**Projet AI Algorithm**

Member of the project : Henri Serano – Eloi Seidlitz

Summary

[Introduction 2](#_Toc153211973)

[Purpose and Context of the Code 2](#_Toc153211974)

[Main Objective 2](#_Toc153211975)

[Overview of the Code Structure 2](#_Toc153211976)

[Detailed Analysis of Key Components 2](#_Toc153211977)

[Classes and Their Roles 2](#_Toc153211978)

[Hill Climbing Search 4](#_Toc153211979)

[Description of the Hill Climbing Search Algorithm 4](#_Toc153211980)

[Iteration and Improvement of Solutions 4](#_Toc153211981)

[Constraint-Based Search 5](#_Toc153211982)

[Approach of Constraint-Based Search 5](#_Toc153211983)

[Differences from Hill Climbing Search 6](#_Toc153211984)

[Solution Validation 6](#_Toc153211985)

[Outline of the Solution Validation Process 6](#_Toc153211986)

[Visualization and Progress Tracking 7](#_Toc153211987)

[Conclusion 8](#_Toc153211988)

[Summary of Key Findings and Results 8](#_Toc153211989)

[Reflection on Learning Outcomes and Applicability 8](#_Toc153211990)

# Introduction

## Purpose and Context of the Code

This code is an intricate example of applying algorithmic solutions to a real-world problem in the food manufacturing industry, specifically in biscuit production. The context is set in an environment where biscuits are cut from a dough roll, which may contain various defects. These defects can significantly impact the quality and yield of the final product. Given the competitive nature of the industry, optimizing the use of raw materials and minimizing waste are crucial for efficiency and profitability.

## Main Objective

The principal objective of the code is to optimize the placement of biscuits on a dough roll, taking into account existing defects. This optimization is crucial the overall efficiency of the biscuit production process.

## Overview of the Code Structure

The code is organized into several distinct components, each serving a specific function:

1. Classes and Data Structures: The code defines several classes, namely `Biscuit`, `Defect`, and `DoughRoll`. These classes represent the essential elements of the problem – biscuits with specific attributes, defects in the dough, and the dough roll itself.
2. Heuristic Function: A key part of the code is the heuristic function, designed to guide the search algorithms towards more effective solutions.
3. Optimization Algorithms: The code includes two main algorithms for optimizing biscuit placement – hill climbing search and constraint-based search. These algorithms are central to finding the most effective arrangement of biscuits on the dough roll.
4. Visualization and Progress Tracking: Although not fully detailed in the code, there are indications of visualization functions and a progress tracking mechanism. These are essential for monitoring the performance of the algorithms and visualizing the solutions.

# Detailed Analysis of Key Components

Classes and Their Roles

The code employs object-oriented programming principles to model the real-world scenario of biscuit cutting from a dough roll with defects. These are the following classes :

### Biscuit Class

- `length`: Represents the length of a biscuit. This attribute is crucial for determining how a biscuit fits onto the dough roll.

- `value`: A numeric representation of the biscuit's worth. This could be based on various factors like market value, consumer preference, or production costs.

- `defect\_thresholds`: A dictionary that maps defect types to their respective tolerance levels in a biscuit. This attribute is key to determining whether a biscuit can be placed at a certain position on the dough roll without being adversely affected by defects.

- `children\_indices`: A list to store indices of other biscuits that could potentially follow this biscuit in the sequence.

- `position`: The position of the biscuit on the dough roll. This is dynamically calculated during the optimization process.

- Methods:

- `\_\_init\_\_`: Constructor to initialize a biscuit with its properties.

- `add\_child`: Adds the index of a potential subsequent biscuit in the placement sequence.

- Comparison methods (`\_\_lt\_\_`, `\_\_eq\_\_`): Facilitate sorting and comparison of biscuits based on their value.

### Defect Class : Represents the defects in the dough roll.

- `position`: The location of the defect on the dough roll. This is crucial for determining if a biscuit will overlap with a defect when placed.

- `defect\_class`: A classification or type of defect, which could be used to determine its severity or impact on biscuit quality.

- Method:

- `\_\_init\_\_`: Constructor to initialize a defect with its position and class.

- `\_\_str\_\_`: A method to return a string representation of the defect, which is useful for debugging and visualization purposes.

### DoughRoll Class : Acts as the base for biscuit placement and optimization.

- `length`: The total length of the dough roll, which is a critical factor in the optimization process as it sets the boundary for biscuit placement.

- `defects`: A list of `Defect` objects representing the defects present in the dough roll.

- Method:

- `\_\_init\_\_`: Constructor to initialize the dough roll with its length and defects.

Each of these classes serves to encapsulate and manage the data and behaviors associated with their respective real-world entities. The `Biscuit` class models the product being produced, the `Defect` class models the impediments to production, and the `DoughRoll` class provides the context in which the production takes place. Together, they form the backbone of the code’s simulation of the biscuit production optimization process.

# Hill Climbing Search

## Description of the Hill Climbing Search Algorithm

Hill climbing is a mathematical optimization technique which belongs to the family of local search algorithms. It is often used for solving computational problems where a maximum or minimum solution is sought in a large search space. The essence of the hill climbing algorithm is to start with an arbitrary solution to a problem and then iteratively make small changes to the solution, each time improving it a bit more, much like climbing a hill step by step.

In the context of optimizing biscuit placement based on dough roll defects, the hill climbing search algorithm operates as follows:

* Initial Solution:

The algorithm begins with an initial solution. This could be a random arrangement of biscuits on the dough roll, respecting the constraints such as the dough roll length and defect positions.

* Evaluation:

It evaluates the current solution based on a predefined metric, like the total value of biscuits placed on the dough roll without overlapping defects.

* Neighbor Solutions: It then generates "neighbor" solutions, which are slight modifications of the current solution. In this scenario, a neighbor solution might involve moving a biscuit, adding a new biscuit, or replacing one biscuit with another.
* Selection of Best Neighbor: Each neighbor solution is evaluated, and the best one (i.e., the one with the highest total value) is selected. If this neighbor solution is better than the current solution, it becomes the new current solution.
* Iteration: This process repeats, continually making small adjustments to the solution and climbing towards a local maximum of the objective function (total biscuit value).

## Iteration and Improvement of Solutions

The hill climbing algorithm iteratively improves upon the current solution as follows:

* Continuous Improvement: At each step, the algorithm only accepts a new solution if it's an improvement over the current one. This ensures that the solution quality never degrades from one iteration to the next.
* Local Maximum: The process continues until no neighbor solutions are better than the current one, meaning a local maximum has been reached. This is the point where the algorithm terminates, as further iterations would not yield a better solution.
* Dependence on Initial Solution: The final solution is heavily dependent on the initial solution. If the initial solution is poorly chosen, the algorithm might converge to a suboptimal local maximum.
* Step Size: The nature of the neighbor solutions (i.e., how different they are from the current solution) can greatly affect the algorithm’s performance. Smaller changes might lead to a more thorough exploration of the solution space but can also increase the computation time.
* Balancing Exploration and Exploitation: The algorithm mainly focuses on exploitation (improving the current solution) rather than exploration (searching for different solutions). This characteristic makes hill climbing fast but also prone to getting stuck in local optima.

Hill climbing is a straightforward yet powerful approach for finding optimized solutions in complex problems like biscuit placement. Its effectiveness in this scenario depends on how well it can navigate the search space of biscuit arrangements to maximize the total value while avoiding defects on the dough roll.

# Constraint-Based Search

Approach of Constraint-Based Search

Constraint-Based Search is a method used in optimization problems where solutions must adhere to a set of constraints. Unlike algorithms that primarily focus on optimizing a single objective (such as maximizing value or minimizing cost), constraint-based search is particularly adept at handling problems where maintaining feasibility under a set of rules or limitations is crucial.

In the context of the biscuit placement optimization problem, the constraint-based search algorithm operates with the following approach:

* Defining Constraints: The key constraints in this problem include the length of the dough roll (the total length of biscuits placed must not exceed it) and the positions of defects on the dough roll (biscuits must be placed in a way that avoids defects or meets the defect thresholds).
* Generating an Initial Solution: The algorithm starts with an initial solution that respects these constraints. This might be a random arrangement of biscuits, but unlike hill climbing, ensuring that this initial arrangement adheres to all constraints is crucial.
* Searching for Neighbors: The algorithm generates neighbor solutions by altering the current arrangement of biscuits. However, unlike hill climbing, each generated neighbor must strictly adhere to all the defined constraints.
* Evaluating and Selecting Neighbors: Each neighbor is evaluated based on the optimization criteria (e.g., maximizing the total value of biscuits on the roll). The best-performing neighbor that complies with the constraints is selected.
* Iterative Improvement: The algorithm iteratively improves the solution, always ensuring that the constraints are satisfied. The search continues until no better solution can be found that adheres to the constraints.

## Differences from Hill Climbing Search

* Focus on Constraints: The most significant difference is the focus on respecting constraints. While hill climbing may occasionally violate constraints for short-term gains, constraint-based search never allows solutions that break the defined rules.
* Initial Solution Quality: In constraint-based search, the quality of the initial solution is paramount since it must satisfy all constraints from the outset. In contrast, hill climbing can start with a less restrictive initial solution.
* Neighbor Generation: The process of generating neighbor solutions is more complex in constraint-based search. Each neighbor must be a feasible solution under the problem's constraints, which can significantly limit the search space and require more sophisticated methods to generate neighbors.
* Objective Function vs. Constraints: Hill climbing primarily focuses on optimizing an objective function (like maximizing total biscuit value), sometimes at the expense of constraints. In contrast, constraint-based search treats adherence to constraints as equally, if not more, important than the objective function.
* Solution Space Exploration: Constraint-based search may explore the solution space more comprehensively, as it needs to find feasible solutions under strict constraints. This can potentially lead to finding globally optimal solutions, but it also might require more computational resources and time.

Constraint-based search is a method well-suited for problems where constraints play a critical role, and solutions must be viable within a set framework. In the biscuit placement problem, it ensures that all solutions adhere to the dough roll's limitations and defect positions, potentially leading to a more consistent and feasible optimization outcome compared to hill climbing.

# Solution Validation

Outline of the Solution Validation Process

Solution validation is a crucial step in optimization problems, ensuring that the proposed solution not only meets the objective criteria but also adheres to all the constraints and requirements of the problem. In the context of the biscuit placement optimization problem, the validation process includes several key steps:

Constraint Adherence:

- Length Constraint: Check if the total length of the biscuits placed on the dough roll exceeds the roll's length. The solution must use the available space efficiently without crossing the boundary set by the dough roll's length.

- Defect Avoidance: Each biscuit's position is evaluated to ensure it does not overlap with any defect on the dough roll, or if it does, it must comply with the predefined defect thresholds of that biscuit.

- Value Calculation: The total value of the biscuits on the dough roll is calculated. This ensures the solution aligns with the objective of maximizing the value derived from the dough roll.

- Feasibility Check: The solution must be practical and feasible. This means that the placement of biscuits, as suggested by the solution, should be realistically achievable in a real-world production scenario.

- Consistency Verification: Ensure that the solution is consistent in terms of biscuit placement. There should be no unexplained jumps or irregularities in the sequence or positioning of biscuits.

- Error Identification: In case the solution fails any of the validation checks, it’s important to identify and report the specific errors or constraints that were violated. This can be crucial for debugging and refining the algorithm.

Visualization and Progress Tracking

Visualization plays a vital role in understanding and analyzing the solution, especially in complex optimization problems:

* Graphical Representation: Visualizing the placement of biscuits on the dough roll can provide an intuitive understanding of how the solution fits within the constraints and maximizes value.
* Defect Mapping: A visual map showing the defects on the dough roll and how biscuits are aligned relative to these defects can be very insightful. It helps in quickly identifying if any biscuits overlap with defects.
* Solution Comparison: When different algorithms (like hill climbing and constraint-based search) are used, visualization can compare their solutions side-by-side, aiding in assessing which algorithm performs better under given scenarios.

In summary, solution validation ensures that the proposed solution is viable, efficient, and meets all the problem requirements. Visualization and progress tracking, on the other hand, are indispensable tools for understanding, evaluating, and improving the solution and the algorithm's performance.

Detail how defects are loaded from a CSV file and stored in a DataFrame.

Heuristic Function

Explain the heuristic function used for search algorithms and its significance.

# Conclusion

Summary of Key Findings and Results

The code for optimizing biscuit placement on a dough roll with defects presents a multifaceted approach to a complex real-world problem in the food manufacturing sector. Through the implementation of hill climbing and constraint-based search algorithms, the code demonstrates effective strategies for maximizing the value of biscuit production while accommodating constraints like the length of the dough roll and the presence of defects.

Effective Optimization Strategies\*\*: Both hill climbing and constraint-based search algorithms proved effective in optimizing biscuit placement. Hill climbing excelled in incrementally improving solutions, whereas constraint-based search was adept at maintaining adherence to constraints.

Constraint Handling: The constraint-based search algorithm particularly highlighted the importance of respecting operational constraints, ensuring that all solutions are viable and practical.

Balanced Solution Approach\*\*: The combination of these algorithms showcased a balanced approach to optimization, where one focuses on maximizing value, and the other ensures adherence to crucial constraints.

Visualization and Progress Tracking\*\*: The incorporation of visualization and progress tracking tools in the code enhanced the understanding and analysis of the algorithms’ performance and the solutions they generated.

Reflection on Learning Outcomes and Applicability

Algorithmic Understanding: The process of implementing and analyzing these algorithms provided deep insights into optimization techniques and their application in real-world scenarios. It emphasized the significance of choosing the right algorithm based on the specific requirements and constraints of the problem.

Practical Application in Industry: The code serves as a practical application model in the food manufacturing industry, demonstrating how algorithmic thinking can lead to more efficient and cost-effective production processes.

Importance of Constraints in Optimization: A key learning is the importance of respecting operational constraints in optimization problems. This is particularly relevant in industrial settings where disregarding practical limitations can render solutions infeasible.

Adaptability and Scalability: The methodologies applied in the code are adaptable and scalable to other similar optimization problems, whether in manufacturing, logistics, or other domains where resource allocation and efficiency are paramount.

Future Enhancements and Research: The exploration of these algorithms opens avenues for future enhancements, such as integrating machine learning for predictive analytics or exploring more complex constraints and objectives.

In conclusion, the code offers a comprehensive approach to solving an optimization problem, balancing the maximization of objectives with the adherence to practical constraints. The insights gained from this exercise extend beyond the specific application of biscuit production, offering valuable lessons in algorithmic problem-solving and its application in various industrial contexts.