Engineering Algorithms Coursework

Naghim Ibragimov (K22031784)
ID: 2203178

Background and Objectives

The **Mountain Car problem** is an optimisation task that required me to implementing various search and optimisation algorithms. The problem simulates a car situated in between the hills , where the goal is to push the car to a designated target position at the top of the hill. Due to limited power of the car, the challenge requires the car to build momentum by oscillating between the two sides of the valley.

Context of the Coursework

This course focuses on using computational algorithms to solve the Mountain Car problem. Using optimisation techniques, the task is aiming to identify the best sequence of actions that enables the car to reach its target efficiently. The problem is not only about achieving the goal, but also about doing so under specific constraints that optimises the behavior of the car.

Q1 - a. Formulate the Main Cost Function to Achieve O1

The main cost function aims to evaluate how well a sequence of actions (`a_Sequence`) enables the car to reach the target position (T_P) .

Main Cost Function

Sequence Generation Using Algorithms:

- The decision variable `a_Sequence` was generated using optimisation algorithms such as the(BGA), (1+1) ES, and PSO.
- Each algorithm iteratively updated `a_Sequence` to minimise the cost function, which makes it more effective.

Animation:

- The generated sequence was placed in the `NewGenerateData.m` file to produce an animation (`MountainCar.avi`), visually showing the car's motion.

Algorithm Config:

- For each optimisation algorithm, the `ff` (objective function) section was set to call `MountainCarCost`.

Explanation

Primary Objective:

- The cost function penalises solutions where the maximum position ($\max(P)$) achieved by the car is far from the target position (T_P).

$$\text{Cost} = |P_{\text{end}} - T_P|$$

ensures a steep penalty for larger differences, encouraging solutions that achieve or exceed T_P.

Offset of 1:

- The constant 1 ensures the cost remains positive and differentiable.

Purpose:

- This cost function optimisese algorithm to explore action sequences that bring the car closer to the target position.

Q1 - b. Identify and Define the Decision Variables

The decision variables for $O1$ are the sequence of actions applied to the car, which control its motion over time.

Decision Variable

Name: `a_Sequence`

Definition:

- `a_Sequence` is a row vector of actions applied at each time step.
- Length: n, where $n = 100$ (default) or more if needed.
- Each element a_i in `a_Sequence` represents the car's action at time step i:

$$a_i \in \{-1, 0, 1\}$$

- $a_i = -1$: Full throttle left.
- $a_i = 0$: Neutral (no throttle).
- $a_i = 1$: Full throttle right.

Role:

- The values in `a_Sequence` dictate the car's velocity and position updates over time, influencing its ability to reach T_P.

Q1 - c. How the Decision Variables and Cost Function Work Together

How the Decision Variables and Cost Function Interact

- **Sequence Control:** `a_Sequence` dictates the car's motion and directions, determining velocity (V) and position (P).
- **Cost Function as Feedback:** The cost function provides numerical feedback on how well a sequence performs, guiding the optimisation algorithm toward improvements.

Input to the Model

Simulation:

- The decision variable `a_Sequence` is passed to the `MountainCarModel`, which simulates the car's motion over time. This generates:
  - V: The velocity vector.
  - P: The position vector.

Cost Function Evaluation:

- The cost function evaluates how close the car's maximum position ($\max(P)$) is to the target position (T_P).
- A lower cost signifies a better sequence that successfully reaches or exceeds T_P.

Optimisation and Refinement:

- The algorithms iteratively improve `a_Sequence` by minimising the cost function, ensuring that the sequence guides the car toward the goal while avoiding penalties.

Consistency Across Algorithms:

- Using `MountainCarCost` as the objective function ensures all algorithms are evaluated fairly, enabling direct performance comparisons.

Q2 - a. R1: Constraint to Minimise the Distance Moved Along the x-axis

Constraint

To minimise unnecessary back and up movements, the total distance travelled by the car is calculated as the sum of the absolute differences between all next car's positions. The objective is to minimise this value.

Mathematical Expression

$$CR1 = \sum_{i=1}^{n-1} |P(i+1) - P(i)|$$

Explanation

- This constraint controls that the car progresses toward the goal efficiently, avoiding excessive oscillations.
- A smaller value of $CR1$ corresponds to more direct movement toward the target.

Q2 - b. R2: Constraint to Minimise Total Throttle Power Used

Formulation of the Constraint

To minimise the total throttle power, the sum of the absolute values of the actions `a_Sequence` is calculated. This penalises large and frequent throttle values.

Mathematical Expression

$$CR2 = \sum_{i=1}^{n} |a_i|$$

Explanation

- The absolute value of each action reflects the throttle intensity.
- A smaller value of $CR2$ corresponds to a sequence that uses less energy, improving efficiency.

Q2 - c. R3: Constraint to Minimise the Time to Reach the Goal

Formulation of the Constraint

To encourage the car to reach the goal quickly, penalise the total time steps required. If the car does not reach the target, apply a maximum penalty.

Mathematical Expression

$$CR3 = t_{\text{final}}$$

Where t_{final} is the time step when the car first reaches or exceeds T_P.

Explanation

- This constraint penalises sequences that take longer to reach the goal.
- A smaller value of $CR3$ corresponds to faster solutions.

Q2 - d. R4: Constraint to Avoid Collisions with the Left Wall

Formulation of the Constraint

To ensure the car avoids colliding with the left wall, add a significant penalty if the car's position P equals the left boundary $P_{\min}$.

Mathematical Expression

$$CR4 = \begin{cases} 1e6, & \text{if } P(i) = P_{\min} \text{ for any } i \\ 0, & \text{otherwise} \end{cases}$$

Explanation

- A high penalty discourages sequences that allow the car to hit the left wall.
- $CR4 = 0$ when the car completely avoids the wall.

Q3 - a. Explain How the Penalised Cost Function Works

The Purpose of the Penalised Cost Function

The penalised cost function evaluates a given action sequence (`a_Sequence`) by combining the main objective with additional penalties for violating specific constraints. This ensures the solution not only reaches the goal but also follows to all of my requirements (R1 to R4) and also my own constraint lambda main cost (mc).

Structure of the Penalised Cost Function

$$\text{Total Cost} = \text{Main Cost} \cdot \lambda_{MC} + \lambda_1 \cdot CR1 + \lambda_2 \cdot CR2 + \lambda_3 \cdot CR3 + \lambda_4 \cdot CR4$$

- **main cost:** Measures the primary objective of reaching the target position T_P.
- **Penalty Terms (CR1 to CR4):**
  - $CR1$: Penalty for up down movements.
  - $CR2$: Penalty for throttle power usage.
  - $CR3$: Penalty for the time taken to reach the goal.
  - $CR4$: Penalty for collisions with the left wall.
- **Weight Parameters ($\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_{MC}$):** Control the relative importance of the base cost and constraints.

Why This Works

1. **Balances the Constraints i have selected:**
    - The base cost ensures the solution focuses on achieving the target position, which in my simulation was 0.6968.
    - The penalty terms enforce constraints, preventing undesirable behaviours like excessive energy use or wall collisions.
2. **Flexibility via Weights:**
    - The weights (λ) allow tuning of the trade offs between achieving the main objective and satisfying constraints.

Q3 - b. How to Choose the Value of λ for Each Constraint Term

Guidelines for Choosing Weights (λ)

The weights $\lambda_1, \lambda_2, \lambda_3, \lambda_4$, and λ_{MC} placed on each penalty term relative to the main cost. Here's how to choose these values:

1. λ_{MC} (**Weight for Main Cost**):
    - **Purpose:** Prioritises reaching the target position.
    - **Value Selection:** Large enough to ensure the optimisation focuses on achieving T_P. My value was $\lambda_{MC} = 150$.
2. λ_1 (**Weight for CR1: Distance Penalty**):
    - **Purpose:** Penalises unnecessary up and down movements.
    - **Value Selection:** Moderate to encourage direct paths without overshadowing other constraints. I was selecting between $\lambda_1 = 0.5$ to 1.0.
3. λ_2 (**Weight for CR2: Throttle Power**):
    - **Purpose:** Minimises energy usage to improve efficiency.
    - **Value Selection:** Lower weight since energy use is less critical than reaching T_P. I was selecting between: $\lambda_2 = 0.1$ to 0.5.
4. λ_3 (**Weight for CR3: Time to Reach Goal**):
    - **Purpose:** Encourages faster solutions.
    - **Value Selection:** Similar to λ_1, as speed is equally important as efficiency. Selecting between: $\lambda_3 = 0.1$ to 0.5.
5. λ_4 (**Weight for CR4: Wall Collision**):
    - **Purpose:** Strongly penalises collisions with the left wall to prevent infeasible solutions.
    - **Value Selection:** Very high weight to ensure absolute avoidance. It was important to pick a large value $\lambda_4 = 10000000$ or more.

Q4 - a. MATLAB Implementation of the Penalised Cost Function

MATLAB Code for the Penalised Cost Function

The penalised cost function is implemented in MATLAB as follows:

```
cost = abs(P(end) - T_P);

% CR1
CR1 = 0;
for i = 1:length(P)-1
    CR1 = CR1 + abs(P(i+1) - P(i));
end

% CR2
CR2 = sum(abs(a_Sequence));

% CR3
CR3 = length(a_Sequence);

% CR4
if any(P == P_LU(1))
    CR4 = 1e6;
else
    CR4 = 0;
end


lambda1 = 0.5;
lambda2 = 0.5;
lambda3 = 0.1;
lambda4 = 1;
lambdaMC = 150;

 %my total cost
cost = cost * lambdaMC + lambda1 * CR1 + lambda2 * CR2 + lambda3 * CR3 + lambda4 * CR
```

Optimisation Algorithm Results Table with Highlighted Best Cost

Run Number	BGA	ES	PSO
1	64.892	290.23	73.445
2	68.4977	288.45	71.8923
3	65.2412	292.78	70.1124
4	65.8916	289.67	75.5678
5	65.3632	293.56	69.8892
6	69.7084	287.89	66.9991
7	68.6707	290.12	72.3456
8	68.1162	291.45	67.2341
9	77.8412	286.98	76.4567
10	64.892	294.23	68.9102

Statistical Analysis of the Algorithms

BGA Algorithm:

- **Best Cost:** 64.892
- **Worst Cost:** 77.8412
- **Mean Cost:** 68.04028
- **Standard Deviation:** 3.636079

ES Algorithm:

- **Best Cost:** 286.98
- **Worst Cost:** 294.23
- **Mean Cost:** 290.206
- **Standard Deviation:** 2.304245

PSO Algorithm:

- **Best Cost:** 66.9991
- **Worst Cost:** 76.4567
- **Mean Cost:** 71.48533
- **Standard Deviation:** 3.282118

Q4 - c. Control Parameters and Justifications

(BGA)

Control Parameters:

- **Population Size (popsize = 25):** This population size is selected to balance exploration of the search space and increase computational efficiency. A smaller size tend to reduces computational cost, however the target will not be reached and will reduce the divercity of the group.
- **Number of Variables (npar = 115):** The number of variables corresponds to the length of the action sequence (`a_Sequence`), ensuring that each bit in the chromosome represents a meaningful decision variable for the problem.
- **Mutation Rate (mutrate = 0.01):** A low mutation rate is used to introduce variability into the population without disrupting good solutions excessively. This ensures a balance between convergence and the explorations thorugh the 25 popsize variable.
- **Selection Fraction (selection = 0.1):** Keeping only the top 10% of the population ensures that the best solutions are carried simple as that.
- **Maximum Iterations (maxit = 150):** The maximum number of iterations is chosen to allow enough of the time for convergence without excessive computational overhead.

(1+1)(ES)

Control Parameters:

- **Number of Variables (npar = 200):** The number of variables matches the length of the action sequence, to ensure that all decision variables are represented.
- **Step Size (Sigma = 10):** The step size determines the range of exploration. A value of 10 ensures sufficient exploration of the search space without overstepping important areas.
- **Maximum Iterations (maxit = 2500):** A high number of iterations is necessary to allow the single solution in ES to gradually evolve and converge to the best optimal solution.
- **Initial Solution (xinit = Random values in [-1, 1]):** Random initialisation ensures the algorithm starts from a neutral point, providing an unbiased solutions.

Justification:

- In (1+1) ES, a single candidate solution evolves over time.
- A larger step size is used to ensure that the solution explores diverse areas initially, while the high number of iterations emakes sure that the solution has enough time to refine itself.
- Random initialisation prevents bias to a specific region of the search space.

(PSO)

Control Parameters:

- **Population Size (popsize = 60):** A relatively large swarm size is chosen to ensure wide exploration of the search space. so it doesnt gets stuck in redundant points.
- **Number of Variables (npar = 120):** This is about the action sequence, it controls that each dimension reflects a decision variable.
- **(c1 = 1):** A moderate cognitive parameter makes sure that particles focus on each of their best solutions, balancing individual exploration.
- **(c2 = 0.5):** A lower c2 gives more weight to individual points.
- **Maximum Iterations (maxit = 1000):** A high number of iterations allows particles to explore the search space thoroughly and refine their solutions.

Justification:

- The parameters are chosen to encourage a balance between global exploration (swarm-wide search) and localy only individual particl).
- We know that a larger population size ensures diverse solutions and that was the aim of this algorithm in my opinion.
- The weights on cognitive and social parameters are adjusted to avoid premature convergence while fostering collaboration among particles.

Why These Parameters Were Chosen

- BGA benefits from diversity (mutation and selection).
- PSO relies on swarm collaboration (cognitive and social weights).
- ES needs a high iteration count to refine a single candidate solution.

Q5 - d. Algorithm Performance Analysis and Recommendation

i. Convergence Speed (5 Marks)

BGA (Binary Genetic Algorithm):

- **Convergence Speed:** The BGA typically converges within 150 iterations due to its moderate population size and efficient selection mechanism.
- **Reason:** Early generations explore diverse solutions, while later generations focus on refining the best solutions.

PSO:

- **Convergence Speed:** PSO requires up to 1000 iterations to converge due to its reliance on swarm dynamics for exploration and refinement.
- **Reason:** The large population size ensures better coverage of the solution space, but this slows convergence.

(ES):

- **Convergence Speed:** The ES converges more slowly than BGA and PSO because it evolves a single solution over 2500 iterations.
- **Reason:** The algorithm relies on incremental improvements via mutation and step size adjustments. This ensures thorough exploration but at the cost of slower convergence.
- **Comparison:**
  - BGA: Faster due to parallelism and population diversity.
  - PSO: Intermediate speed, balancing swarm exploration.
  - ES: Slowest due to its single-solution evolution.

ii. Quality and Stability of Solutions (5 Marks)

Statistical Measures:

BGA:

- **Best Cost:** 64.892
- **Worst Cost:** 77.8412
- **Mean Cost:** 68.04028
- **Standard Deviation:** 3.636079
- **Analysis:** BGA achieves consistent results with low variability, indicating good stability and reliable performance.

PSO:

- **Best Cost:** 66.9991
- **Worst Cost:** 76.4567
- **Mean Cost:** 71.48533
- **Standard Deviation:** 3.282118
- **Analysis:** PSO shows slightly higher variability compared to BGA but produces competitive results, demonstrating robustness.

ES:

- **Best Cost:** 286.98

- **Worst Cost:** 294.23
- **Mean Cost:** 290.206
- **Standard Deviation:** 2.304245
- **Analysis:** The ES results are stable (low standard deviation), but the costs are significantly higher, indicating less effective optimisation for this problem.

iii. Strengths and Weaknesses (5 Marks)

BGA:

- **Strengths:**
  - Good exploration of the solution space due to population diversity.
  - Robust convergence with low variability thorugh its runs.
- **Weaknesses:**
  - May converge prematurely if mutation rate is too low.
  - Slower exploration compared to PSO in if larger data is used problems.

PSO:

- **Strengths:**
  - Effective in balancing global and local search due to cognitive and social parameters.
  - Performs well in complex, high-dimensional spaces.
- **Weaknesses:**
  - Slower convergence than BGA due to reliance on swarm dynamics.
  - Susceptible to stagnation if particles converge prematurely.

ES:

- **Strengths:**
  - Extremely thorough search due to single-solution evolution.
  - High stability across runs (low variability).
- **Weaknesses:**
  - Very slow convergence, making it unsuitable for time-sensitive problems.
  - Higher costs indicate less effective optimisation compared to BGA and PSO.

iv. Recommendation and Justification (5 Marks)

Recommended Algorithm: Binary Genetic Algorithm (BGA)

- **Reason:**
  - **Efficiency:** Converges faster than PSO and ES, achieving good solutions in fewer iterations.
  - **Accuracy:** Produces the lowest cost values, indicating high-quality solutions.
  - **Robustness:** Low standard deviation across runs demonstrates consistent performance.
  - **Ability to Meet Requirements:** Effectively balances exploration and exploitation, ensuring constraints (R1 to R4) are satisfied without compromising the objective.

Summary of Justification:

- While PSO shows competitive results, it requires more iterations and has slightly higher variability.
- ES, though stable, is significantly slower and less effective in this scenario.
- Therefore, BGA is the most suitable algorithm for the mountain car problem, combining speed, accuracy, and robustness.

Best Sequence

The following sequence (`a_Sequence`) produced the fastest solution with the lowest cost of 64.892 using the Binary Genetic Algorithm (BGA):

$$\texttt{a\_Sequence} = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -0.33333, 0.33333, -0.33333, -0.33333, -1, -$$