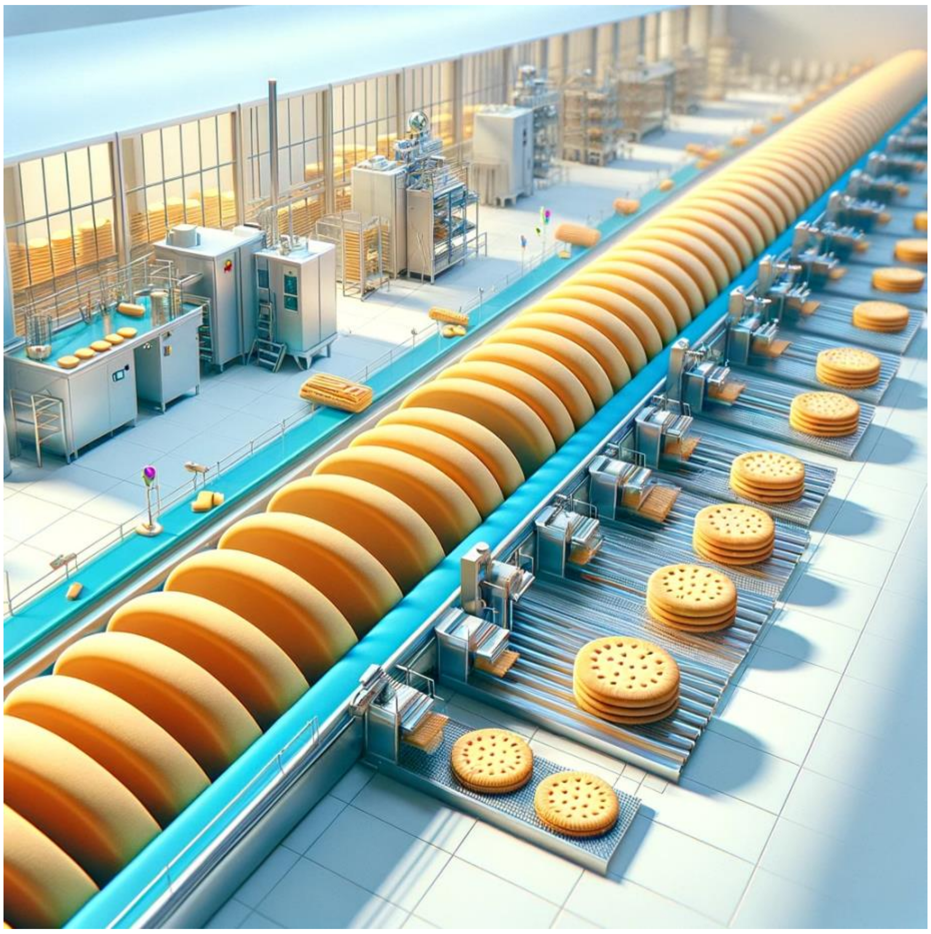
Project Report for AI Algorithms Course Project

Aurore Pistono – Lily La Morella – Louis Gauthier (DIA 1)



1. Introduction

1.1 Background

The AI Algorithms course project was an extensive task incorporating complex problem-solving using AI methodologies. The project was titled "Optimization of Biscuit Production in a Factory Setting" and revolved around maximizing output and profit by efficiently utilizing available resources, particularly focusing on a one-dimensional dough-rolling scenario with various constraints.

1.2 Purpose

The project's primary aim was to apply AI algorithms to solve real-world production problems in a simulated factory environment. It provided an opportunity to demonstrate the practical applicability of AI techniques learned during the course.

2. Problem Statement

2.1 Overview

A biscuit manufacturing factory plans to produce different types of biscuits using a single roll of dough. The task is to maximize the number of biscuits produced from this dough while considering the size, value, and defect constraints of each biscuit type.

2.2 Specific Challenges

- Handling the one-dimensional dough-rolling process with predefined length (LENGTH).
- Managing irregularities (defects) in the dough roll, each with a specific position and class.
- Each biscuit type has a unique size, value, and defect threshold.
- Biscuits must be placed without overlapping and within the defect constraints.

2.3 Problem constants

The project is defined by several constants, which are as follows:

- **Dough Length**: The total length of the dough available for biscuit production is set at 500 units.
- **Biscuit Types**: An assortment of biscuit types are available for production, each with its unique properties:
  - Biscuit 0: Length 4 units, Value 6, Defects thresholds {"a": 4, "b": 2, "c": 3}, ID 0
  - Biscuit 1: Length 8 units, Value 12, Defects thresholds {"a": 5, "b": 4, "c": 4}, ID 1
  - Biscuit 2: Length 2 units, Value 1, Defects thresholds {"a": 1, "b": 2, "c": 1}, ID 2
  - Biscuit 3: Length 5 units, Value 8, Defects thresholds {"a": 2, "b": 3, "c": 2}, ID 3
- **Defects Database**: Detailed information about defects can be found in the provided **defects.csv** file.

3. Methodology

3.1 Algorithm Selection

In our pursuit to devise an effective strategy for biscuit placement, three distinct problem-solving approaches were considered and implemented. Initially, we utilized Dynamic Programming (DP) to establish a baseline for the optimal solution. This approach was instrumental in providing us with an understanding of the best possible result in an O(n) complexity. However, it's important to note that DP was not an integral part of the course curriculum and, consequently, was not a viable option for the project. We employed DP primarily as an auxiliary tool to enhance our understanding of the problem.

Subsequently, we shifted our focus to more relevant algorithms that aligned with the course's objectives:

1. **Genetic Algorithm (GA):** This method involved the creation of a diverse population of solutions. These solutions underwent iterative improvements through a series of genetic operations: selection, crossover, and mutation. The goal was to evolve the solutions over generations, aiming to find an optimal or near-optimal arrangement of biscuits.
2. **Simulated Annealing (SA):** In addition to GA, we explored Simulated Annealing. This algorithm mimics the process of heating and slowly cooling a material to decrease defects. In the context of our problem, it involves exploring the solution space by probabilistically accepting solutions based on a temperature parameter that gradually decreases. This approach is particularly effective in avoiding local optima and striving towards a globally optimal solution.

These methodologies, particularly GA and SA, were chosen for their compatibility with the course's framework and their potential to provide robust solutions to the biscuit placement problem. The use of DP, although not directly applicable to the project, served as a valuable reference point in our algorithmic exploration.

3.2 Implementation

All approaches were implemented in Python. The corresponding code can be found in the 3 Jupyter Notebook files of the project.

3.2.1 Dynamic Programming

Concept of DP:

Dynamic Programming is a method used in computer science and mathematics to solve problems by breaking them down into simpler subproblems. It is particularly effective for optimization problems, where the goal is to find the best solution among many possibilities.

Application to Biscuit Placement:

In the context of the biscuit placement problem, we are trying to find the optimal way to arrange different types of biscuits on a dough of a certain length to maximize the overall value, while respecting certain constraints (like the maximum number of defects allowed for each biscuit type).

How It Works:

1. **Breaking Down the Problem**:
    - The dough is considered as a series of segments (or units of length).
    - For each segment, we calculate the best combination of biscuits that can be placed up to that point.
2. **Subproblems**:
    - Each subproblem involves determining the optimal value that can be obtained for a smaller length of the dough.
    - The solution to a larger segment of the dough builds upon the solutions to its smaller segments.
3. **Iterative Approach**:
    - We start from the smallest segment of the dough and work our way up to the full length.
    - At each step, we consider adding each type of biscuit to the current segment, only if it fits and meets the defect constraints.
4. **Defect Constraints**:
    - For each biscuit placement, we ensure that the number of defects does not exceed the limits for that biscuit type.
5. **Maximizing Value**:
    - For each segment, we calculate the maximum value that can be achieved either by adding a new biscuit (if it fits and meets constraints) or by retaining the previous best arrangement.
    - This involves comparing the value obtained by including or excluding each biscuit type at each segment.
6. **Optimal Solution**:
    - The Dynamic Programming process constructs a table (or an array) where each entry corresponds to the maximum value achievable for a specific segment length of the dough.
    - For example, the element at index 46 in this table represents the maximum value that can be achieved for a dough segment of exactly 46 units in length. This means it considers all possible combinations of biscuits that can fit within the first 46 units of the dough, adhering to the constraints, and selects the one with the highest total value.
    - The final entry in this table gives the optimal value for the entire length of the dough.

Outcome:

Using DP, the algorithm efficiently explores all possible arrangements of biscuits and finds the one that maximizes the total value while adhering to the defect constraints. This approach ensures that the solution is both optimal and computationally efficient, especially for larger problems where the number of possible arrangements is prohibitively large for simpler methods. The algorithm has an O(n) complexity, n being the dough length.

After running our DP implementation, we found an optimal value for the of 760. Optimal biscuit placements can be found in the solution_dp.txt files.

3.2.2 Genetic Algorithm

Concept of Genetic Algorithm:

Genetic Algorithm (GA) is a search heuristic inspired by Charles Darwin’s theory of natural evolution. It reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Application to Biscuit Placement:

The biscuit placement problem can be approached using GA by considering each potential arrangement of biscuits as an individual in a population. The fitness of each individual is measured based on the total value obtained from the arrangement of biscuits, while adhering to the defect constraints.

How It Works:

1. Initial Population:
    - The algorithm begins with a randomly generated population of biscuit arrangements. Each individual in the population represents a possible solution to the problem.
2. Fitness Function:
    - Each individual is evaluated using a fitness function that calculates the total value of the biscuit arrangement, considering the length, value, and defect constraints of each biscuit type.
3. Selection:
    - Individuals are selected for reproduction based on their fitness. Higher fitness individuals have a higher chance of being selected. This can be achieved through methods like tournament selection.
4. Crossover:

    - Selected individuals undergo crossover to produce offspring. Parts of two individuals are combined to create new offspring, representing a new arrangement of biscuits.
5. Mutation:
    - To maintain genetic diversity in the population and to explore new solutions, offspring may undergo mutations where random changes are made to the biscuit arrangement.
6. Iteration:
    - The process of selection, crossover, and mutation is repeated over several generations. With each generation, the population is expected to improve in terms of fitness.

Outcome:

The GA effectively explores a wide range of potential solutions, gradually improving the fitness of the population. After a set number of generations or when a satisfactory fitness level is achieved, the algorithm yields an optimal or near-optimal arrangement of biscuits. In this case, the GA found a solution with an optimal value of 740. The specific arrangement of biscuits for this solution can be found in the file **solution_genetic_algorithm.txt**.

3.2.3 Simulated Annealing

Concept of Simulated Annealing:

Simulated Annealing (SA) is an optimization technique inspired by the process of annealing in metallurgy. It is a probabilistic technique for approximating the global optimum of a given function.

Application to Biscuit Placement:

In the context of the biscuit placement problem, SA can be used to search for an optimal arrangement of biscuits. It explores the solution space by making random changes to the current arrangement and accepting these changes based on a probability that depends on the change in fitness and a temperature parameter.

How It Works:

1. Initial Solution:
    - The algorithm starts with a random arrangement of biscuits as the initial solution.
2. Temperature Schedule:
    - SA uses a temperature parameter that gradually decreases over time. The temperature controls the probability of accepting worse solutions as the algorithm progresses.
3. Iterative Process:

    - At each iteration, the algorithm randomly alters the current arrangement (e.g., changing the type or position of a biscuit) and calculates the new fitness.
4. Acceptance Criteria:
    - If the new arrangement has a higher fitness, it is accepted as the current solution. If the fitness is lower, it may still be accepted with a probability that depends on the difference in fitness and the current temperature.
5. Cooling:
    - The temperature is gradually decreased according to a cooling schedule, reducing the likelihood of accepting worse solutions.
6. Termination:
    - The algorithm terminates after a certain number of iterations or when the temperature falls below a threshold.

Outcome:

SA is particularly effective in avoiding local optima and exploring a broader solution space. In this implementation, SA successfully identified an arrangement with a total value of 725. This solution demonstrates the ability of SA to navigate the complex solution space of the biscuit placement problem.

4. Results and Analysis

4.1 Dynamic Programming (DP) — Best value = 760

- **Execution Time**: Very fast, has O(n) complexity
- **Quality of Solution**: Provided optimal solutions consistently.
- **Limitation**: Complexity increased linearly with the size of the problem.

4.2 Genetic Algorithm (GA) — Best value = 740

- **Execution Time**: Slower due to iterative nature and the need for a lot of defect verifications in the population.
- **Quality of Solution**: Achieved near-optimal solutions.
- **Advantage**: More adaptable to changes in problem parameters.

4.3 Simulated Annealing (SA) — Best value = 725

- **Execution Time:** Moderate execution time, balancing between exploration and exploitation phases.
- **Quality of Solution**: Good quality solutions, effectively balancing between finding a global optimum and avoiding local optima.
- **Advantage**: The algorithm's ability to probabilistically accept worse solutions at higher temperatures made it versatile in exploring a wide solution space. This characteristic was crucial in escaping local optima and moving towards a more optimal arrangement.

The genetic algorithm's and simulated annealing’s progresses were visually tracked through a fitness history graph.

4.4 Comparative Analysis

- DP was the most reliable for providing optimal solutions.
- GA offered flexibility and adaptability, making it suitable for larger and more complex scenarios. Its evolutionary nature allowed for continuous improvement and adaptation to changing constraints.
- SA was effective in avoiding local optima and exploring a broader solution space, making it a strong contender for problems with complex landscapes.

5. Conclusion

5.1 Learning Outcome

The project significantly enhanced understanding and application of AI algorithms in practical scenarios. It demonstrated how different AI techniques could be tailored to address specific challenges in industrial settings.

5.2 Reflections

The project highlighted the importance of choosing the right algorithm based on the problem's nature and constraints. It also underscored the potential of AI in optimizing production processes, a vital aspect of modern manufacturing industries.

5.3 Future Scope

Future work could explore hybrid approaches combining DP and GA or investigate other AI techniques like neural networks for more complex multidimensional problems.

6. Appendices

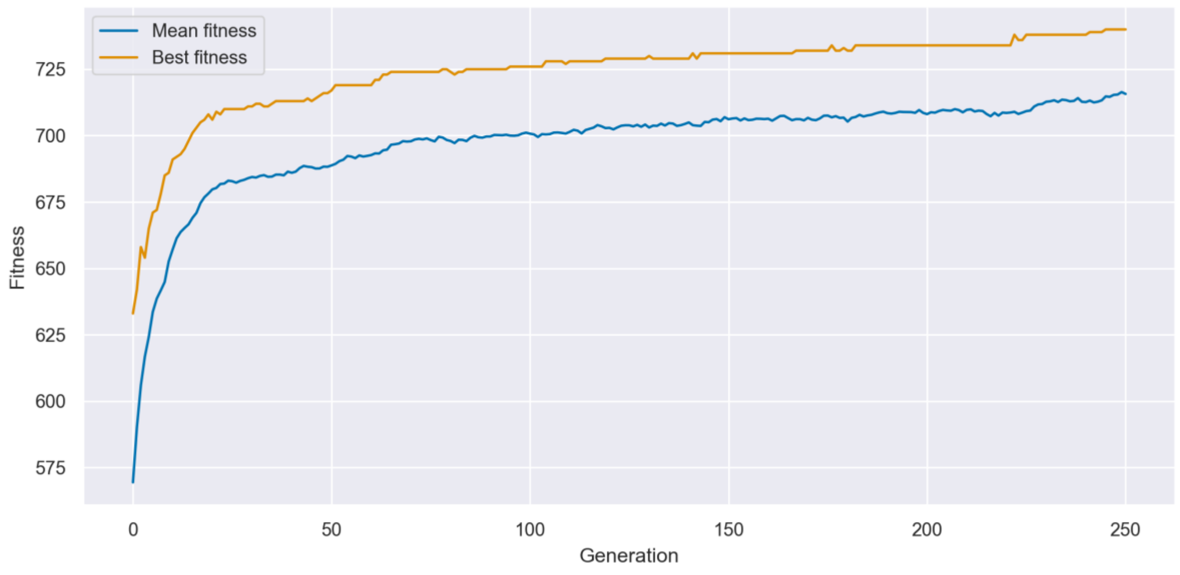
6.1 Code Implementation

The code snippets and detailed implementation for both Dynamic Programming and Genetic Algorithm approaches are included in the project notebooks. The utils.py script is also required to run the notebooks and contains code that was needed for several implementations, in order to reduce code duplication.

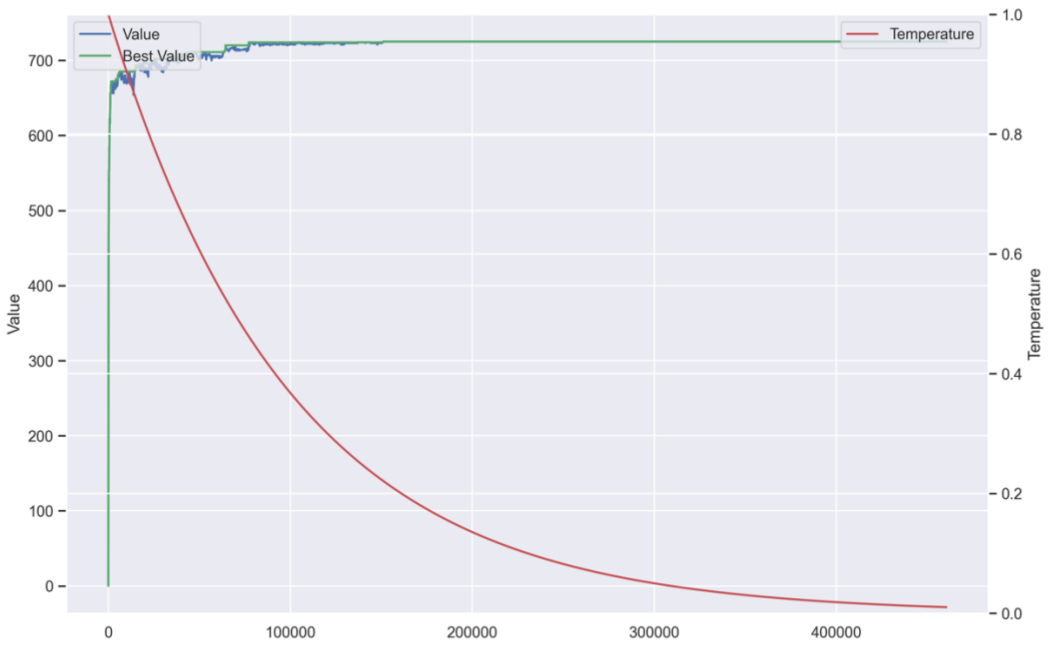
6.2 Fitness History Graphs

The fitness history graphs for the Genetic Algorithm and Simulated Annealing approaches are provided, showcasing the evolution of the algorithms over time.

Fitness history for Genetic Algorithm:



Fitness history for Simulated Annealing:



7. Acknowledgments

Special thanks to the course instructors and peers for their valuable insights and support throughout the project duration.