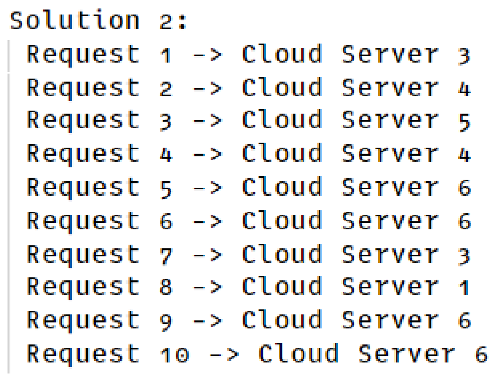
```
Solution 9:
  Request 1 -> Cloud Server 1
  Request 2 -> Cloud Server 4
  Request 3 -> Cloud Server 5
  Request 4 -> Edge Server 2
  Request 5 -> Cloud Server 4
  Request 6 -> Edge Server 2
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 6
  Request 9 -> Cloud Server 7
  Request 10 -> Cloud Server 4

Solution 10:
  Request 1 -> Cloud Server 1
  Request 2 -> Cloud Server 4
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 5
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 5
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 6

Solution 11:
  Request 1 -> Cloud Server 1
  Request 2 -> Cloud Server 4
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 6
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 3
  Request 8 -> Cloud Server 5
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 6

Solution 12:
  Request 1 -> Cloud Server 3
  Request 2 -> Cloud Server 4
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 1
  Request 5 -> Cloud Server 5
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 1
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 6
```

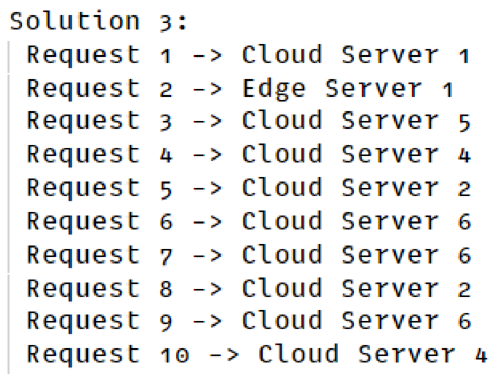

```
Solution 13:
  Request 1 -> Cloud Server 1
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 2
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 2
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 4
```

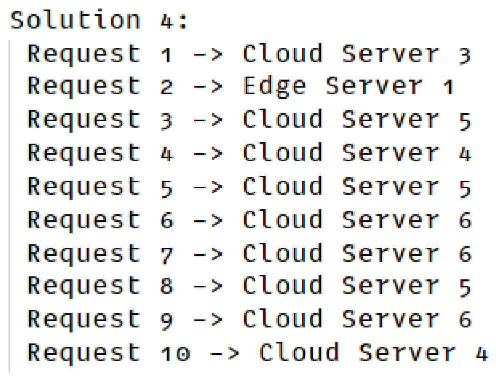

```
Solution 14:
  Request 1 -> Cloud Server 1
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 2
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 6
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 4
```

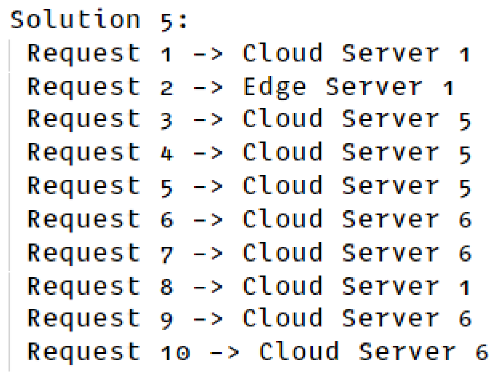
<u>Gen100: Step5.txt</u>

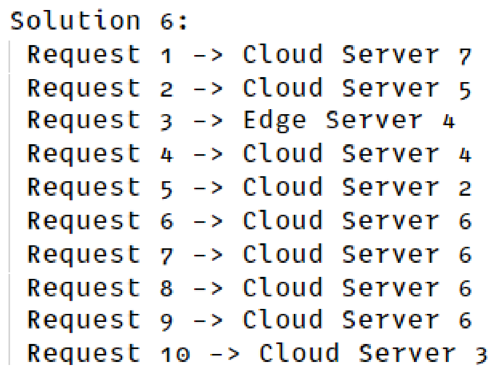

```
Mutated Offloading Solutions:
Solution 1:
  Request 1 -> Cloud Server 3
  Request 2 -> Cloud Server 4
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 6
  Request 5 -> Cloud Server 6
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 3
  Request 8 -> Cloud Server 5
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 6
```


```
Solution 2:
  Request 1 -> Cloud Server 3
  Request 2 -> Cloud Server 4
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 6
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 3
  Request 8 -> Cloud Server 1
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 6
```


```
Solution 3:
  Request 1 -> Cloud Server 1
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 2
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 2
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 4
```


```
Solution 4:
  Request 1 -> Cloud Server 3
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 5
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 5
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 4
```

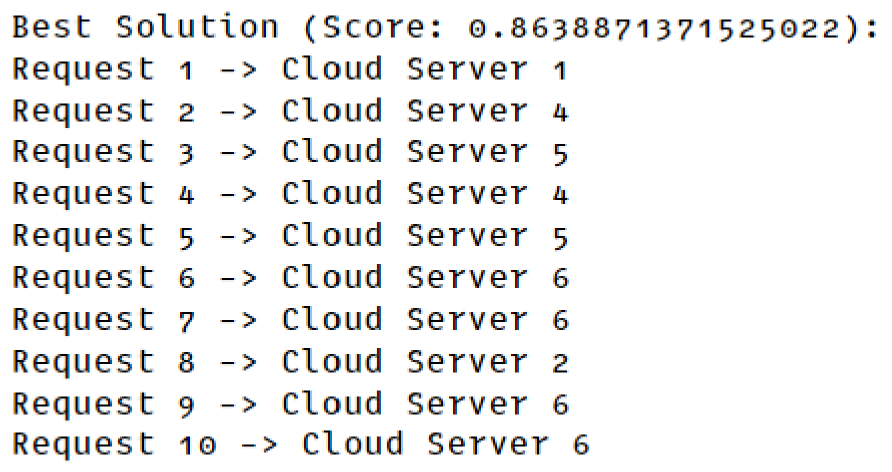

```
Solution 5:
  Request 1 -> Cloud Server 1
  Request 2 -> Edge Server 1
  Request 3 -> Cloud Server 5
  Request 4 -> Cloud Server 5
  Request 5 -> Cloud Server 5
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 1
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 6
```


```
Solution 6:
  Request 1 -> Cloud Server 7
  Request 2 -> Cloud Server 5
  Request 3 -> Edge Server 4
  Request 4 -> Cloud Server 4
  Request 5 -> Cloud Server 2
  Request 6 -> Cloud Server 6
  Request 7 -> Cloud Server 6
  Request 8 -> Cloud Server 6
  Request 9 -> Cloud Server 6
  Request 10 -> Cloud Server 3
```

```
Solution 7:
 Request 1 -> Cloud Server 3
 Request 2 -> Cloud Server 4
 Request 3 -> Cloud Server 5
 Request 4 -> Cloud Server 4
 Request 5 -> Cloud Server 6
 Request 6 -> Cloud Server 6
 Request 7 -> Cloud Server 3
 Request 8 -> Cloud Server 5
 Request 9 -> Cloud Server 6
 Request 10 -> Cloud Server 6

Solution 8:
 Request 1 -> Cloud Server 1
 Request 2 -> Cloud Server 5
 Request 3 -> Cloud Server 6
 Request 4 -> Cloud Server 4
 Request 5 -> Cloud Server 2
 Request 6 -> Cloud Server 6
 Request 7 -> Cloud Server 6
 Request 8 -> Cloud Server 6
 Request 9 -> Cloud Server 6
 Request 10 -> Cloud Server 3

Solution 9:
 Request 1 -> Cloud Server 1
 Request 2 -> Cloud Server 4
 Request 3 -> Cloud Server 5
 Request 4 -> Edge Server 2
 Request 5 -> Cloud Server 4
 Request 6 -> Edge Server 2
 Request 7 -> Cloud Server 6
 Request 8 -> Cloud Server 6
 Request 9 -> Cloud Server 7
 Request 10 -> Cloud Server 4

Solution 10:
 Request 1 -> Cloud Server 1
 Request 2 -> Cloud Server 4
 Request 3 -> Cloud Server 5
 Request 4 -> Cloud Server 4
 Request 5 -> Cloud Server 5
 Request 6 -> Cloud Server 6
 Request 7 -> Cloud Server 6
 Request 8 -> Cloud Server 2
 Request 9 -> Cloud Server 6
 Request 10 -> Cloud Server 6
```

```
Solution 11:
 Request 1 -> Cloud Server 1
 Request 2 -> Cloud Server 4
 Request 3 -> Edge Server 4
 Request 4 -> Cloud Server 4
 Request 5 -> Cloud Server 6
 Request 6 -> Cloud Server 6
 Request 7 -> Cloud Server 3
 Request 8 -> Cloud Server 5
 Request 9 -> Cloud Server 6
 Request 10 -> Cloud Server 6

Solution 12:
 Request 1 -> Cloud Server 3
 Request 2 -> Cloud Server 4
 Request 3 -> Cloud Server 5
 Request 4 -> Cloud Server 1
 Request 5 -> Cloud Server 5
 Request 6 -> Cloud Server 6
 Request 7 -> Cloud Server 6
 Request 8 -> Cloud Server 1
 Request 9 -> Cloud Server 6
 Request 10 -> Cloud Server 6

Solution 13:
 Request 1 -> Cloud Server 1
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 5
 Request 4 -> Cloud Server 4
 Request 5 -> Cloud Server 2
 Request 6 -> Cloud Server 6
 Request 7 -> Cloud Server 6
 Request 8 -> Cloud Server 2
 Request 9 -> Cloud Server 2
 Request 10 -> Cloud Server 4

Solution 14:
 Request 1 -> Cloud Server 1
 Request 2 -> Edge Server 1
 Request 3 -> Cloud Server 5
 Request 4 -> Cloud Server 4
 Request 5 -> Cloud Server 2
 Request 6 -> Cloud Server 6
 Request 7 -> Cloud Server 6
 Request 8 -> Cloud Server 6
 Request 9 -> Cloud Server 6
 Request 10 -> Cloud Server 4
```

Final.txt



```
Best Solution (Score: 0.8638871371525022):
Request 1 -> Cloud Server 1
Request 2 -> Cloud Server 4
Request 3 -> Cloud Server 5
Request 4 -> Cloud Server 4
Request 5 -> Cloud Server 5
Request 6 -> Cloud Server 6
Request 7 -> Cloud Server 6
Request 8 -> Cloud Server 2
Request 9 -> Cloud Server 6
Request 10 -> Cloud Server 6
```

The output of the code includes several files and folders, each providing different insights into the genetic algorithm's processing and results.

Here's a breakdown:

1. **Generations Folder:**

   This folder contains a separate folder for each generation (e.g., gen1, gen2, … up to the maximum number of generations defined).

   Each generation's folder contains the intermediate outputs at each stage of the genetic algorithm:

- Evaluated Solutions: Lists all solutions evaluated in that generation, with fitness scores indicating how well each solution met the processing time and energy constraints.
- Selected Solutions: Shows the solutions chosen based on fitness scores for crossover, representing the best-performing solutions that will help create the next generation.
- Crossover Results: Contains the newly created solutions from combining selected parent solutions. These solutions carry features from both parents and help explore new possibilities.
- Mutated Solutions: Shows the solutions after mutation, where random alterations introduce diversity. This prevents the algorithm from getting stuck in local optima by exploring other potential solutions.

2. Parameters.txt:

This file stores the simulation parameters used throughout the program, such as:

- Number of edge and cloud servers.
- Requests' details like processing times and instruction lengths.

This allows for easy reference to the environment settings used in the simulation.

3. final.txt:

This file is generated after the last generation completes.

It contains:

- **Best Solution:** The solution with the highest fitness score across all generations, representing the optimal offloading decisions.
- **Allocation Details:** For each request, this shows the assigned server (edge or cloud), illustrating how the tasks were distributed.
- **Fitness Score:** Reflects how well this solution performed in terms of processing time.

Summary

The output files include detailed logs of each generation’s population, fitness scores, and the changes in task allocation strategies as the algorithm progresses. They also record metrics such as processing time, energy consumption, and the degree of SLA compliance for each solution, allowing for comprehensive analysis. Additionally, final output files highlight the optimal offloading configuration achieved, illustrating how computational loads are distributed across edge and cloud resources. This data helps in understanding the trade-offs made between energy efficiency and performance, providing valuable insights into potential improvements in vehicular network resource management.

CONCLUSION

The convergence of cloud and edge computing with vehicular ad hoc networks (VANETs) offers substantial promise in transforming transportation systems by enabling responsive, scalable, and efficient communication between vehicles and surrounding infrastructure. This project underscores the importance of adopting advanced computational techniques, such as our Evolutionary Genetic Algorithm (EGA), which enhances resource management by dynamically optimizing task offloading. Through this approach, we not only improve response times in critical applications like accident prevention and real-time traffic management but also address the high energy demands that come with large-scale data processing in VANET environments.

One of the fundamental challenges in VANETs is balancing the need for low latency with energy efficiency to meet strict service level agreements (SLAs). Our proposed algorithm prioritizes energy conservation while minimizing delays, which is essential for applications requiring instant data processing and decision-making. This delicate balance ensures the operational reliability of VANETs in highly dynamic conditions, where computational loads and connectivity fluctuate constantly. By reducing the reliance on onboard vehicle processing and efficiently offloading tasks to edge and cloud servers, the algorithm contributes to a more robust and adaptive network architecture capable of supporting future growth in connected vehicle deployments.

Looking ahead, further exploration into lightweight, adaptive algorithms and AI-driven optimization will be vital for scaling VANET solutions. Enhancing in-vehicle computational capabilities and integrating green energy sources for data centers are also pivotal to addressing the environmental impact of widespread VANET adoption. As the demand for smart, connected mobility solutions grows, the insights from this project provide a foundation for developing a sustainable and resilient infrastructure that meets the demands of next-generation transportation while mitigating ecological concerns. The continued evolution of VANET technology, supported by edge and cloud computing, will be instrumental in fostering safer, more efficient, and environmentally conscious transportation systems.