ACIT 4610

Portfolio-2024

Purpose: Implement and study the application of algorithms/techniques discussed in this course.

The project will be carried out in **groups (2-4 students)**.

Submission Deadline: November 15 (Friday) at 15:00.

Submission Channel: To be confirmed.

Cheating / Plagiarism: Any cheating/plagiarism will be handled by OsloMet's policy. You can find more about it at https://student.oslomet.no/en/cheating

1 Assignment

There are **Four (04)** problems in this Portfolio. You need to implement any **Three (03)** of these problems as per the requirements and instructions mentioned in each problem.

2 Submission

You should deliver your report (in PDF), **which should include the GitHub link containing the code, datasets, and necessary instructions to run and test your implementation.** Note that we will download the code immediately after the submission deadline to limit the possibility of later updating.

All the group members should individually submit the SAME report for the group and the GitHub link.

3 Report

A combined report addressing all the requirements mentioned in the corresponding three problems. The combined report should be between 7500-3000 words in total. **The report must be submitted in PDF format**. The report should contain the names of all group members. Also, as mentioned above, the report should include the GitHub link containing the code, datasets, and necessary instructions to run and test your implementation.

Problem 1: Traffic Management Optimization Using Multi-Objective Evolutionary Algorithms

Objective:

Urban traffic management requires balancing multiple conflicting objectives, such as minimizing travel time, reducing fuel consumption, and minimizing air pollution. Your task is to apply a Multi-Objective Evolutionary Algorithm (MOEA) to optimize traffic management strategies for selected New York City (NYC) areas. The goal is to minimize conflicting objectives, Total Travel Time (TTT) and Fuel Consumption (FC), using real-world traffic data from NYC Open Data.

Optimization Objectives:

1. Minimize Total Travel Time (TTT):
    - Reduce the overall travel time for all vehicles within the network.
2. Minimize Fuel Consumption (FC):
    - Reduce the total fuel consumption across the network, influenced by factors such as speed, acceleration, and traffic congestion.

Data Source:

The necessary data for this assignment can be accessed from the NYC Open Data portal:

- NYC Traffic Volume Counts (2014-2022): NYC Open Data - Traffic Volume (https://data.cityofnewyork.us/Transportation/Traffic-Volume-Counts/btm5-ppia/about_data)
- Traffic Speed Data: NYC Open Data - Traffic Speed (https://data.cityofnewyork.us/Transportation/DOT-Traffic-Speeds-NBE/i4gi-tjb9/about_data)
- Fuel Consumption Data: Estimated using empirical models based on traffic speed and volume

Problem Statement:

You will focus on optimizing traffic management for the three to five road segments in New York City. For example:

1. 5th Ave between 42nd St and 47th St (Manhattan)
2. Atlantic Ave between Flatbush Ave and Bedford Ave (Brooklyn)
3. Queens Blvd between Union Tpke and Yellowstone Blvd (Queens)
4. Grand Concourse between E 161st St and E 170th St (Bronx)
5. Victory Blvd between Richmond Ave and Clove Rd (Staten Island)

The traffic management strategy involves controlling traffic signal timings (green, yellow, and red light durations) and setting speed limits on these segments. Your task is to develop an MOEA (discussed in lectures) that optimizes these parameters to achieve the best trade-off between minimizing TTT and FC.

Tasks:

1. Data Exploration:
    - Download and explore the NYC traffic volume and speed data for the selected road segments.
    - Identify and preprocess relevant data points, such as peak-hour traffic volumes, average speeds, and any available environmental indicators.
2. Fuel Consumption Calculation:
    - Define the Fuel Consumption Model:
        - Fuel consumption typically depends on vehicle speed. Any empirical model can estimate fuel consumption per vehicle based on average speed.
        - A common empirical model is:

          Fuel Consumption per vehicle = $a \times V + b \times \frac{1}{V} + c$

          Where:
            - V is the average speed (in mph).
            - a, b, and c are empirical constants that you will define based on literature or assumptions.
            - For example, you might assume:
                - a=0.01 (fuel consumption increases linearly with speed).
                - b=2 (fuel consumption increases at low speeds due to inefficiencies like idling).
                - c=0.1 (base fuel consumption that doesn't depend on speed).
    - Calculate Total Fuel Consumption:
        - For each road segment and time interval, calculate the total fuel consumption using the following formula:

          Total Fuel Consumption (FC) =

          $$\sum_{i=i}^{n} \left(\text{Volume}_i \times \left(a \times V_i + b \times \frac{1}{V_i} + c \right) \times \text{Segment Length}_i \right)$$

          Where:
            - n is the number of time intervals.
            - Volume_i is the vehicle count in interval i from the Traffic Volume dataset.
            - V_i is the average speed in interval $_i$ from the Traffic Speed dataset.
            - Segment Length_i is the length of the road segment.
3. Formulate the Optimization Problem:
    - Decision Variables: Determine which variables (e.g., signal timings, speed limits) will be controlled in your optimization strategy.
    - Objectives: Define the objectives (TTT, FC) your MOEA will optimize.
    - Constraints: Consider any practical constraints (e.g., total signal cycle time, minimum and maximum speed limits).
4. Implement the MOEA:
    - Design an initial population of potential solutions (chromosomes), each representing a different traffic management strategy.

    - Define the genetic operators (e.g., crossover, mutation) used to evolve the population.
    - Implement elitism.
    - Evaluate the performance of each solution based on the defined objectives.
5. Analysis and Results:
    - Run the MOEA to generate a set of Pareto-optimal solutions.
    - Analyze the trade-offs between the different objectives by examining the Pareto front.
    - Experiment with different configurations and strategies within the MOEA to understand how different trade-offs can be achieved.
    - Discuss the implications of your findings for traffic management in NYC.

Submission Requirements:

1. Code Implementation: Submit the source code used to implement the MOEA. Include a README file with instructions for running the code and explanations of the approach used.
2. Report: A report explaining:
    - The data extraction and preprocessing steps.
    - Your formulation of the optimization problem, including the decision variables, objectives, and constraints.
    - Discuss your implemented chromosome, population, genetic operators, and elitism strategy.
    - The results, including a discussion of the Pareto front and trade-offs between objectives.
    - Discussion of different configurations (parameter values) and strategies within the MOEA to understand how different trade-offs can be achieved.
    - Recommendations for traffic management based on your findings.
3. Visualization: Provide visualizations of the Pareto front and any other relevant data (e.g., traffic patterns, speed distributions).

Evaluation Criteria:

- Problem Formulation: How well you define and justify your decision variables, objectives, and constraints.
- Implementation: How efficiently you implement the MOEA (chromosome, population, genetic operators, and elitism strategy).
- Heuristic: Implementing any heuristic is not mandatory but an extra scope to grading.
- Optimization Results: The quality of the Pareto front and the exploration of trade-offs between objectives.
- Report Quality: Clarity, depth of analysis, and quality of recommendations.
- Code Quality: Organization, readability, and documentation.

Problem 2: Comparative Analysis of Evolutionary Programming and Evolutionary Strategies for Portfolio Optimization

Overview

This assignment focuses on optimizing a financial portfolio with the single objective of maximizing expected return. You will implement and compare different versions of Evolutionary Programming (EP) and Evolutionary Strategies (ES), including the $(\mu + \lambda)$ and (μ, λ) strategies, to determine which approach yields the best portfolio allocation.

Objectives

- Implement and compare different versions of EP and ES algorithms.
- Optimize a portfolio to maximize expected return.
- Analyze the performance of various algorithmic strategies, including $(\mu + \lambda)$ and (μ, λ) ES.

Data Requirements

Data Source: Yahoo Finance
Data Type: Historical Daily Stock Prices
Data Period: January 1, 2018 to December 31, 2022
Description: You will use historical daily stock price data for 20 selected stocks to compute monthly returns, which will be used in optimization.

Steps to Obtain Data:

1. Download Dataset: Go to Yahoo Finance, search for each of the 20 selected stocks, and download the historical daily stock price data in CSV format.
2. Data Columns: Ensure each CSV file contains at least the following columns: Date, Open, High, Low, Close, Volume.

Example Stocks:

- Apple Inc. (AAPL)
- Microsoft Corp. (MSFT)
- Alphabet Inc. (GOOGL)
- Amazon.com Inc. (AMZN)
- Meta Platforms Inc. (META)
- [Additional stocks as needed]

Consistency Note: To ensure a fair comparison, all students must use data from January 1, 2018, to December 31, 2022.

Detailed Instructions

Data Preprocessing

1. Download Dataset: Using the specified date range, obtain historical daily stock prices for 20 selected stocks from Yahoo Finance.

2. Calculate Monthly Returns: Convert daily stock prices into monthly returns for each stock.
3. Covariance Matrix:
    - Calculate the covariance matrix from the monthly returns. This matrix will help assess the portfolio's risk by measuring how the returns of different stocks move relative to each other.
    - **Purpose in Optimization:** While the primary goal is return maximization, understanding the covariance between assets is crucial for risk management. Students are encouraged to discuss the potential impact of risk in their analysis, even though the focus remains on return.

Algorithm Implementation

1. Version 1: Basic Evolutionary Programming (EP)
    - Implement a basic version of EP to optimize the portfolio weights, focusing on maximizing the expected return.
2. Version 2: Advanced Evolutionary Programming (EP)
    - Implement an advanced version of EP, including self-adaptive mutation strategies, different selection mechanisms, or elitism.
3. Version 3: Basic Evolutionary Strategies (ES)
    - Implement a basic version of ES to optimize the portfolio weights.
4. Version 4: Advanced Evolutionary Strategies (ES)
    - Implement an advanced version of ES, incorporating techniques like self-adaptive mutation rates or more sophisticated recombination methods.
5. Version 5: $(\mu + \lambda)$ Evolutionary Strategies
    - Implement the $(\mu + \lambda)$ version of ES, in which the next generation is selected from the union of the parent and offspring populations.
6. Version 6: (μ, λ) Evolutionary Strategies
    - Implement the (μ, λ) version of ES, where the next generation is selected only from the offspring population, introducing greater selection pressure.

Experimental Setup

1. Parameter Setup: Define and document the parameters for each algorithm, including population size (μ), offspring size (λ), mutation rates, and the number of generations.
2. Run Algorithms: Execute each version of EP and ES multiple times to ensure robust results.
3. Collect and Compare Results: Gather data on the optimized return for each algorithm version. Compare the performance in terms of final portfolio return, convergence speed, and algorithm stability.

Data Collection and Analysis

1. Performance Metrics:
    - Return: Measure the expected return of the optimized portfolio.
    - Convergence: Track how quickly each algorithm converges to an optimal or near-optimal solution.

- Stability: Assess the consistency of the results across multiple runs.

2. Visualization:
   - Create graphs comparing the performance of the different algorithm versions, including return, convergence speed, and stability.
   - Use tables to summarize key findings from the comparisons.
   -

Report Writing

1. Structure:
   - Introduction: Outline the problem and the objectives of the assignment.
   - Methodology: Describe the different versions of EP and ES implemented, including the $(\mu + \lambda)$ and (μ, λ) strategies.
   - Results: Present the results of the optimization, including visualizations.
   - Discussion: Analyze and compare the performance of the different algorithms.
   - Conclusion: Summarize the findings and suggest potential improvements or future work.
2. Content:
   - Discuss the impact of algorithmic variations on the optimization results.
   - Provide detailed comparisons between the basic and advanced versions of EP and ES, as well as the $(\mu + \lambda)$ and (μ, λ) strategies.

Deliverables

1. Code Implementation:
   - Submit well-documented code for each version of EP and ES, including $(\mu + \lambda)$ and (μ, λ) strategies.
   - Include a README file with instructions for running the code and explanations of the approach used.
2. Experimental Data:
   - Provide raw data files (CSV) and the results of the optimization runs for each algorithm version.
3. Final Report:
   - A report that includes the following, focusing on the requirements mentioned earlier:
     - Introduction and methodology
     - Results and analysis of the different algorithm versions
     - Visualizations and discussion of findings
     - Conclusions and recommendations

Evaluation Criteria

- Correctness and Completeness of Implementation
- Comparison and Analysis of Algorithm Variations
- Depth and Insight of Analysis
- Quality and Clarity of the Report
- Code Quality and Documentation

Problem 3: Solving the Vehicle Routing Problem with Time Windows (VRPTW) Using Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO)

Overview

This exercise focuses on optimizing the delivery routes for a fleet of vehicles using two nature-inspired optimization algorithms: Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). The objective is to find the most efficient routes for a set of vehicles, ensuring that all customers receive their deliveries within specified time windows. You will implement both ACO and PSO to solve the Vehicle Routing Problem with Time Windows (VRPTW) and compare the effectiveness of these algorithms.

Problem Description

The VRPTW is a variant of the Vehicle Routing Problem (VRP), which is crucial in logistics, transportation, and supply chain management. In VRPTW, a fleet of vehicles must deliver goods to multiple customers. Each customer has a specific time window during which the delivery must occur. The challenge is to design routes that minimize the total travel distance while ensuring all deliveries meet their respective time constraints.

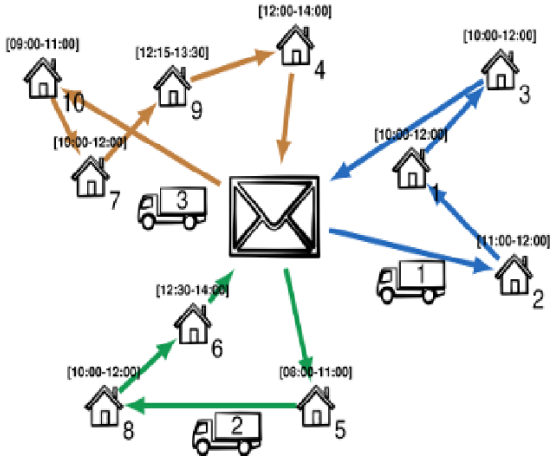

Fig: Vehicle Routing Problem with Time Windows (VRPTW)

The primary objective of the VRPTW is to minimize the total distance travelled by all vehicles while ensuring that:

1. Each customer is visited exactly once by one vehicle.
2. Deliveries occur within the specified time windows.
3. The total demand on any route does not exceed the vehicle's capacity.

Key Components

- Depot: The starting and ending point for all vehicles.
- Customers: Each customer has a location, a demand for goods, a time window for receiving the delivery, and a service time.
- Vehicles: Each vehicle has a limited capacity and must return to the depot after completing its route.

Objectives of This Exercise:

1. Implement Ant Colony Optimization (ACO) for VRPTW:
    - Develop an ACO algorithm tailored to the VRPTW.
    - Optimize the routes to minimize total travel distance while respecting the time windows.
    - Compare the results of ACO with PSO.
2. Implement Particle Swarm Optimization (PSO) for VRPTW:
    - Adapt PSO to handle the discrete nature of VRPTW.
    - Focus on finding the optimal sequence of customer visits that minimizes distance and adheres to time constraints.
    - Analyze the performance of PSO in comparison with ACO.
3. Comparative Analysis:
    - Evaluate the performance of ACO and PSO in solving VRPTW.
    - Metrics for comparison include total distance, adherence to time windows, convergence behavior, and computational efficiency.

Data Requirements

- Data Source: Solomon's VRPTW Benchmark Problems (https://www.sintef.no/projectweb/top/vrptw/solomon-benchmark/)
- Example Dataset: C101.txt (100 customers, 25 vehicles)

Dataset Details:

- Location Data: Coordinates of the depot and customers.
- Demand: The amount of goods each customer requires.
- Service Time: The time required to deliver goods at each customer's location.
- Time Windows: The earliest and latest times service can begin at each customer's location.
- Vehicle Capacity: The maximum load that each vehicle can carry.

Steps to Obtain Data:

1. Download Dataset:
    - Access the Solomon Benchmark website and download the dataset (for example, the C101.txt dataset).
    - The dataset includes customer locations, time windows, demands, and service times, along with vehicle capacity and depot details.
2. Data Preprocessing:
    - Parse the dataset to extract the necessary details for algorithm implementation.
    - Ensure that all data is correctly formatted for input into the ACO and PSO algorithms.

Detailed Instructions

1. Problem Formulation:

- Define the VRPTW for the chosen dataset, ensuring clear identification of constraints (e.g., time windows, vehicle capacity).
- Formulate the objective function that minimizes total travel distance while penalizing violations of time windows.

2. Algorithm Implementation:
    - Implement both ACO and PSO to solve the VRPTW. Focus on ensuring that each algorithm respects the problem's constraints.
    - Incorporate heuristic and penalty mechanisms as needed to handle time windows effectively.
3. Performance Evaluation:
    - Run each algorithm on the C101.txt dataset and document the results.
    - Compare the total distance, number of time window violations, and the time taken by each algorithm to converge to a solution.
4. Report Writing:
    - Methodology: Describe the problem formulation and the implementation details for ACO and PSO.
    - Results: Present the outcomes of the algorithms, including route visualizations and performance metrics.
    - Discussion: Analyze the strengths and weaknesses of ACO and PSO in solving VRPTW.
    - Conclusion: Summarize the findings and suggest potential improvements or future work.

Deliverables

1. Code Implementation:
    - Submit the code for ACO and PSO, ensuring it is well-documented and organized.
    - Include a README file explaining how to run the code and any dependencies.
2. Final Report:
    - A comprehensive report contains the points mentioned above as well as visualizations of the routes and any charts or tables used for comparison.

Evaluation Criteria

1. Correctness and Completeness of Implementation: Ensure that both ACO and PSO are correctly implemented and respect the problem constraints.
2. Effectiveness of Optimization: Assess how well each algorithm minimizes the total distance and adheres to time windows.
3. Comparative Analysis: The depth and insight of comparing ACO and PSO.
4. Clarity of Documentation: The quality and clarity of the report, including the explanation of methods and presentation of results.

Problem 4: Solving a Real-World Problem Using Reinforcement Learning

Overview

This lab exercise aims to apply reinforcement learning techniques to solve a real-world problem. Students will use a publicly available dataset to train an RL agent, evaluate its performance, and optimize it to achieve the best possible outcome.

Problem Statement:

Autonomous Taxi Navigation

You will develop an autonomous taxi driver who picks up passengers and drops them off at their destinations in the shortest possible time. The environment is a grid-based city, and the taxi must navigate to specific locations while avoiding obstacles, optimizing routes, and maximizing the efficiency of passenger pickups and drop-offs.

Dataset:

The exercise will utilize the **Taxi-v3** environment available in the OpenAI Gym repository. This environment simulates a simplified grid world where an agent must pick up and drop off passengers at the correct locations while avoiding walls and other obstacles.

- **Link to Dataset/Environment:** Taxi-v3 on OpenAI Gym

Environment Description:

- Grid Size: 5x5 grid.
- Taxi: The agent (taxi) starts at a random location.
- Passenger: A passenger is located at one of the grid cells and has a target destination.
- Actions: The taxi can move North, South, East, West, Pick-up, or Drop-off.
- Rewards: The taxi receives a reward for successfully dropping off the passenger and a penalty for every wrong action or delay.

Tasks:

1. Understanding the Environment:
    - Explore the Taxi-v3 environment.
    - Understand the state space, action space, and reward system.
    - Visualize the grid and how the taxi moves within it.
2. Setting Up the RL Agent:
    - Implement a basic Q-learning algorithm.
    - Initialize the Q-table and implement the learning loop.
3. Training the RL Agent:
    - Train the RL agent on the Taxi-v3 environment.
    - Tune hyperparameters such as learning rate, discount factor, and exploration strategy (ε-greedy).
4. Evaluation:
    - Evaluate the performance of the trained RL agent.
    - Plot the cumulative rewards over time.

- Compare the performance with a random policy and a heuristic-based policy.

5. Optimization:
    - Experiment with different RL algorithms like SARSA, Deep Q-Networks (DQN), or policy gradient methods.
    - Optimize the agent's performance by tweaking the algorithm or adjusting the hyperparameters.
6. Reporting:
    - Document the approach, challenges faced, and the results.
    - Discuss potential improvements or alternative approaches to solving the problem.

Deliverables:

- Code: A Python notebook or script containing the implemented RL agent, training process, and evaluation. Also, Include a README file with instructions for running the code and explanations of the approach used.
- Report: A brief report describing the problem, the approach taken, the results obtained, and conclusions drawn.

Tools and Libraries:

- Python (3.x)
- OpenAI Gym (for the environment)
- NumPy (for numerical computations)
- Matplotlib (for plotting results)
- TensorFlow/PyTorch (optional for more advanced RL algorithms like DQN)

Evaluation Criteria

- Correctness and Completeness of Implementation
- Comparison and Analysis of Algorithm Variations
- Depth and Insight of Analysis
- Quality and Clarity of the Report
- Code Quality and Documentation