**IA Project : Final Report**

**Introduction**

This project embodies a rigorous challenge for aspiring Research Engineers in Data Science & Optimization aiming to join a prestigious French company. With a four-day timeline, candidates were tasked to grapple with a complex biscuit production optimization problem. The objective is to strategically allocate biscuits on a roll of dough, leveraging AI algorithms and problem-solving skills gained from the AI algorithms course.

**Problem description**

The scenario presents a biscuit manufacturing scenario for the upcoming Christmas season. The factory plans to maximize profits by creating diverse biscuit shapes and sizes from a single roll of dough. However, this endeavor comes with constraints. The roll of dough, defined by its length ('LENGTH'), introduces a one-dimensional problem. Moreover, the presence of defects, each characterized by position ('x') and class ('a', 'b' and 'c'), further complicates the optimization process.

To optimize biscuit placement, several critical considerations come into play. Each biscuit has specific attributes: size (along the same dimension as the roll), a value (representing its price), and thresholds dictating the maximum allowable defects of each class. The task demands the formulation of solutions adhering to stringent guidelines:

**Integer positions**: Biscuits must be placed at integer positions along the roll.

**No overlaps**: Overlapping placements are forbidden; a biscuit's placement excludes adjacent positions for other biscuits.

**Defect thresholds**: Each biscuit's placement must ensure defects within the interval meet or are fewer than the specified thresholds for each class.

**Roll length constraint**: The cumulative size of the assigned biscuits cannot exceed the length of the dough roll.

The value of any solution is determined by **the collective value of individual biscuits placed on the roll**.

During the technical interview, we must present our **strategies**, **solutions**, and **methodologies** based solely on the provided information and the knowledge acquired from the AI algorithms course. Success hinges on the ability to tackle this intricate problem, showcasing expertise in AI-driven optimization techniques and problem-solving acumen within stringent constraints.

**Approaches Employed**

**Greedy-Search**

An algorithm employing a greedy-search strategy was developed. It seeks to place biscuits, considering overlap and defect threshold constraints. This method aims to maximize the total value of placed biscuits.

We use firstly Greedy Search for this problem because it presents several advantages due to its **suitability for specific characteristics of the biscuit placement optimization scenario**:

Greedy Search excels in **local optimization**, making iterative decisions based on immediate, locally optimal choices. In the context of biscuit placement**, it iterates through possible positions**, selecting **the best immediate option without considering future implications extensively**.

This algorithm was relatively **simple to implement** and **computationally efficient**.

Our algorithm allows us to **choose the number of occurrences of each biscuit** we want to place on the roll, and returns the total value accordingly.

This enabled us to quickly (2 min max) check that the algorithm was working properly on small numbers of biscuits.

**Genetic Algorithm**

A genetic algorithm was implemented as a secondary approach to solve the problem iteratively.

It is inspired by the principles of natural selection and genetics. Its utilization aims to address the complex optimization problem of placing biscuits on a dough roll while adhering to various constraints.

**1. Population initialization**:

The **initialize\_population()** function **generates a diverse set of potential solutions** (individuals) represented as configurations of biscuit placements on the roll. Each solution comprises a combination of biscuits positioned randomly within the roll's constraints.

**2. Evaluation of solutions:**

The **evaluate\_solution**() function assesses the fitness of each solution. **It calculates the total value of the biscuits placed on the roll and verifies whether the solution conforms to the specified constraints**. These constraints include defect thresholds and spatial limitations, ensuring the validity of each solution.

**3. Selection of parents:**

The **select\_parents()** function identifies the most promising solutions (parents) from the population based on their evaluated fitness scores. Solutions with higher fitness values are more likely to be selected as parents for the next generation, mimicking the principle of natural selection.

**4. Crossover and Mutation:**

Crossover and mutation, two essential genetic operations, facilitate the creation of offspring solutions from selected parents. Crossover combines genetic material (biscuit placement configurations) from two parents to generate new solutions (offspring), potentially inheriting good characteristics from both parents.

**5. Iterative improvement:**

The main **genetic\_algorithm()** function iterates through generations of populations, refining solutions over multiple cycles. It involves selecting parents, generating offspring through crossover and mutation, and replacing the previous population with the new set of solutions. **This iterative process aims to improve the overall fitness and quality of solutions.**

**CSP (Constraint Satisfaction Problem)**

**Problem formulation:**

The function formulates the biscuit placement problem as a CSP using **the Problem class from the python-constraint module**. It defines variables and constraints to find optimal positions for biscuits on the roll while considering various constraints.

**Variable definition:**

Variables representing positions of biscuits on the roll are created using addVariable. Each biscuit's position variable is within the permissible range on the roll, constrained by its length.

**Constraint specification:**

Constraints are established to ensure valid biscuit placements:

No overlaps 🡪 Constraints prevent overlapping placements between biscuits. It employs the intervals\_overlap() function, ensuring that no two biscuits occupy the same space on the roll.

Defect thresholds check 🡪 Constraints validate that each biscuit's placement adheres to defect thresholds. The can\_place\_biscuit function verifies if the placement complies with maximum defect limits.

**Main problem with the CSP implémentation:**

The primary drawback of this algorithm lies in its **time and memory costs**, **significantly complicating the attainment of a final result.** The extensive computation required to explore the solution space and validate configurations imposes a substantial computational burden, leading to prolonged execution times and demanding memory requirements. This complexity often hampers the algorithm's ability to swiftly converge towards an optimal or near-optimal solution, posing challenges in efficiently obtaining a conclusive outcome within reasonable timeframes.

**Conclusion**

This project embodies a rigorous test, demanding adeptness in AI algorithms and optimization techniques to tackle a complex biscuit production optimization problem. The successful navigation of stringent constraints, demonstrated through Greedy-Search, Genetic Algorithms, and Constraint Satisfaction Problem approaches, showcases our prowess in AI-driven problem-solving. Despite the CSP's computational overheads, these strategies underscore the ability to maneuver intricate challenges, marking our adeptness in AI application for real-world problem-solving.